EM シリーズ

組込み Linux OS 搭載

パネルコンピュータ／ティーチングペンダント

ソフトウェア開発マニュアル

株式会社ディ・エム・シー
https://www.dush.co.jp/

17A4A5-00018-11

はじめに

本マニュアルは、EM シリーズ用の Linux アプリケーション開発手順及びその他アプリケーション仕様を記載しています。

以下の型式の製品が対象になります。

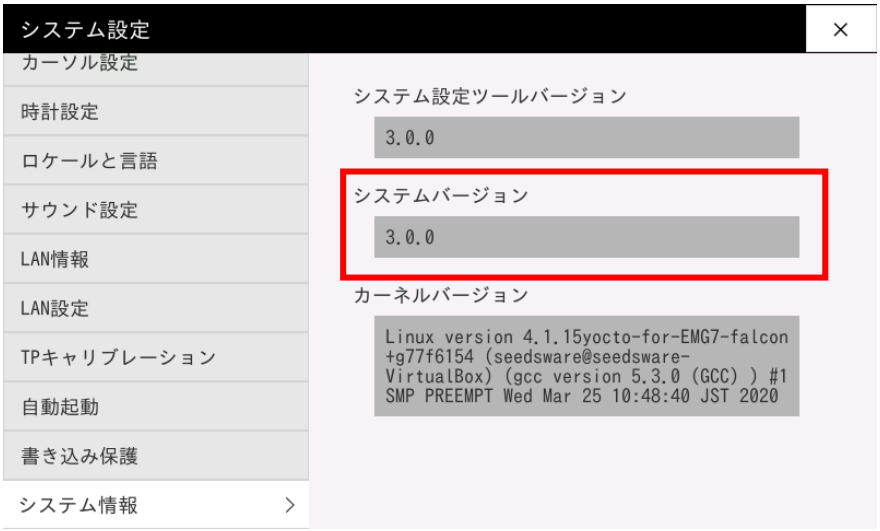
[型式対応表]

型式	本書では以下のように表示します	
EM8-W104A7-00**-*07	EM(G)8	EM(G)8-4
EMG8-W104A7-00**-*07		
EM8-W104A7-00**-*57		EM(G)8-4-SS ※EM(G)8-4 と記載された項目も対象
EMG8-W104A7-00**-*57		
EM8-205A7-00**-*07		EM(G)8-5
EMG8-205A7-00**-*07		
EM8-205A7-00**-*57		EM(G)8-5-SS ※EM(G)8-5 と記載された項目も対象
EMG8-205A7-00**-*57		
EM8-W207A7-00**-*07		EM(G)8-7W
EMG8-W207A7-00**-*07		
EM8-W207A7-00**-*57		EM(G)8-7W-SS ※EM(G)8-7W と記載された項目も対象
EMG8-W207A7-00**-*57		
EM8-W310A7-00**-*07		EM(G)8-10W
EMG8-W310A7-00**-*07		
EM8-W310A7-00**-*57		EM(G)8-10W-SS ※EM(G)8-10W と記載された項目も対象
EMG8-W310A7-00**-*57		
EMP-W207A7-00**-*07	EMP	EMP-7W
EMP-W207A7-00**-*57		EMP-7W-SS ※EMP-7W と記載された項目も対象
EMG7-W207A8-00**-*07	EMG7	EMG7-7W
EMG7-310A8-00**-*07		EMG7-10
EMG7-312A8-00**-*07		EMG7-12

以下のシステムバージョンの製品が対象になります。

対象システムバージョン	3.0.0 ～

EM シリーズ本体のシステムバージョンは、システム設定ツールから確認下さい。
システム設定ツールについては、別紙「EM シリーズ ツールマニュアル」を参照下さい。



著作権および商標に関する記述

EM(G)8-4-SS / EM(G)8-5-SS / EM(G)8-7W-SS / EM(G)8-10W-SS / EMP-7W-SS

目次

1章　開発環境について

本機（以下、EM と記載）の Linux アプリケーション開発環境の作成方法を記載しています。

1.1　開発環境の構築

1.1.1　必要なもの

機材

項目	備考
PC（Windows、Linux）	Linux 用の SDK をインストールできる環境が必要です。 本マニュアルでは、参考例として WSL2(Windows Subsystem for Linux 2)に SDK をインストールする例を掲載しています。
LAN ケーブル	データ転送に使用

データ

項目	備考
Linux SDK	Linux クロスツールチェイン Rootfs イメージ
USB デバイスドライバ	EM 本体の USB デバイスを USB-Ether として使用する為のドライバ

アプリケーション

項目	備考
ターミナルエミュレータ	コンソール接続に使用。 SSH 接続が可能なもの ※本書では Tera Term4.84 を使用して説明しています。
FTP クライアント	データ転送に使用。 SFTP 接続が可能なもの ※本書では FileZilla 3.9.0.2 を使用して説明しています。

1.1.2　Linux 環境の準備（参考資料）

Linux 環境を準備してください。

本書では参考例として、WSL2 (Windows Subsystem for Linux 2) (以下 WSL と記載)で Ubuntu を用いる場合を記載しています。

※ WSL は Microsoft Corporation、Ubuntu は Canonical Ltd./Ubuntu プロジェクトが開発・提供しています。記載項目以外のご使用方法や利用条件など詳細はお客様にてご確認をお願い致します。

※ 本項目の内容（他の Linux 環境、バージョン、手順）以外でも実施可能です。お客様でご都合の良い方法で環境をご準備ください。

[ご注意]
本項目は参考資料です。ご使用されている PC 環境、Windows バージョン、実施時期等により、同じ手順では実施できない場合がございます。その際はお使いの環境に合わせてご対応ください。

本書では以下の環境で確認しています。

Windows

エディション	Windows 11 Pro
バージョン	24H2

仮想環境(Linux)

WSL	2.4.13.0
Ubuntu	20.04

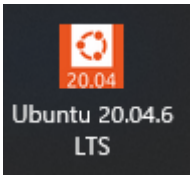
① Windows PowerShell を管理者権限で実行します。

② WSL の有効化と、Ubuntu(v20.04)のインストールを行います。
以下のコマンドを実行してください。

```
> wsl --install -d Ubuntu-20.04
```

③ インストール後、任意のユーザ名とパスワードを設定して起動します。

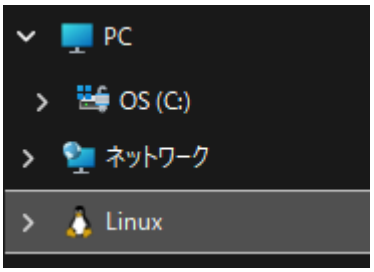
インストール後は、スタートメニューから起動できます。



共有フォルダ

WSL の Linux 環境は以下からアクセス可能です。

エクスプローラーの Linux をクリック



もしくは、アドレス欄に以下のパスを入力してください。

```
¥¥wsl$
```

環境名(Ubuntu-20.04 等)のフォルダから Linux 環境へアクセスできます。

1.1.3　Linux SDK のインストール

準備した Linux 環境に SDK をインストールします。

① DVD-ROM（開発環境一式）より下記ファイル(SDK)を共有フォルダなど利用して、Linux 環境のホームディレクトリなどにコピーします。

[ご注意]
ご使用されている製品に対応した SDK を片方のみインストールしてください。
製品型式との対応は「はじめに」内の「型式対応表」を参照ください。

【EM(G)8 / EMP の場合】

DVD-ROM(開発環境一式)内の「software」-「SDK」-「EM8」フォルダに格納されています。

ファイル名

1.	poky-glibc-x86_64-meta-toolchain-cortexa7hf-neon-toolchain-2.1.2.sh
2.	poky-glibc-x86_64-meta-toolchain-qt5-cortexa7hf-neon-toolchain-2.1.2.sh
3.	poky-glibc-x86_64-em-image-mx6ul-cortexa7hf-neon-toolchain-2.1.2.sh

【EMG7 の場合】

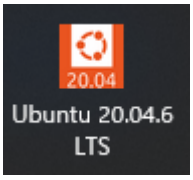
DVD-ROM(開発環境一式)内の「software」-「SDK」-「EM7」フォルダに格納されています。

ファイル名

1.	poky-glibc-x86_64-meta-toolchain-armv7a-neon-toolchain-2.1.2.sh
2.	poky-glibc-x86_64-meta-toolchain-qt5-armv7a-neon-toolchain-2.1.2.sh
3.	poky-glibc-x86_64-em-image-mx53-armv7a-neon-toolchain-2.1.2.sh

② インストールを実行します。インストール作業はLinux環境の端末を使用します。

Linux環境を起動します。WSLはスタートメニューから起動できます。



以下のようにコマンドを実行してください。

SDKをコピーしたディレクトリに移動（例：ホームディレクトリ）

```
$ cd ~
```

実行権限付与

```
$ chmod +x poky-glibc-x86_64*.sh
```

インストール実行（****部分はコピーしたファイルに合わせて変更してください）

```
$ ./poky-glibc-x86_64-meta-toolchain-****-neon-toolchain-2.1.2.sh
```

```
$ ./poky-glibc-x86_64-meta-toolchain-qt5-****-neon-toolchain-2.1.2.sh
```

```
$ ./poky-glibc-x86_64-em-image-****-neon-toolchain-2.1.2.sh
```

それぞれ、画面の指示に従いインストールしてください。
インストール先を指定しなかった場合のデフォルトパスは以下になります。

【デフォルトインストールパス】

```
/opt/poky/2.1.2/
```

1.2　PCとEM本体の接続

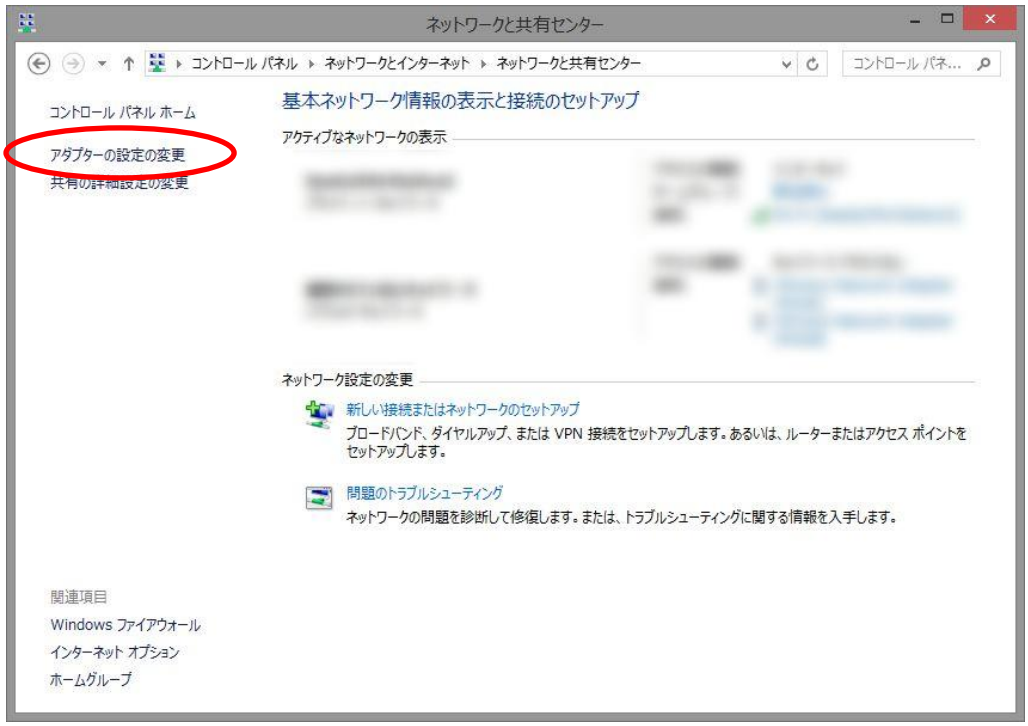
PC と EM をネットワークで接続する為の設定を行います。
LAN ポート、または USB デバイスポートを使用して接続することができます。

1.2.1　　LAN ポートでのネットワーク設定

Windows を操作して、EM と接続できるようにネットワーク設定を変更します。

① コントロールパネルを開いて、「ネットワークの状態とタスクの表示」をクリックして下さい。
※ 表示方法が異なる場合は、右上の「表示方法」を「カテゴリ」に変更して下さい。

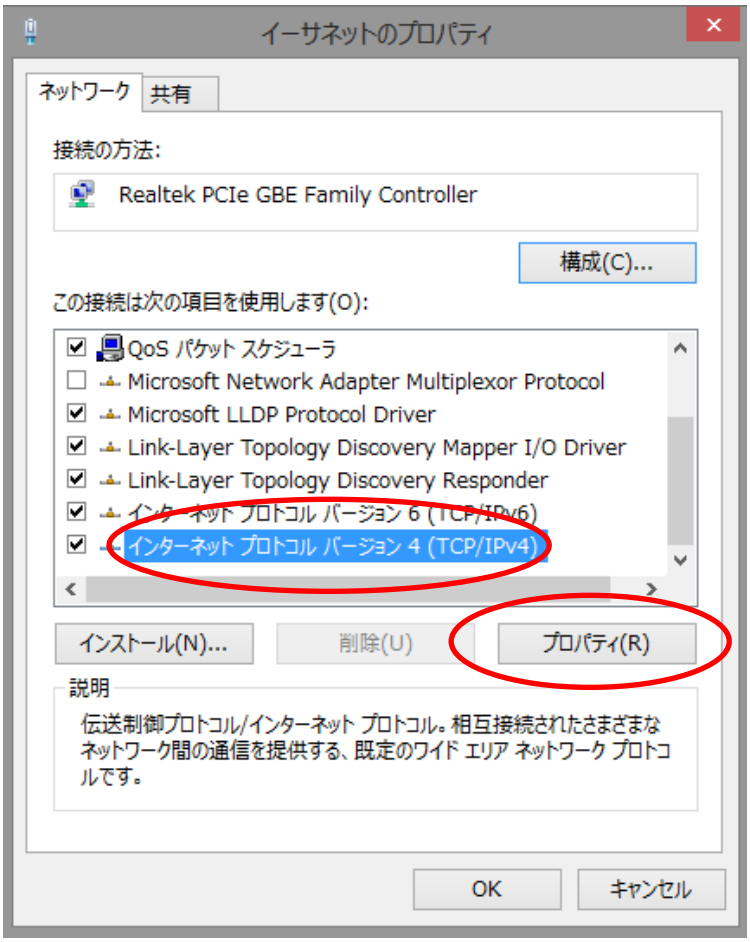
② 「アダプター設定の変更」をクリックして下さい。



③ 「イーサネット」を右クリックして、「プロパティ」をクリックして下さい。

※ 環境によっては、「ローカルエリア接続」など名称が異なる場合があります。有線 LAN ポートのアダプタを選択して下さい。

④　「インターネットプロトコルバージョン 4(TCP/IPv4)」を選択して、「プロパティ」をクリックして下さい。

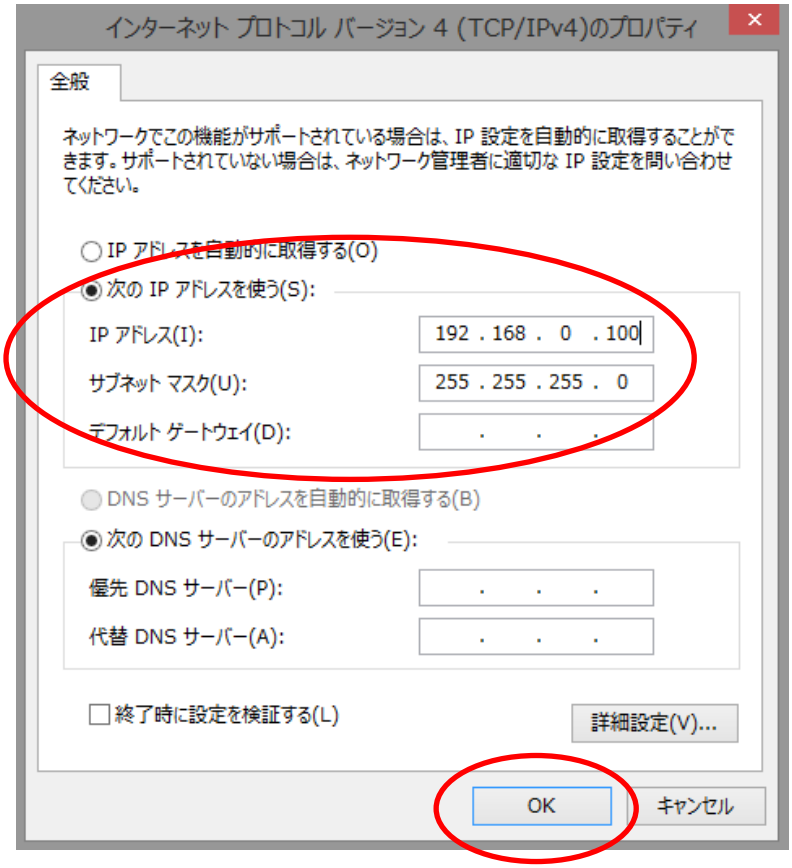


⑤　IP アドレス、サブネットマスク、デフォルトゲートウェイを設定して OK ボタンをクリックして下さい。
ここでは以下に設定します。

IP アドレス	192.168.0.100
サブネットマスク	255.255.255.0
デフォルトゲートウェイ	未設定

※　お使いの環境に合わせて IP アドレス等を変更して下さい。

※　「192.168.10.*」は USB-Ether (usb0) で使用している為、ご使用になれません。



Windows のネットワーク設定が変更されました。

1.2.2　　USB デバイスポートでのネットワーク設定

EM 本体の USB デバイスを USB-Ether として使用する為、PC に USB デバイスドライバのインストールとネットワーク設定を行う必要があります。
本設定を行うと、USB デバイスポートを使用して、EM 本体と PC を LAN 接続することが可能です。

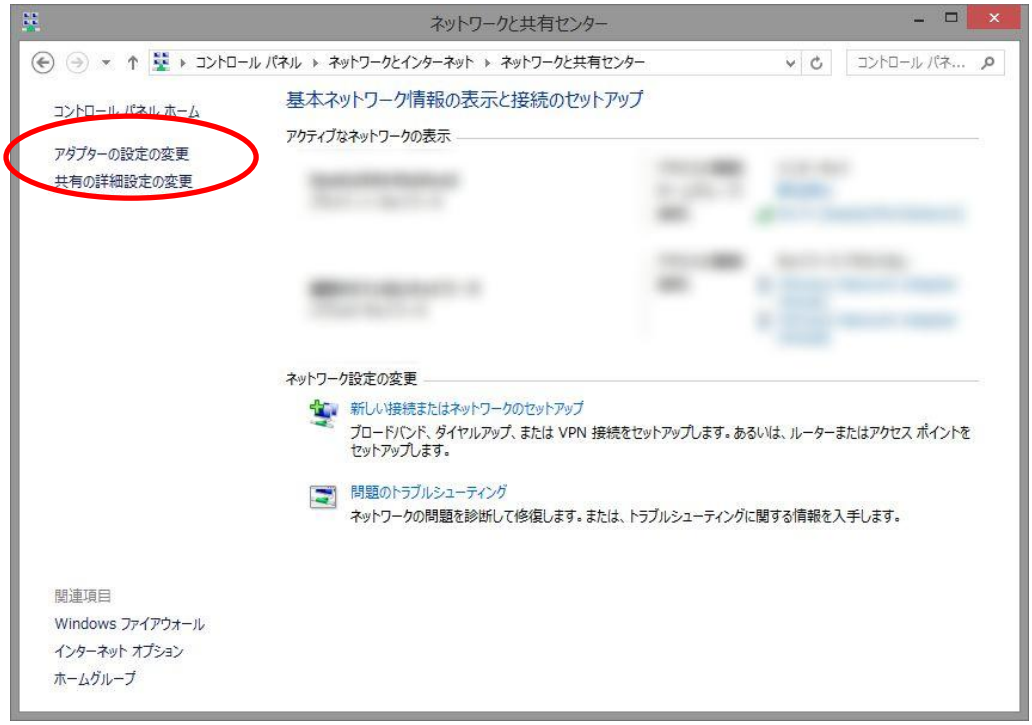
①　PC に USB デバイスドライバをインストールします。

DVD-ROM(開発環境一式)内の「software」-「driver」-「em」フォルダ内の「em_usbd.inf」の右クリックメニューから「インストール」を実施して下さい。

②　PC のコントロールパネルを開いて、「ネットワークの状態とタスクの表示」をクリックして下さい。

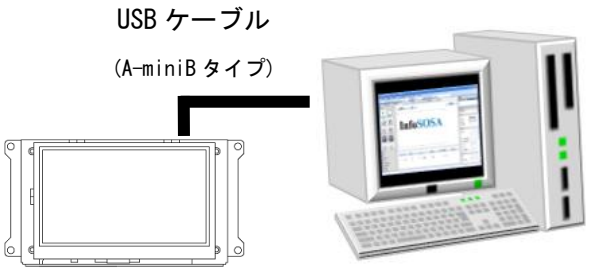
※表示方法が異なる場合は、右上の「表示方法」を「カテゴリ」に変更して下さい。

③　「アダプター設定の変更」をクリックして下さい。



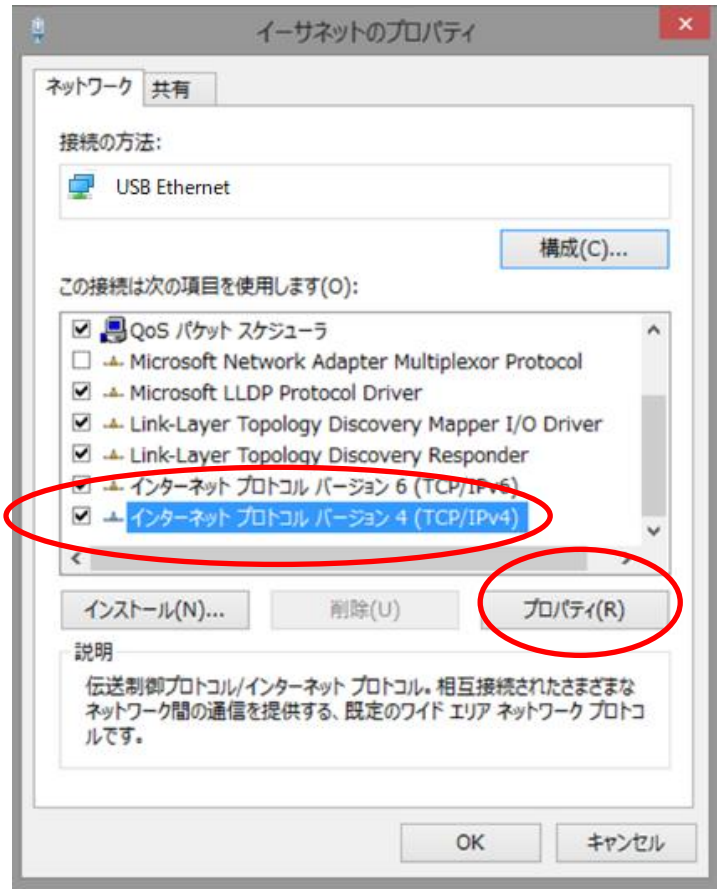
④　「アダプター設定の変更」のウィンドウが開きます。

⑤　EM 本体の電源が ON の状態で PC と USB ケーブル(A-miniB タイプ)で接続します。



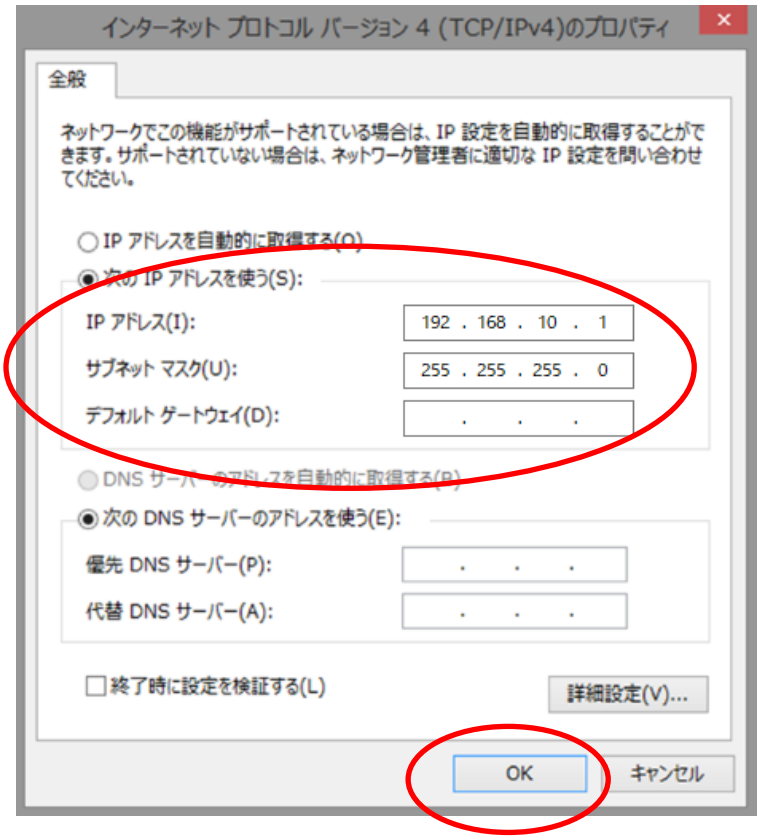
⑥　追加されたアダプターを右クリックして、「プロパティ」をクリックして下さい。

⑦　「インターネットプロトコルバージョン 4(TCP/IPv4)」を選択して、「プロパティ」をクリックして下さい。



⑧　IP アドレス、サブネットマスク、デフォルトゲートウェイを設定して OK ボタンをクリックして下さい。

IP アドレス	192.168.10.1
サブネットマスク	255.255.255.0
デフォルトゲートウェイ	未設定



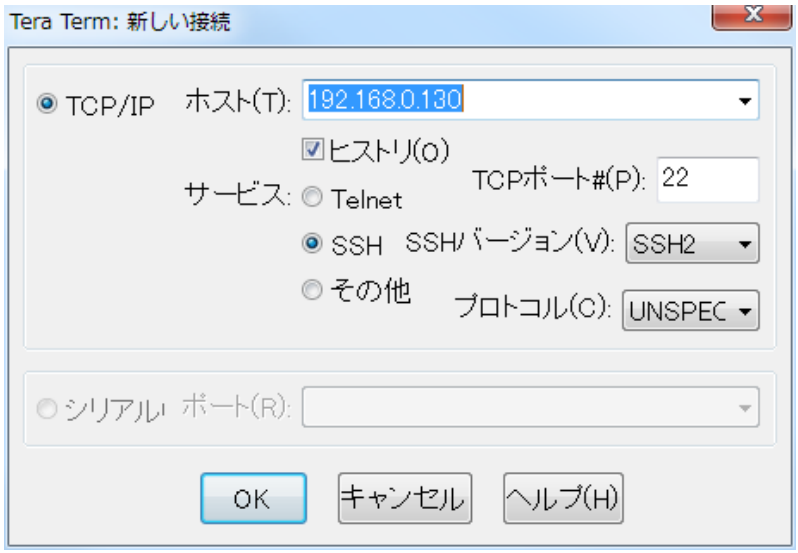
Windows のネットワーク設定が変更されました。

1.2.3　　コンソールの接続方法

SSH プロトコルが使用可能なターミナルエミュレータで EM に接続します。
ここでは Tera Term 4.84 を使用しています。

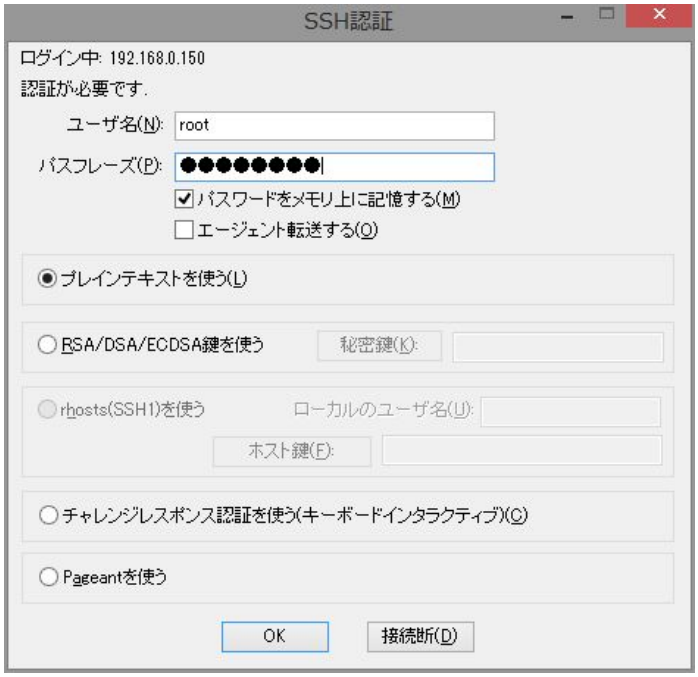
① ターミナルエミュレータを起動して、接続設定を行います。
以下のように設定して下さい。

IP アドレス	LAN ケーブルの場合：192.168.0.130 USB ケーブルの場合：192.168.10.130 ※EM の IP アドレスを変更している場合は合わせて下さい。
ポート	22

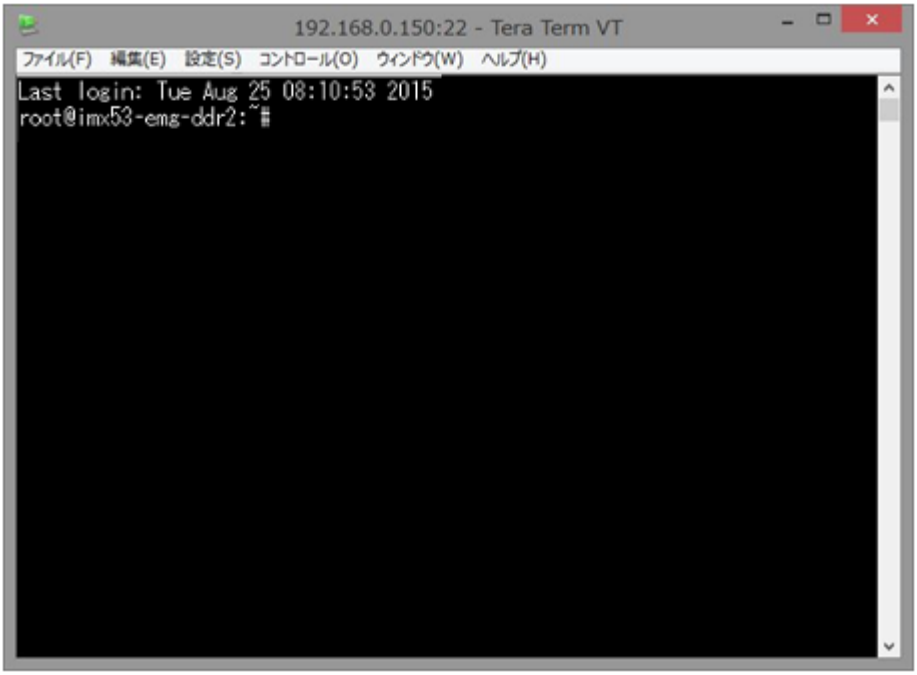


② 以下のユーザでログインして下さい。

ログイン可能ユーザ	ユーザアカウントと同様 ※「1.3 EM ユーザアカウント設定」を参照



接続が完了しました。



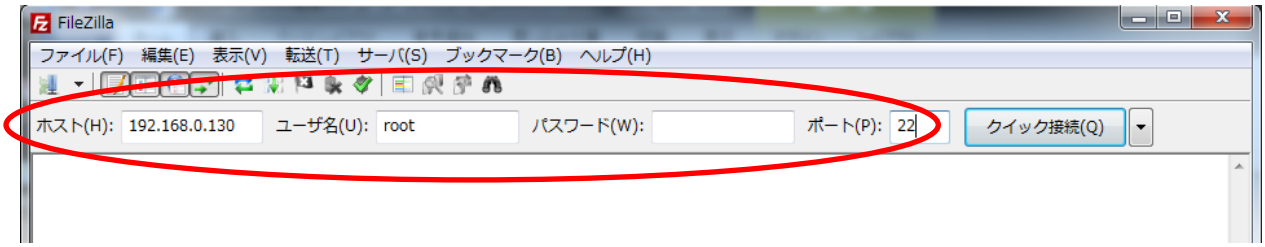
1.2.4　　ファイル転送方法(FTP)

FTP クライアントで EM に転送します。

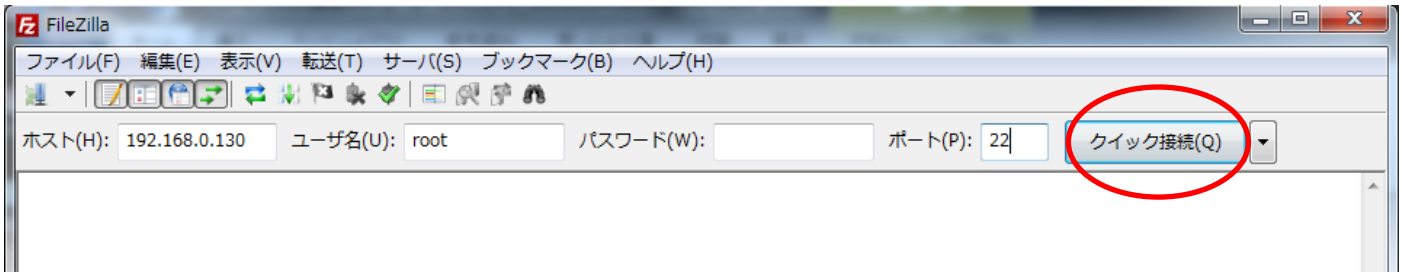
① SFTP で接続可能な FTP クライアントを起動して接続設定を行います。
　ここでは FileZilla 3.9.0.2 を使用しています。

以下のように設定して下さい。

プロトコル	SFTP
IP アドレス	LAN ケーブルの場合：192.168.0.130 USB ケーブルの場合：192.168.10.130 ※EM の IP アドレスを変更している場合は合わせて下さい。
ポート	22
ログイン	ユーザアカウントと同様 ※「1.3 EM ユーザアカウント設定」を参照



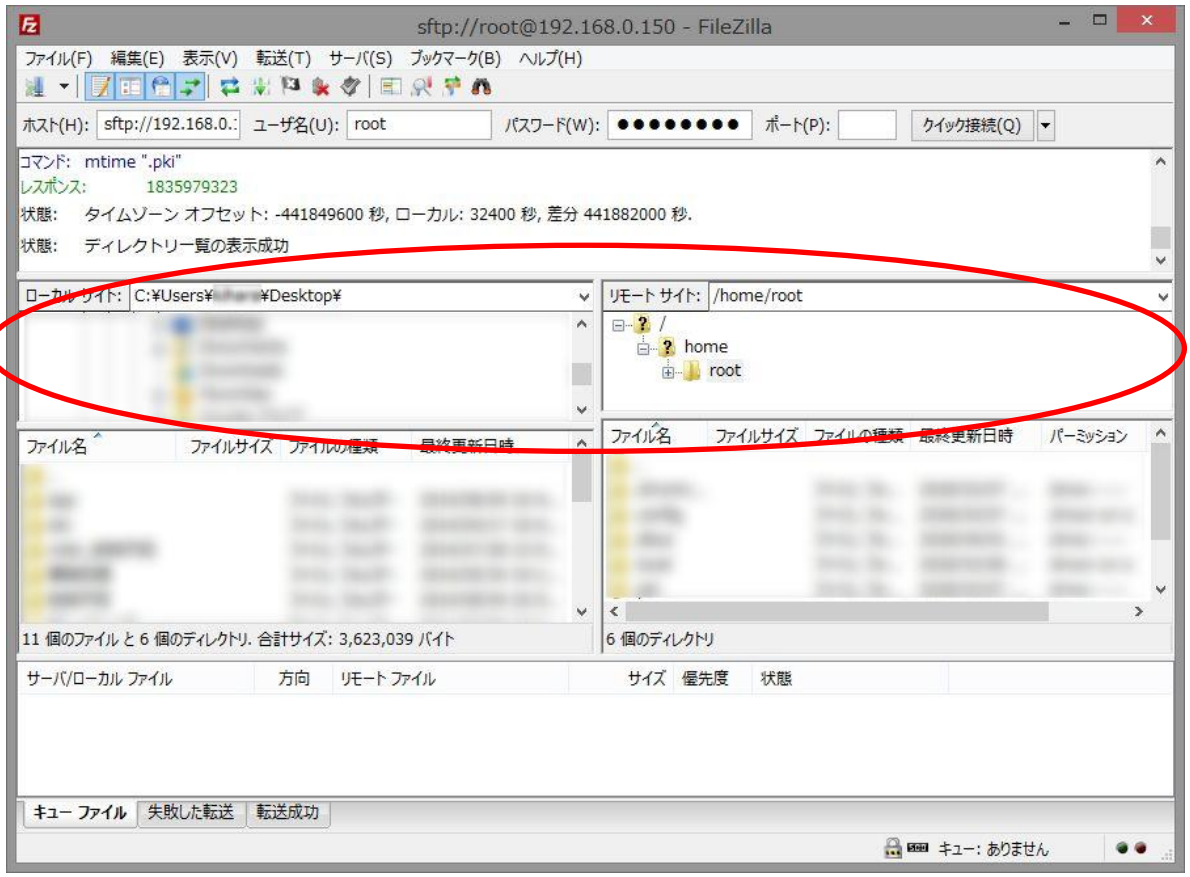
② クイック接続をクリックします。



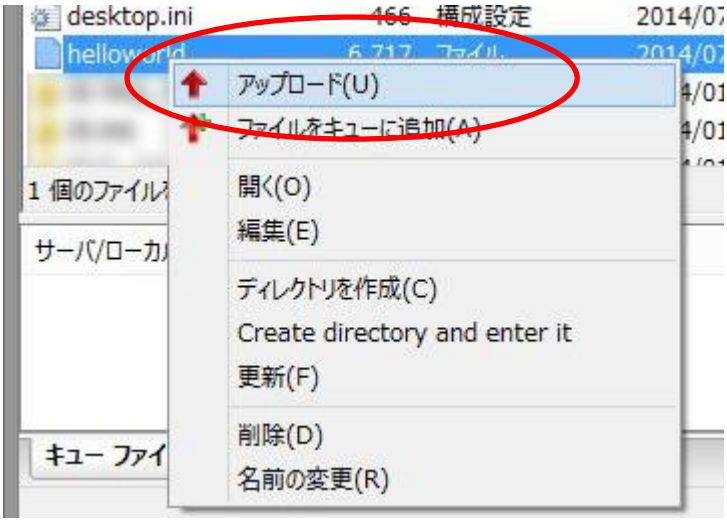
③　ローカルサイトとリモートサイトを設定します。

ここでは以下のように設定します。

ローカルサイト	デスクトップ
リモートサイト	/mnt/user/



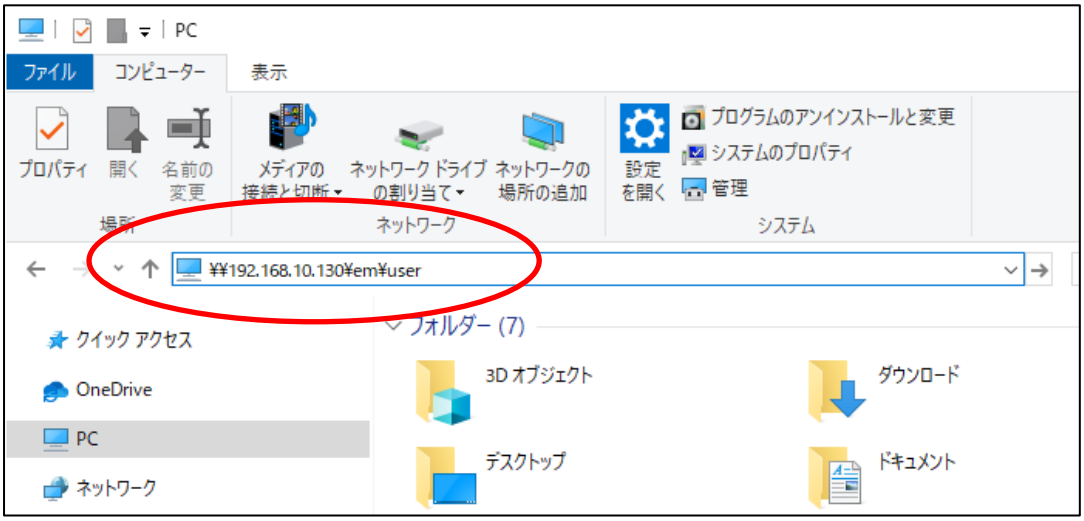
④　転送対象のファイルを右クリックしてメニューを開き、アップロードをクリックして下さい。



/mnt/user/に対象のファイルがコピーされました。

1.2.5　　ファイル転送方法(Samba)

①　エクスプローラーなどで、以下のアドレスにアクセスします。



ユーザフォルダ１	LAN ケーブルの場合：¥¥192.168.0.130¥em¥user USB ケーブルの場合：¥¥192.168.10.130¥em¥user ※EM の IP アドレスを変更している場合は合わせて下さい。
ユーザフォルダ２※	LAN ケーブルの場合：¥¥192.168.0.130¥em¥user2 USB ケーブルの場合：¥¥192.168.10.130¥em¥user2 ※EM の IP アドレスを変更している場合は合わせて下さい。

※ユーザフォルダ２は、製品によっては使用できない場合があります。

②　以下のユーザでログインして下さい。

ログイン可能ユーザ	ユーザアカウントと同様 ※「1.3 EM ユーザアカウント設定」を参照

上記操作で、EM 内のユーザフォルダにアクセスできます。
エクスプローラーでファイルをコピーして下さい。

1.3　EMユーザアカウント設定

ユーザ	ユーザ ID	パスワード
root	root	未設定

デフォルトでは、パスワードは設定されておりません。
下記コマンドにて設定して下さい。
コンソール接続方法については「1.2 PC と EM 本体の接続」を参照下さい。

```
# passwd
root のパスワードを変更しています
新しいパスワードを入れてください（最短 5 文字）
大文字・小文字・数字を混ぜて使うようにしてください。
新しいパスワード: (任意のパスワード)
新規パスワード再入力: (任意のパスワード)
passwd: パスワードは変更されました。
```

もしくは、Smart e-Studio からパスワードを設定することも可能です。
詳しくは、別紙「EM シリーズ Smart e-Studio 取扱説明書」を参照ください。

2章　書き込み保護設定

本機は誤動作や予期せぬトラブルでシステムの破損を防ぐため、ユーザ領域以外は読み取り専用（RO）になっております。読み取り専用フォルダに書き込む場合は一時的に書き込み保護の解除を行う必要があります。

> ⚠ **注意**
>
> **書き込み保護の解除を行い読み取り専用フォルダに書き込んだ後は、速やかに書き込み保護を有効に戻してください。書き込み保護を解除したまま使用された場合、誤動作や予期せぬトラブルでシステムが破損し、動作異常につながる可能性があります。**

2.1　ユーザ領域の書き込み保護領域の設定

ユーザ領域（/mnt/user/、/mnt/user2/）も書き込み保護範囲に含めることが可能です。

2.1.1　　コンソールを接続する

※　接続方法は「1.2 PC と EM 本体の接続」を参照下さい。

2.1.2　　ユーザ領域を書き込み保護範囲に含める

EM のコンソールを操作します。

ユーザ領域を書き込み保護範囲に含める場合は以下のコマンドを実行して下さい。

ユーザ領域 1(mnt/user/)

```
# set_mode_of_user_area1 1
```

ユーザ領域 2(mnt/user2/)

```
# set_mode_of_user_area2 1
```

※ EM(G)8-4-SS / EM(G)8-5-SS / EM(G)8-7W-SS / EM(G)8-10W-SS / EMP-7W-SS の場合、ユーザ領域 2 はありません。

2.1.3　　ユーザ領域を書き込み保護範囲に含めない

EM のコンソールを操作します。

ユーザ領域を書き込み保護範囲に含めない場合は以下のコマンドを実行して下さい。

ユーザ領域 1(mnt/user/)

```
# set_mode_of_user_area1 0
```

ユーザ領域 2(mnt/user2/)

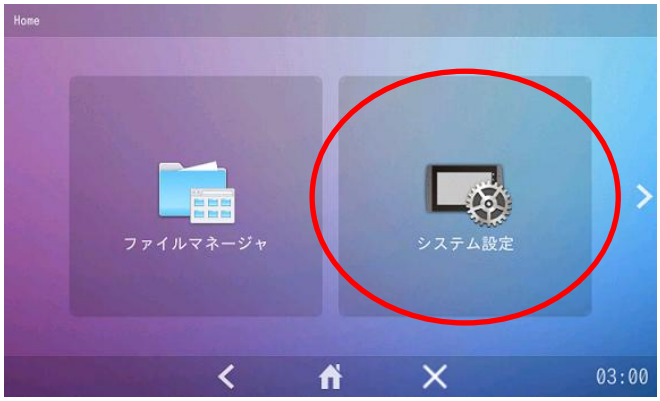
```
# set_mode_of_user_area2 0
```

※ EM(G)8-4-SS / EM(G)8-5-SS / EM(G)8-7W-SS / EM(G)8-10W-SS / EMP-7W-SS の場合、ユーザ領域 2 はありません。

2.2　システム設定ツールでの一時保護解除

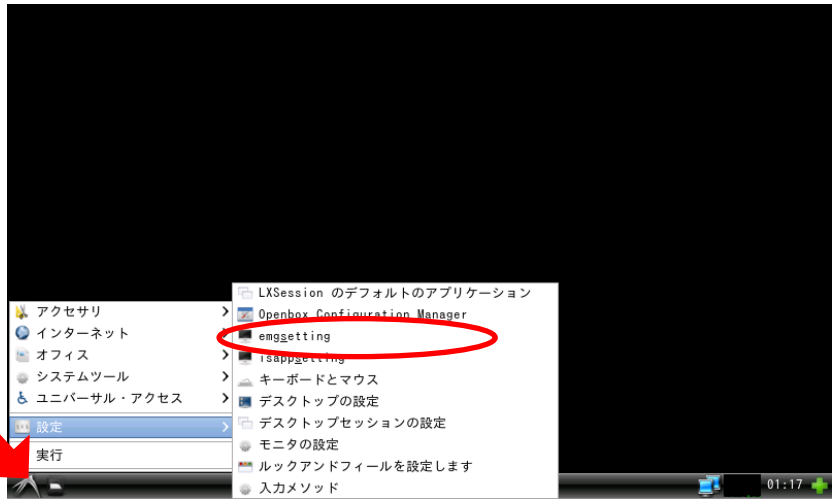
2.2.1　　システム設定ツールを起動する

方法 1）
[EMG ランチャー]の[システム設定]から起動できます。



方法２）

[スタートメニュー]-[設定]-[emgsetting]から起動できます。



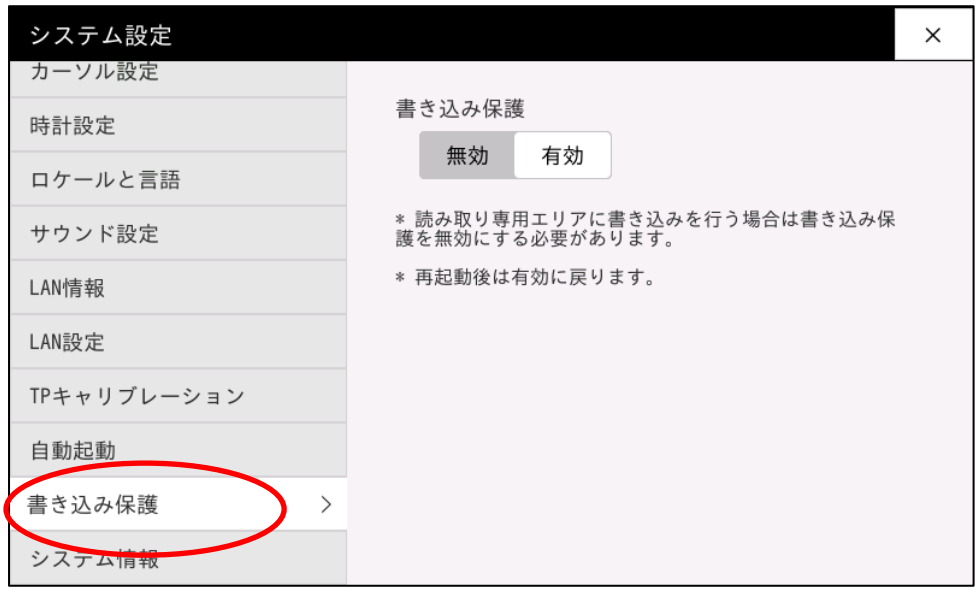
方法３）

以下のプログラムを実行して下さい。

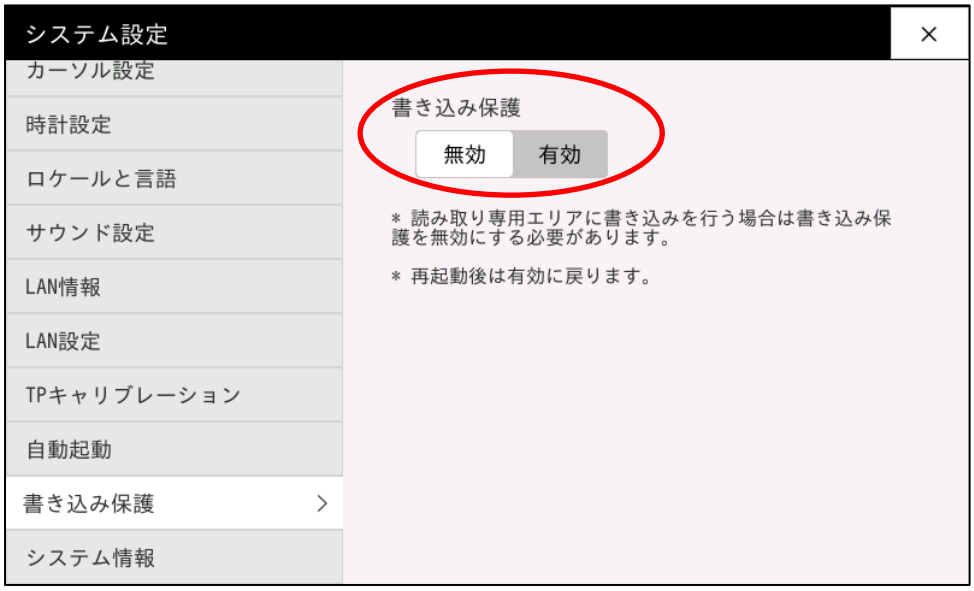
/usr/bin/emg_setting

2.2.2　書き込み保護を無効にする

①　メニューから書き込み保護を選択して下さい。



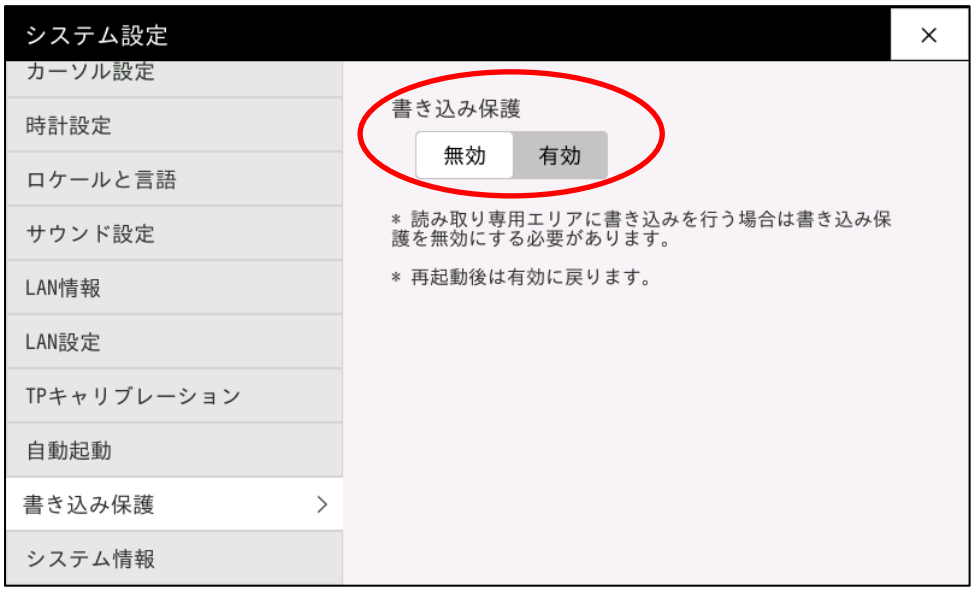
② 書き込み保護を無効にして下さい。



※ユーザ領域を書き込み保護の対象に設定している場合は、ユーザの書き込み保護も無効になります。
※再起動すると有効に戻ります。

2.2.3 書き込み保護を有効にする

再度、書き込み保護をタッチすると有効に戻ります。



※ユーザ領域を書き込み保護の対象に設定している場合は、ユーザの書き込み保護も有効になります。

2.3　コマンドでの一時保護解除

2.3.1　　コンソールを接続する

※　接続方法は「1.2 PC と EM 本体の接続」を参照下さい。

2.3.2　　書き込み保護を無効にする

EM のコンソールを操作します。

書き込み保護を無効にするには以下のコマンドを実行して下さい。

```
# wprotect_off
```

※ユーザ領域を書き込み保護の対象に設定している場合は、ユーザの書き込み保護も無効になります。
※再起動すると読み取り専用に戻ります。

2.3.3　　書き込み保護を有効にする

EM のコンソールを操作します。

書き込み保護を有効にするには以下のコマンドを実行して下さい。

```
# wprotect_on
```

※ユーザ領域を書き込み保護の対象に設定している場合は、ユーザの書き込み保護も有効になります。

3章　アプリケーション開発

EM のコンソール上に”Hello, world!”を表示するアプリケーションの開発方法を記載します。

3.1　VSCode連携（参考資料）

ここでは参考例として、VSCode（Visual Studio Code）を使用した開発方法を記載しています。

※　VSCode は Microsoft Corporation が開発・提供しています。記載項目以外のご使用方法や利用条件など詳細はお客様にてご確認をお願い致します。

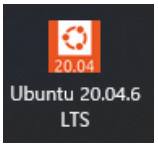
※　本項目の内容（他の開発ツール、バージョン、手順）以外でも実施可能です。お客様でご都合の良い方法で開発環境をご準備ください。

［ご注意］
本項目は参考資料です。ご使用されている PC 環境、Windows バージョン、実施時期等により、同じ手順では実施できない場合がございます。その際はお使いの環境に合わせてご対応ください。

① Windows 環境に VSCode をインストールします。ここではバージョン 1.102.0 を使用しています。

② VSCode に拡張機能「WSL」（識別子：ms-vscode-remote.remote-wsl）をインストールします。

③ Linux 環境を起動します。WSL の場合はスタートメニューから起動できます。



④　Linux 環境で以下のコマンドを実行します。

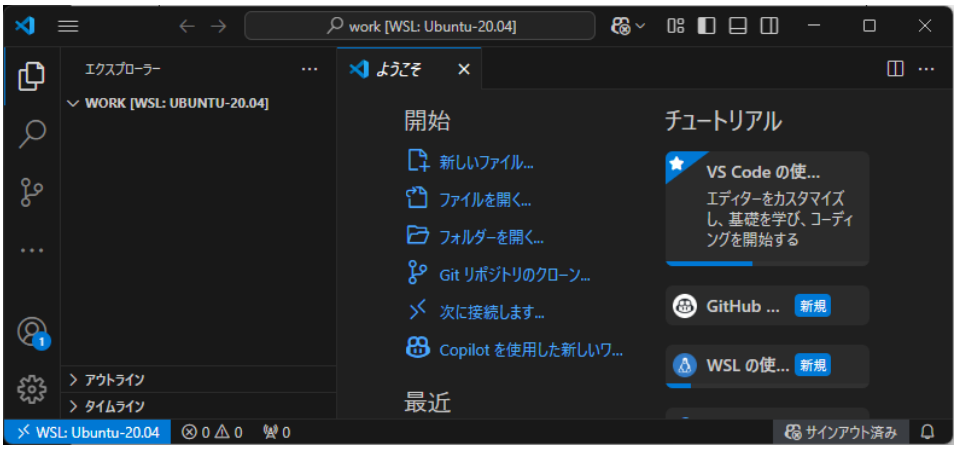
ソースファイルを作成する作業ディレクトリを作成して移動

```
$ mkdir ~/work
$ cd ~/work
```

VSCode を実行

```
$ code .
```

VSCode が起動し、連携が完了します。



次回起動時は Windows 環境で VSCode を起動すると WSL(Linux 環境)も自動的に起動して接続されます。

3.2　ソースファイル作成

開発環境上でソースファイルを作成します。
VSCode と連携している場合は、VSCode を起動します。

① ソースコード作成します。ここでは、以下のように入力します。

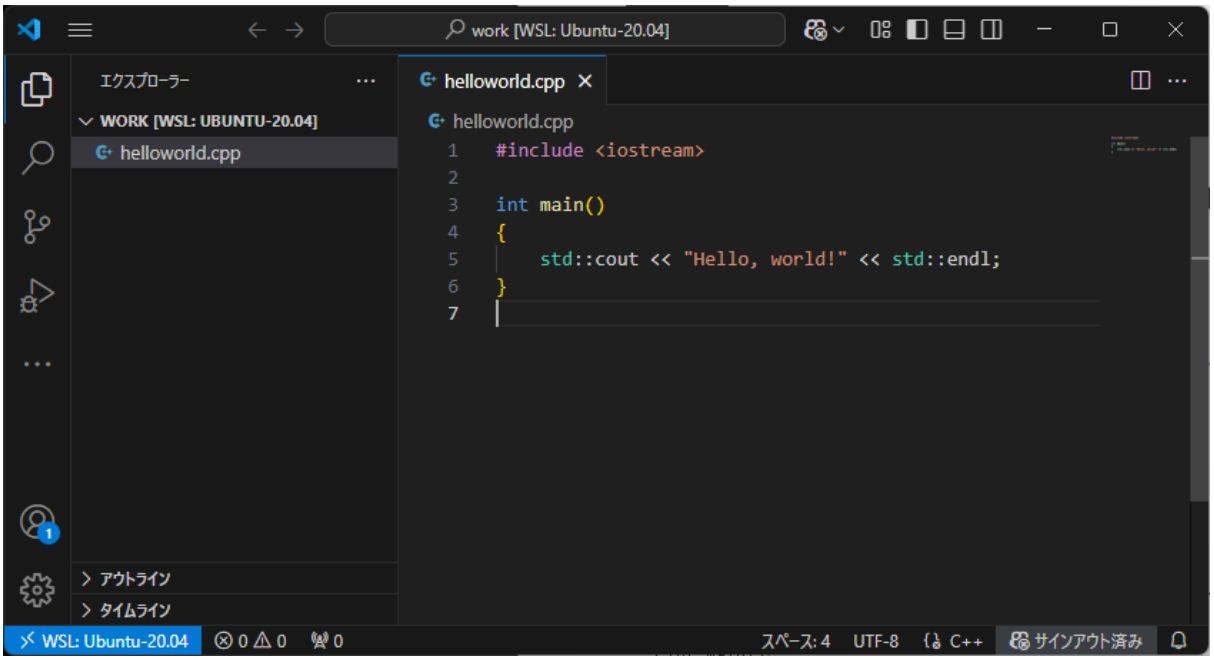
```
#include <iostream>

int main()
{
        std::cout << "Hello, world!" << std::endl;
}
```

② ここでは、以下のように保存します。

保存ディレクトリ	~/work
ファイル名	helloworld.cpp

[VSCode]



3.3　ビルド

作成したソースファイルをビルドします。引き続き開発環境を操作します。

①　ターミナルから以下のようにコマンドを入力して下さい。

ソースファイルを保存したディレクトリに移動

```
$ cd ~/work
```

ビルド環境のセットアップ

※ PATH などの変数の設定を行います。端末起動ごとに実行が必要です。

※ 型式との対応は「はじめに」内の「型式対応表」を参照ください。

【EM(G)8 / EMP の場合】

```
$ source /opt/poky/2.1.2/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
```

【EMG7 の場合】

```
$ source /opt/poky/2.1.2/environment-setup-armv7a-neon-poky-linux-gnueabi
```

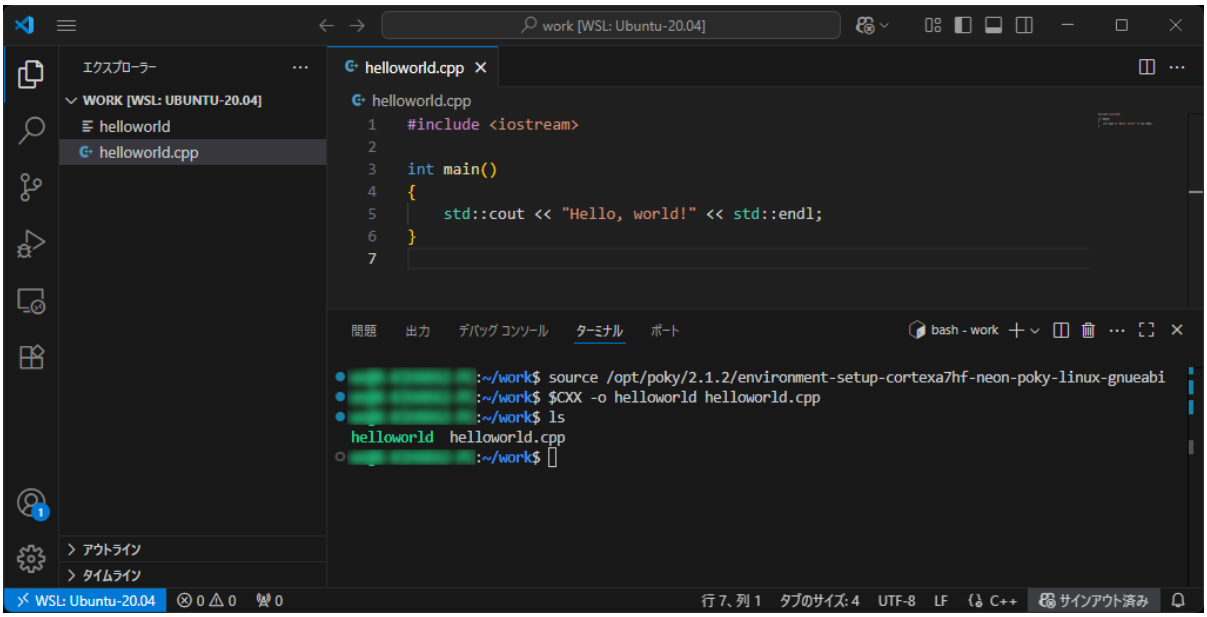
helloworld.cpp のビルド

```
$ $CXX -o helloworld helloworld.cpp
```

以下の実行ファイルが生成されます。

実行ファイル	helloworld

[VSCode]



[オプション]

デフォルトでは、デバッグシンボルが付加された実行ファイルが生成されます。デバッグシンボルが不要な場合は、セットアップスクリプトを以下のように変更して下さい。(-g を削除)

※デバッグシンボルが付加された実行ファイルは通常より大きくなります。

セットアップスクリプト

【EM(G)8 / EMP の場合】

```
/opt/poky/2.1.2/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
```

【EMG7 の場合】

```
/opt/poky/2.1.2/environment-setup-armv7a-neon-poky-linux-gnueabi
```

デバッグシンボル有り

```
export CFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "
export CXXFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "
```

デバッグシンボル無し

```
export CFLAGS=" -O2 -pipe -feliminate-unused-debug-types "
export CXXFLAGS=" -O2 -pipe -feliminate-unused-debug-types "
```

3.4　転送

3.4.1　　コンソールを接続する

※　接続方法は「1.2　PC と EM 本体の接続」を参照下さい。

3.4.2　　実行ファイルを転送する

ビルドした実行ファイルを EM へ転送します。
コンソールを接続した状態で行って下さい。

手順①　ネットワーク設定を行う

※　設定方法は「1.2　PC と EM 本体の接続」を参照下さい。

手順②　Linux 環境から Windows にコピーする

共有フォルダなどを使って、ビルドした実行ファイルを Windows へコピーします。

手順③　実行ファイルを EM に転送する

※　転送方法は「1.2　PC と EM 本体の接続」を参照下さい。

コンソールを操作して、ファイルが転送されたか確認します。

①　転送したディレクトリに移動します。

```
# cd /mnt/user/
```

②　ディレクトリのファイル情報を表示します。

```
# ls
```

転送したディレクトリに実行ファイル(helloworld)がコピーされました。

3.5　実行

転送した実行ファイルを実行します。

EM のコンソールを操作します。

① 実行ファイルを転送したディレクトリに移動します。

```
# cd /mnt/user/
```

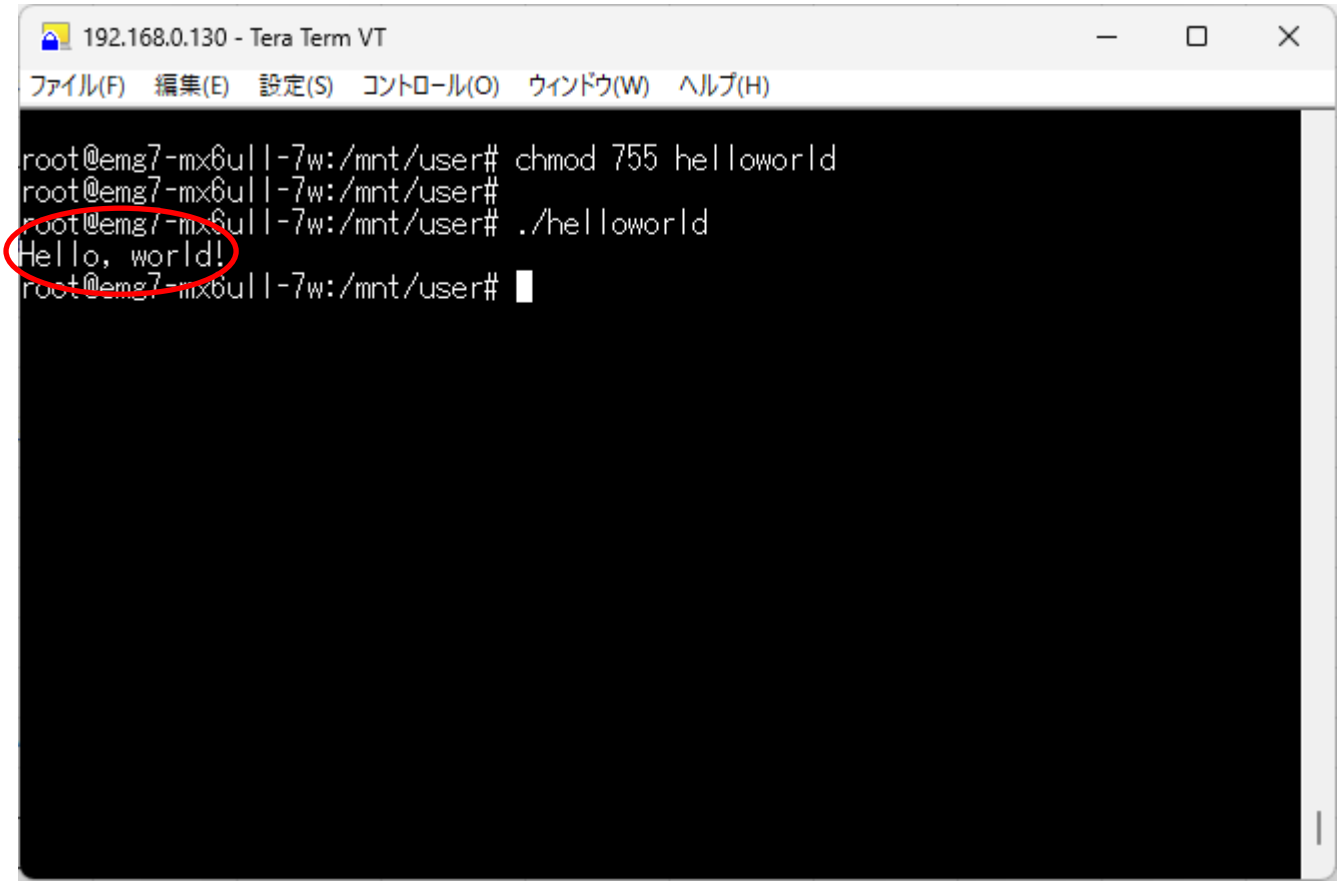
② 実行ファイルのアクセス権を設定します。

```
# chmod 755 helloworld
```

③ 以下のコマンドを入力して、実行します。

```
# ./helloworld
```

コンソールに「Hello, world!」と表示されます。



4章　Qt アプリケーション開発（参考資料）

ここでは参考例として QtCreator を使用して、EM の画面上に”Hello EM”の文字を表示する Qt アプリケーションの開発方法を記載します。

※　Qt/QtCreator は The Qt Company/Qt Project が開発・提供しています。記載項目以外のご使用方法や利用条件など詳細はお客様にてご確認をお願い致します。

［ご注意］
本項目は参考資料です。ご使用されている PC 環境、Windows バージョン、実施時期等により、同じ手順では実施できない場合がございます。その際はお使いの環境に合わせてご対応ください。

4.1　QtCreatorのインストール

ここでは、WSL(Ubuntu20.04)、QtCreator(v4.11.0)を使用した場合の例を記載しています。

Linux 環境に QtCreator をインストールします。

① パッケージリストを更新
以下のコマンドを実行

```
$ sudo apt update
```

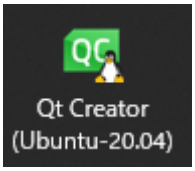
② 続いて QtCreator をインストール

```
$ sudo apt install qtcreator
```

③ QtCreator を実行

```
$ qtcreator &
```

インストール後は、スタートメニューからも起動できます。



4.2　QtCreatorの日本語表示(オプション)

QtCreator の日本語表示方法の一例を記載します。

※　日本語表示は必須ではありません。ご使用の環境によっては、正常に表示されない場合があります。

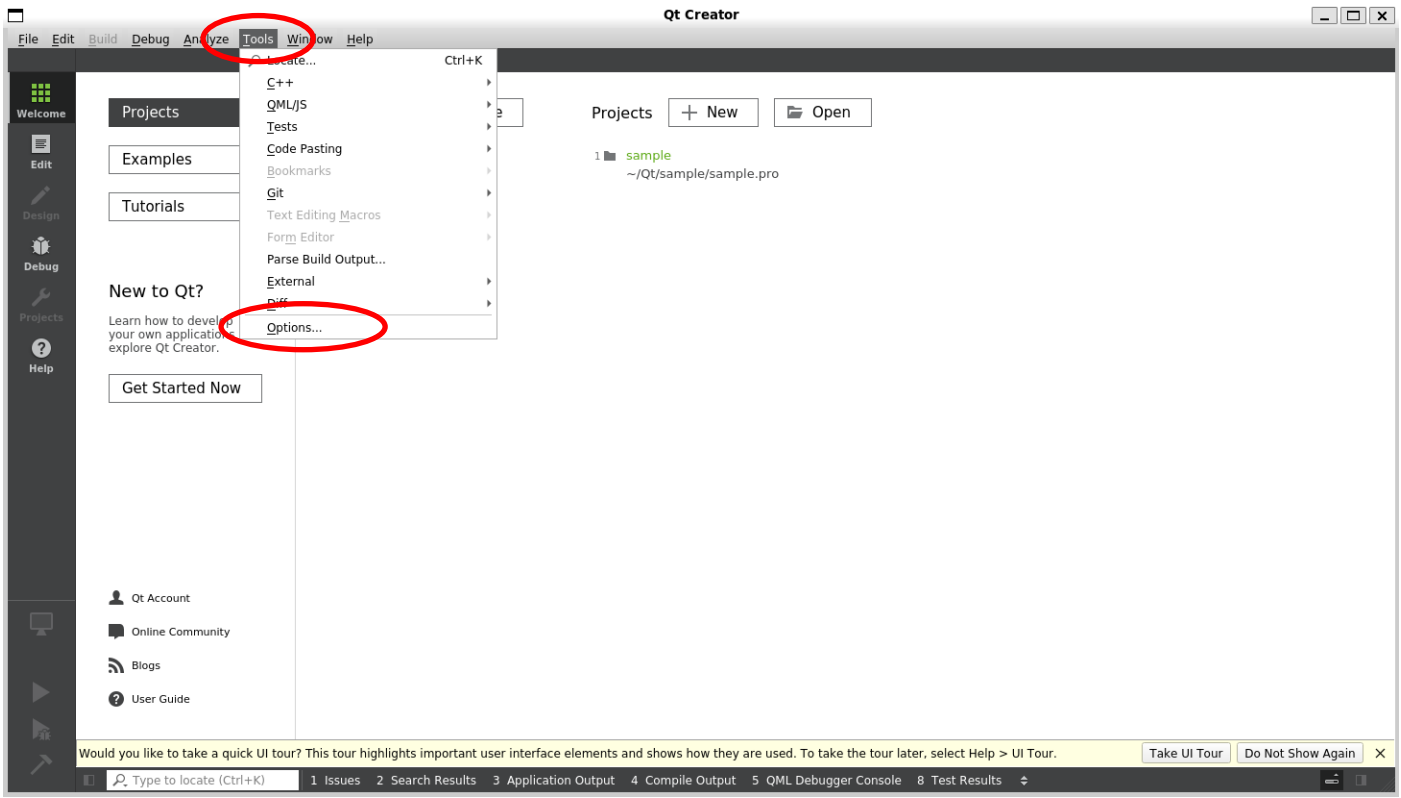
① QtCreator が起動していれば終了し、任意の日本語フォントをインストールします。
ここでは、Noto フォントを導入します。
※ Noto フォントは Google LLC が提供しています。利用条件など詳細はお客様にてご確認をお願い致します。

```
$ sudo apt install fonts-noto-cjk
```

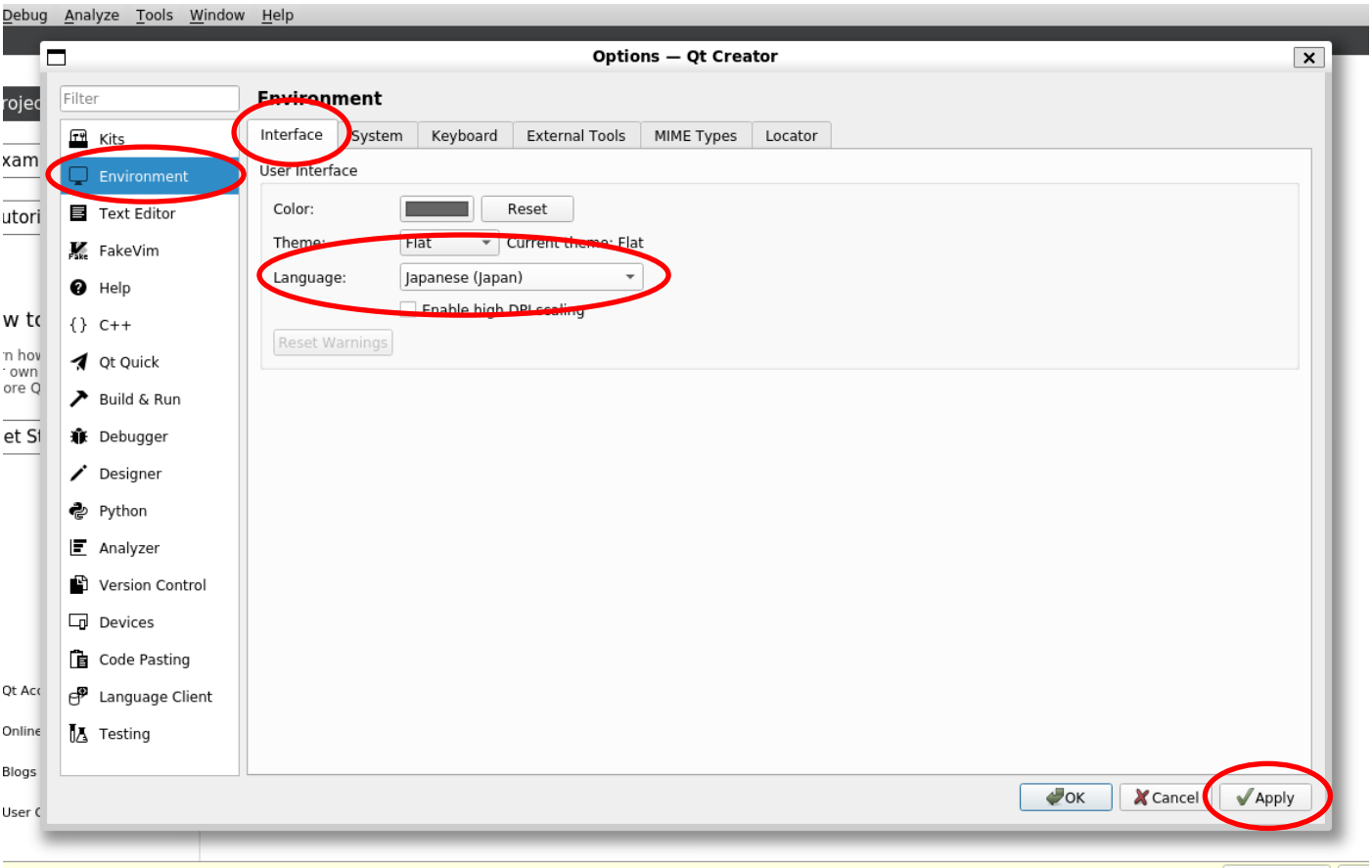
② QtCreator を実行

```
$ qtcreator &
```

③ [tools]→[Options]をクリック



④ [Environment]→[Interface]→[Language]を「Japanese(Japan)」に変更し[Apply]をクリック

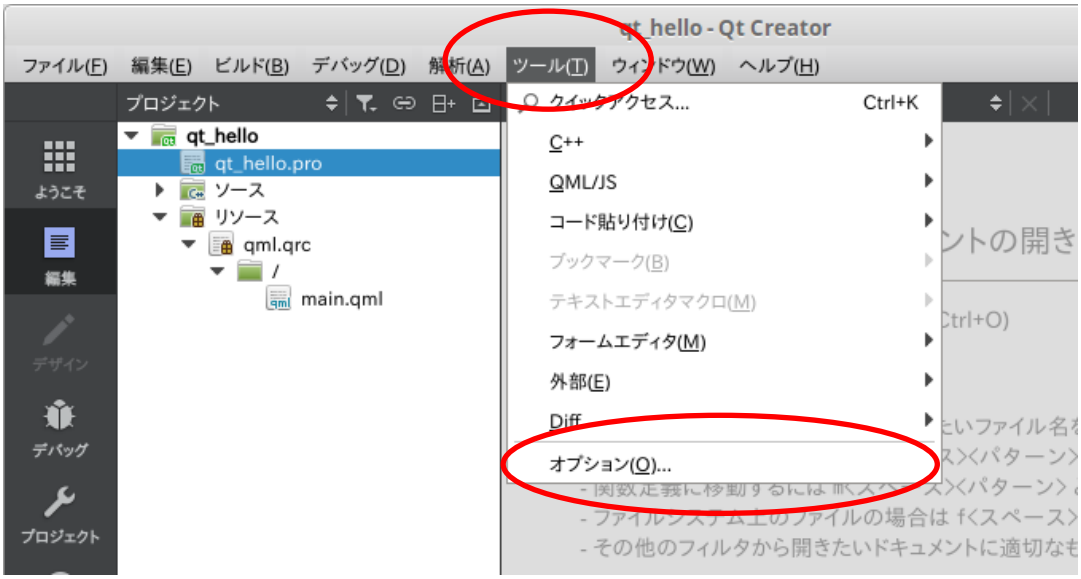


⑤ QtCreator を再起動すると、日本語表示に変更されます。

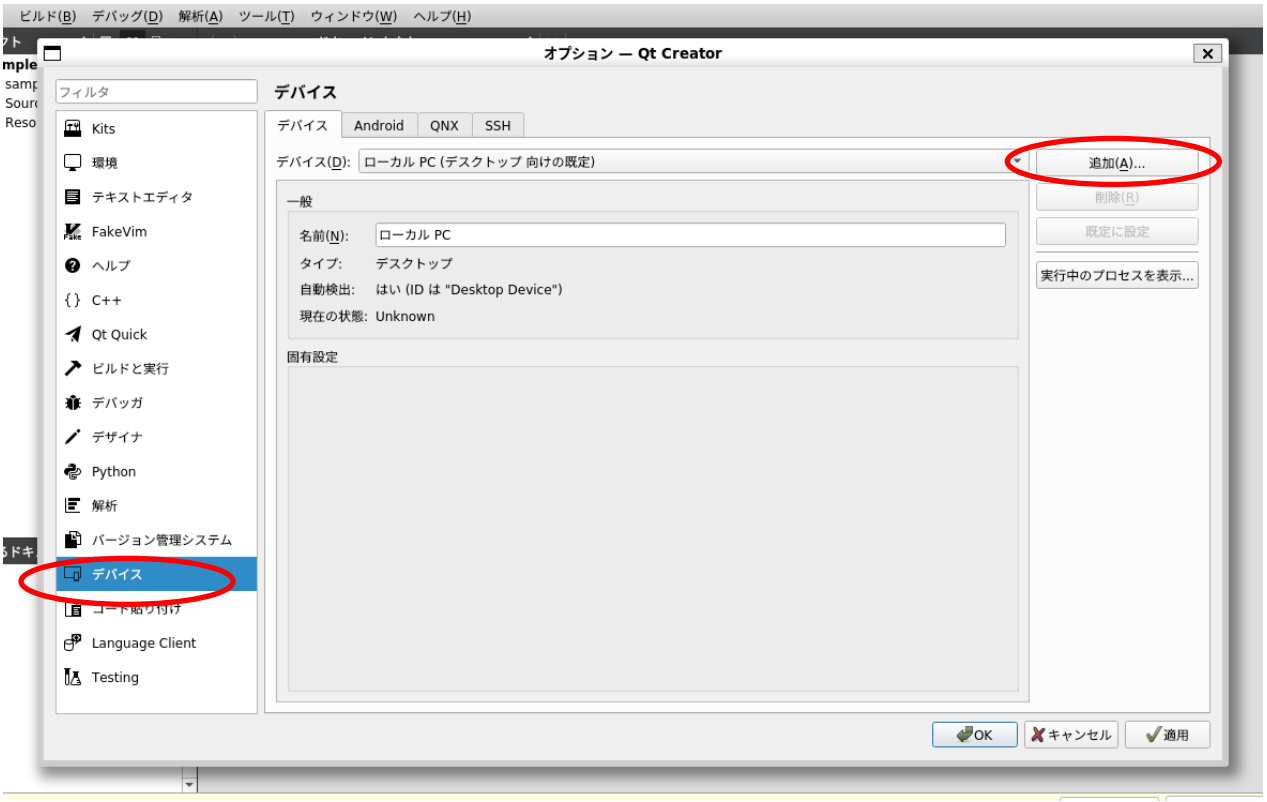
4.3　デバイス設定

QtCreator から EM に接続ができるように、EM の IP アドレスやパスワードを設定します。

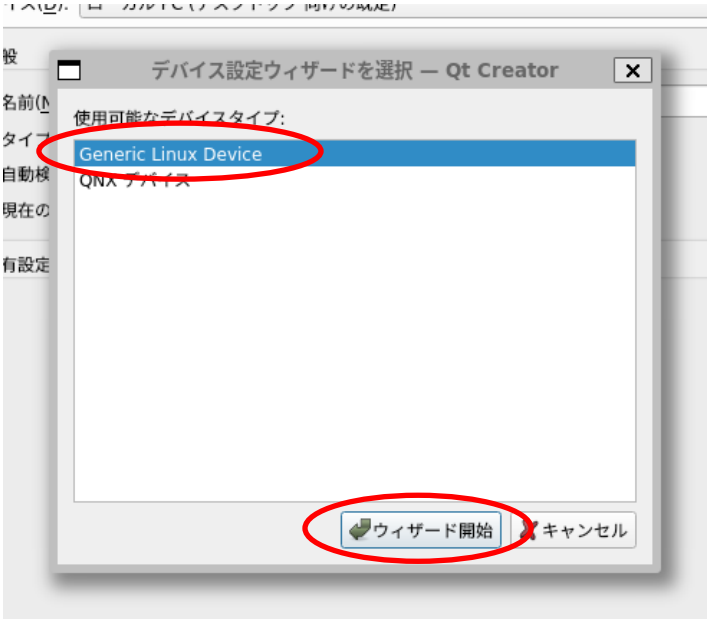
①　上部メニューの[ツール]から[オプション]を選択



②　メニューから「デバイス」を選択し、[追加]をクリック

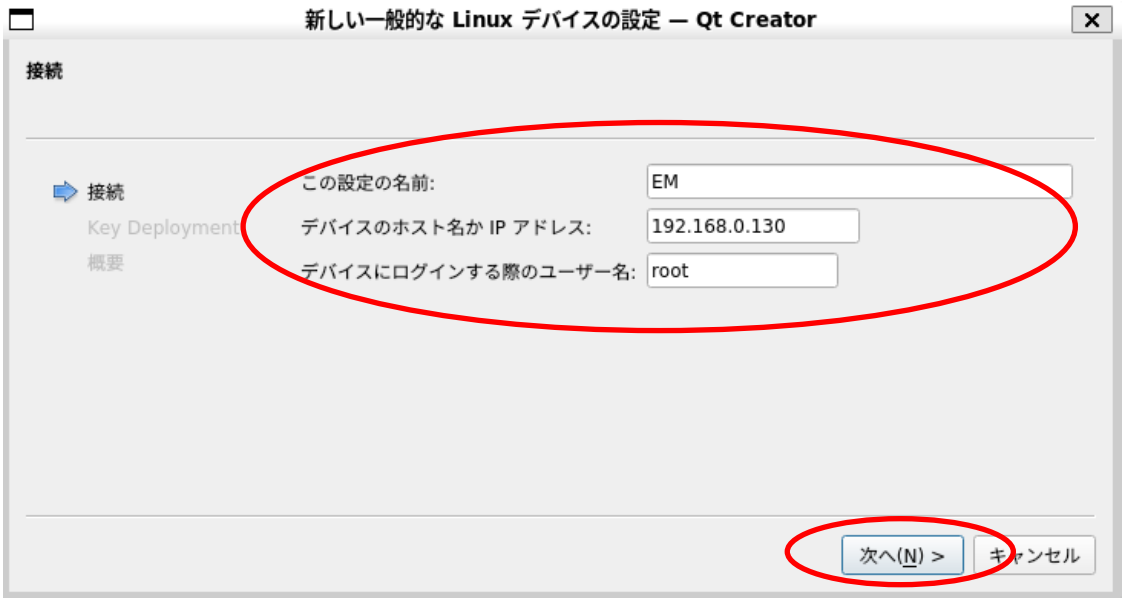


③ [Generic Linux Device]を選択し[ウィザード開始]をクリック



④ [接続]の項目で以下のように設定し、[次へ]

この設定の名前	EM
デバイスのホスト名か IP アドレス	192.168.0.130 ※EM の IP アドレスを変更している場合は合わせて下さい
デバイスにログインする際のユーザー名	root

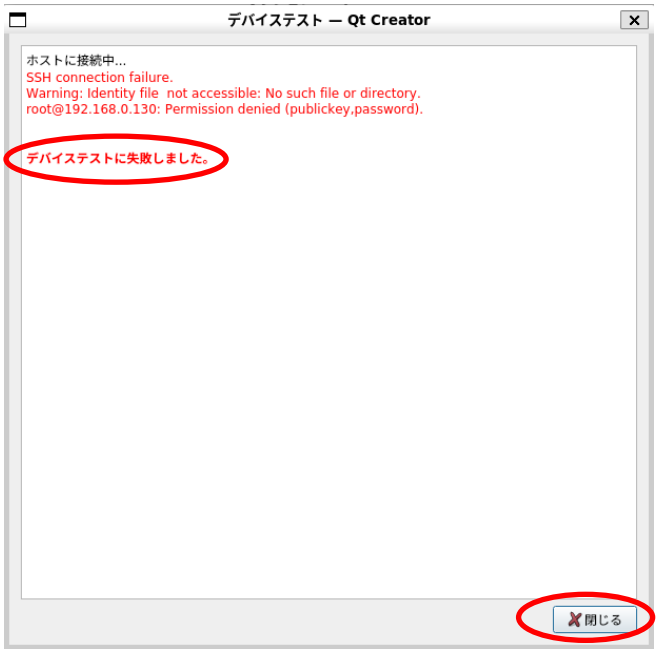


⑤ [Key Deployment]は、今回はパスワード認証を行う為、何も設定せずに[次へ]をクリック

[補足情報]
本手順ではパスワード認証の方法を記載しておりますが、公開鍵認証を設定する場合は、公開鍵をデプロイする前に書き込み保護を解除する必要があります。
解除方法は、「2 章 書き込み保護設定」を参照下さい。

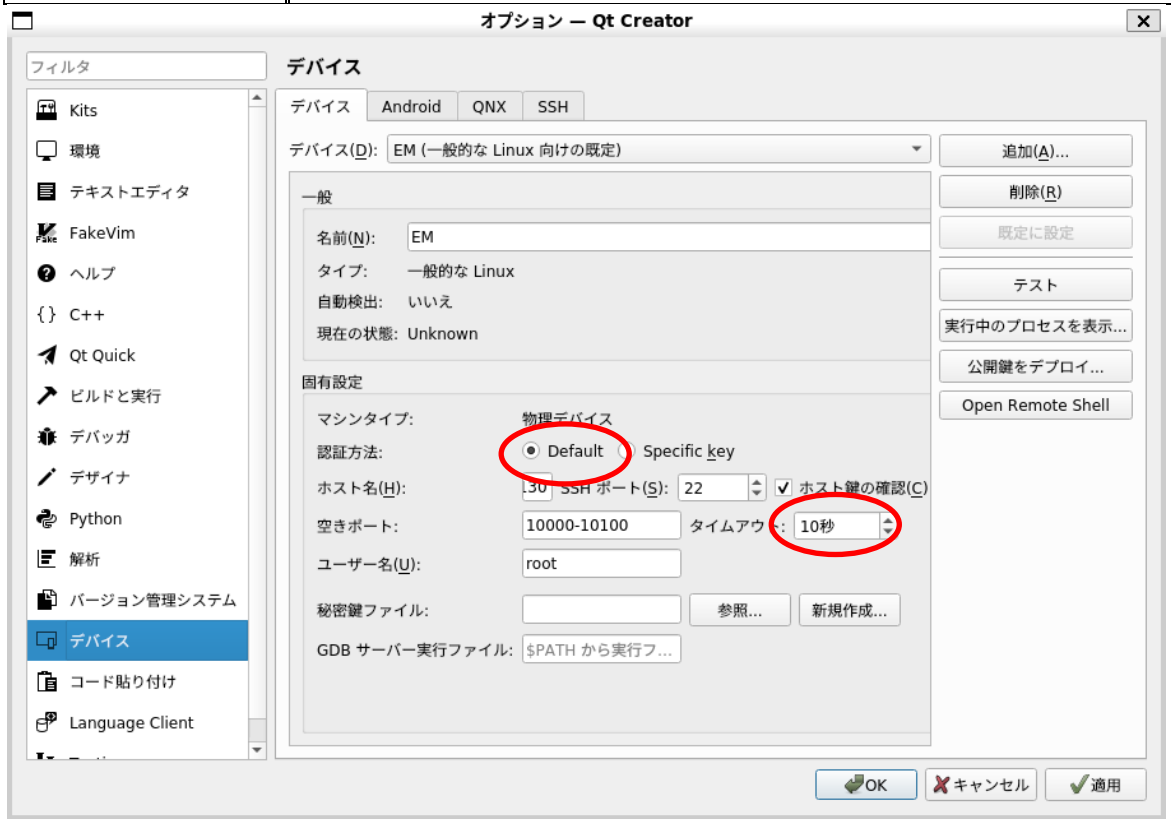
⑥ [概要]画面へ進みましたら[完了]をクリック
デバイステストへと進みます。

⑦ パスワードを設定している場合、テストは失敗します。
デバイステストダイアログを閉じます。



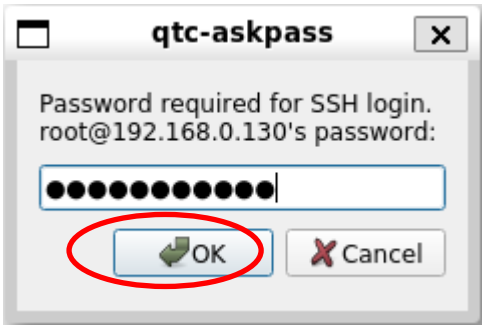
⑧ デバイス設定の画面に戻りましたら[デバイス]に先程作成した[EM]を選択し、以下の内容で設定

認証方法	Default
タイムアウト	10 秒

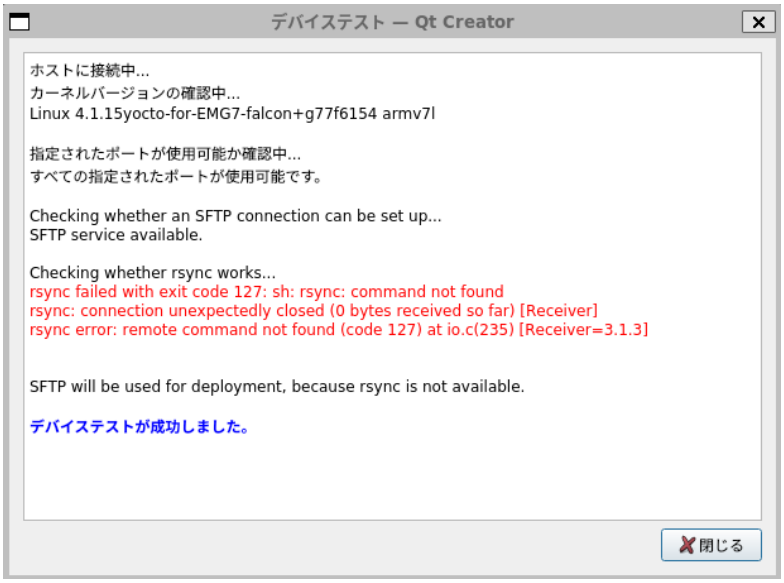


⑨　設定を適用したのち、接続テストを行うため「テスト」をクリック

⑩　パスワードを入力するダイアログが表示されます。
お客様が設定されたパスワードを入力し、[OK]をクリック



⑪　接続が成功した場合、「デバイステストが成功しました」と表示されます。

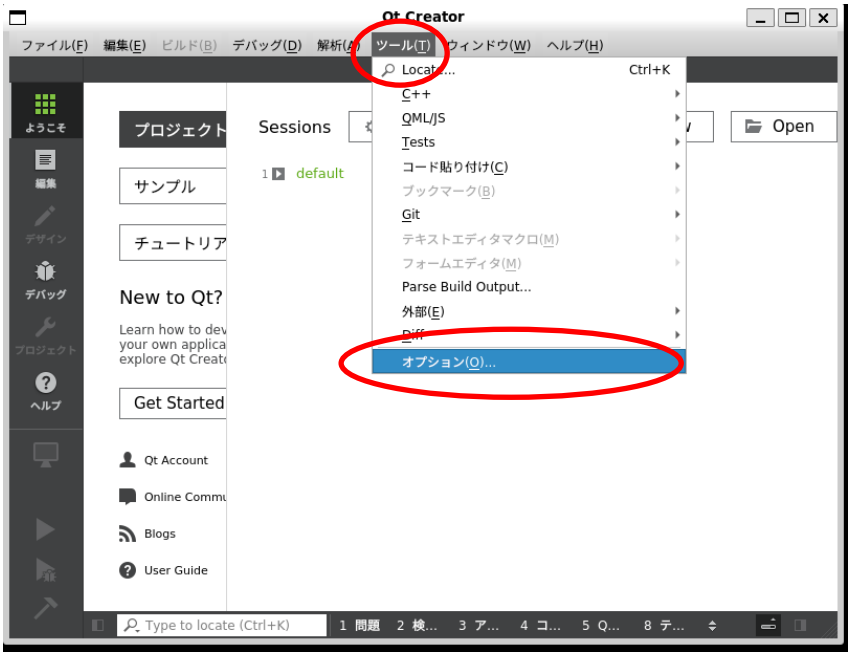


※　赤字で表示されている rsync に関するエラーは問題ありません。
（EM シリーズには rsync が搭載されていない為、SFTP を使用して接続します）

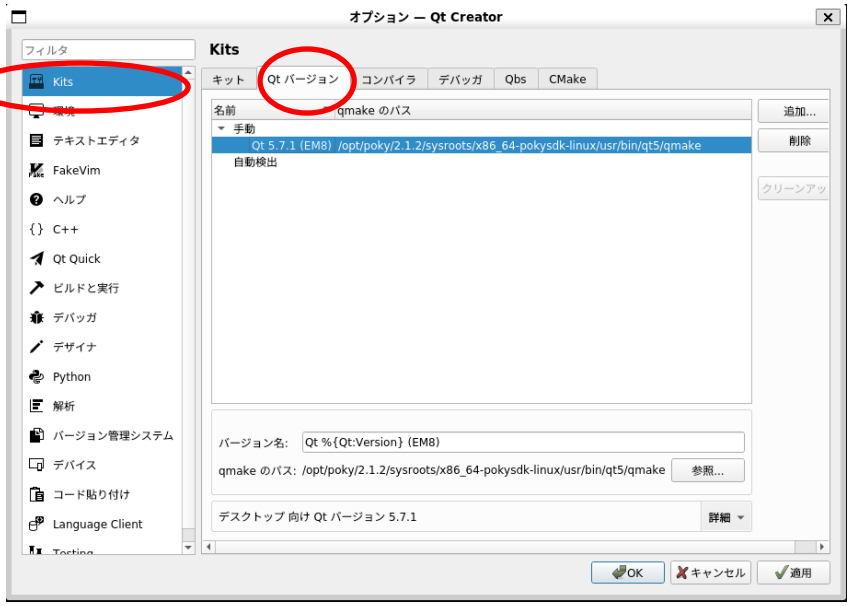
4.4　キット設定

EM 用の実行ファイルを作成する為のキット設定を行います。

① 上部メニューの[ツール]から[オプション]を選択



② メニューから「Kits」を選択し、[Qt バージョン]のタブを表示
以下のように登録します。
※お使いの製品によって登録内容は異なります。「はじめに」内の「型式対応表」を参照ください。



【EM(G)8 / EMP の場合】

バージョン名	Qt %{Qt:Version} (EM8)
qmake のパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake

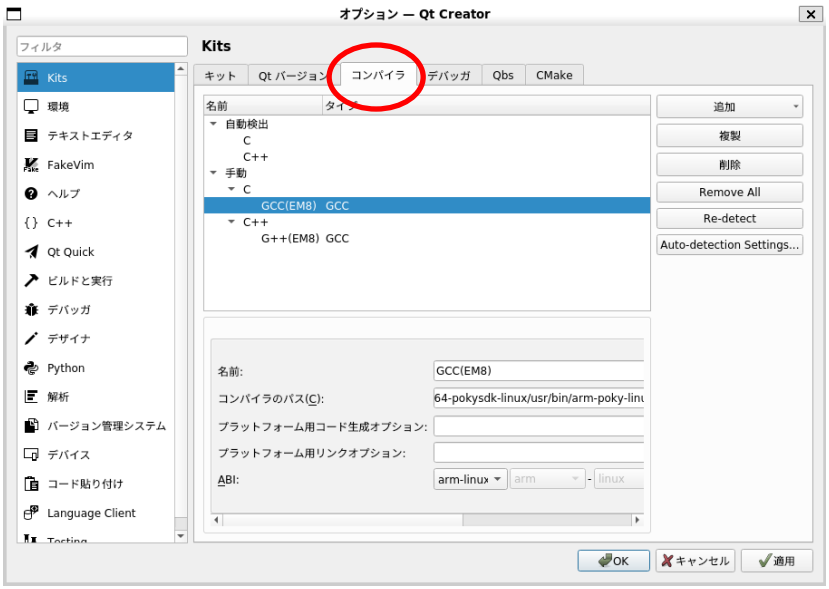
【EMG7 の場合】

バージョン名	Qt %{Qt:Version} (EM7)
qmake のパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake

③ [コンパイラ]のタブを表示

以下のように登録します。

※お使いの製品によって登録内容は異なります。「はじめに」内の「型式対応表」を参照ください。



【EM(G)8 / EMP の場合】

GCC C	名前	GCC(EM8)
	コンパイラのパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc
	ABI	arm-linux-generic-elf-32bit
GCC C++	名前	G++(EM8)
	コンパイラのパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++
	ABI	arm-linux-generic-elf-32bit

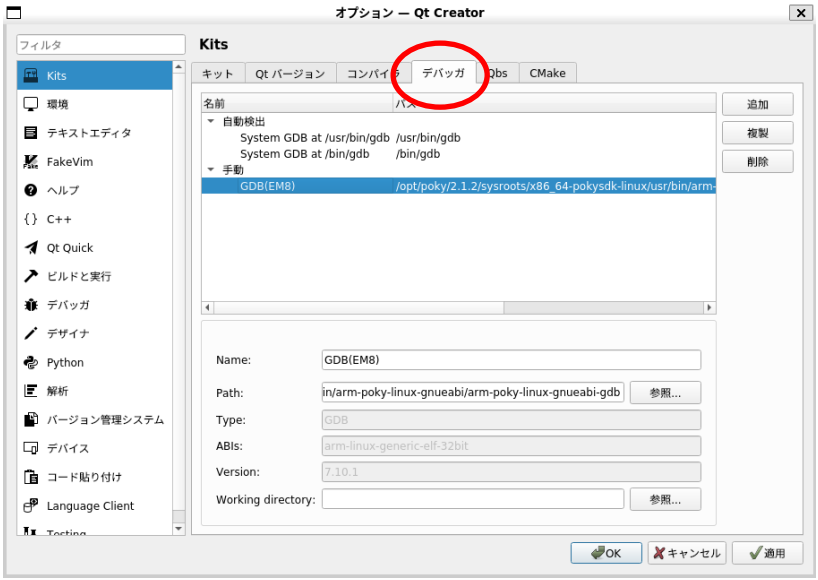
【EMG7 の場合】

GCC C	名前	GCC(EM7)
	コンパイラのパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc
	ABI	arm-linux-generic-elf-32bit
GCC C++	名前	G++(EM7)
	コンパイラのパス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++
	ABI	arm-linux-generic-elf-32bit

④ [デバッガ]のタブを表示

以下のように登録します。

※お使いの製品によって登録内容は異なります。「はじめに」内の「型式対応表」を参照ください。



【EM(G)8 / EMP の場合】

名前	GDB(EM8)
パス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb

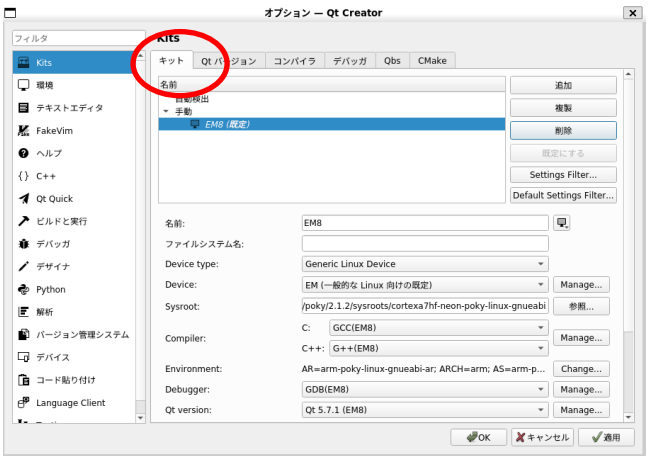
【EMG7 の場合】

名前	GDB(EM7)
パス	/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb

⑤ [キット]のタブを表示

以下のように登録します。

※お使いの製品によって登録内容は異なります。「はじめに」内の「型式対応表」を参照ください。



【EM(G)8 / EMP の場合】

名前	EM8
Device type	Generic Linux Device
Device	EM ※1
sysroot	/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
Compiler (C)	GCC(EM8) ※2
Compiler (C++)	G++(EM8) ※2
Enverment	[change]ボタンを押して、後述の Enverment(EM8) の内容をコピー (56 行)
Debugger	GDB(EM8) ※3
Qt Version	Qt 5.7.1 (EM8) ※4
Qt mkspec	linux-oe-g++
CMake Configuration	[change]ボタンを押して、後述の CMake Configuration (EM8)の内容をコピー (4 行)

※1 「4.3 デバイス設定」で作成したものを選択

※2 [コンパイラ]タブで作成したものを選択

※3 [デバッグ]タブで作成したものを選択

※4 [Qt バージョン]タブで作成したものを選択

CMake Configuration (EM8) 4 行

“XXX=YYYYY”の形式になります。

```
CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}
CMAKE_C_COMPILER:STRING=%{Compiler:Executable:C}
CMAKE_PREFIX_PATH:STRING=%{Qt:QT_INSTALL_PREFIX}
QT_QMAKE_EXECUTABLE:STRING=%{Qt:qmakeExecutable}
```

Enverment(EM8) ※次ページ 56 行

“XXX=YYYYY”の形式になります。

マニュアルでは表示領域の都合上、長い行は折り返されて表示されておりますが、設定時は途中で改行が入らないようにご注意ください。行の始まりは必ず XXX=になります。

```
AR=arm-poky-linux-gnueabi-ar
ARCH=arm
AS=arm-poky-linux-gnueabi-as
CC=arm-poky-linux-gnueabi-gcc  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
CCACHE_PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-
linux/usr/bin/../x86_64-pokysdk-linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-
gnueabi:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-uclibc:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-
linux/usr/bin/arm-poky-linux-musl:
CFLAGS=  -O2 -pipe -g -feliminate-unused-debug-types
CONFIGURE_FLAGS=--target=arm-poky-linux-gnueabi --host=arm-poky-linux-gnueabi --build=x86_64-linux --with-libtool-
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
CONFIG_SITE=/opt/poky/2.1.2/site-config-cortexa7hf-neon-poky-linux-gnueabi
CPP=arm-poky-linux-gnueabi-gcc -E  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
CPPFLAGS=
CROSS_COMPILE=arm-poky-linux-gnueabi-
CXX=arm-poky-linux-gnueabi-g++  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
CXXFLAGS= -O2 -pipe -g -feliminate-unused-debug-types
GDB=arm-poky-linux-gnueabi-gdb
KCFLAGS=--sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
LD=arm-poky-linux-gnueabi-ld  --sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
LDFLAGS=-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
LD_LIBRARY_PATH=/opt/poky:/opt/poky/qtcreator
M4=m4
NM=arm-poky-linux-gnueabi-nm
OBJCOPY=arm-poky-linux-gnueabi-objcopy
OBJDUMP=arm-poky-linux-gnueabi-objdump
OECORE_ACLOCAL_OPTS=-I /opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/share/aclocal
OECORE_DISTRO_VERSION=2.1.2
OECORE_NATIVE_SYSROOT=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux
OECORE_SDK_VERSION=2.1.2
OECORE_TARGET_SYSROOT=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
OE_QMAKE_AR=arm-poky-linux-gnueabi-ar
OE_QMAKE_CC=arm-poky-linux-gnueabi-gcc  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
OE_QMAKE_CFLAGS=  -O2 -pipe -feliminate-unused-debug-types
OE_QMAKE_CXX=arm-poky-linux-gnueabi-g++  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
OE_QMAKE_CXXFLAGS= -O2 -pipe -feliminate-unused-debug-types
OE_QMAKE_INCDIR_QT=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/include/qt5
OE_QMAKE_LDFLAGS=-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
OE_QMAKE_LIBDIR_QT=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib
OE_QMAKE_LINK=arm-poky-linux-gnueabi-g++  -march=armv7ve -marm -mfpu=neon  -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
OE_QMAKE_MOC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/moc
OE_QMAKE_PATH_HOST_BINS=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5
OE_QMAKE_QDBUSCPP2XML=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qdbuscpp2xml
OE_QMAKE_QDBUSXML2CPP=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qdbusxml2cpp
OE_QMAKE_QT_CONFIG=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/qconfig.pri
OE_QMAKE_RCC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/rcc
OE_QMAKE_UIC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/uic
OLDPWD=/opt
PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-
linux/usr/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/sbin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-
linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/sbin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/../x86_64-
pokysdk-linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi:/opt/poky/2.1.2/sysroots/x86_64-
pokysdk-linux/usr/bin/arm-poky-linux-uclibc:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-
musl:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-
linux/usr/bin/qt5:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
PKG_CONFIG_PATH=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/pkgconfig
PKG_CONFIG_SYSROOT_DIR=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
QMAKESPEC=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/linux-oe-g++
QT_CONF_PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qt.conf
QT_IM_MODULE=ibus
RANLIB=arm-poky-linux-gnueabi-ranlib
SDKTARGETSYSROOT=/opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
STRIP=arm-poky-linux-gnueabi-strip
TARGET_HARD=EM-A7
TARGET_IO=EM_A7
TARGET_PREFIX=arm-poky-linux-gnueabi-
```

【EMG7 の場合】

名前	EM7
Device type	Generic Linux Device
Device	EM ※1
sysroot	/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
Compiler (C)	GCC(EM7) ※2
Compiler (C++)	G++(EM7) ※2
Enverment	[change]ボタンを押して、後述の **Enverment(EM7)** の内容をコピー (55 行)
Debugger	GDB(EM7) ※3
Qt Version	Qt 5.7.1 (EM7) ※4
Qt mkspec	linux-oe-g++
CMake Configuration	[change]ボタンを押して、後述の **CMake Configuration (EM7)** の内容をコピー (4 行)

※1 「4.3 デバイス設定」で作成したものを選択

※2 [コンパイラ]タブで作成したものを選択

※3 [デバッグ]タブで作成したものを選択

※4 [Qt バージョン]タブで作成したものを選択

CMake Configuration (EM7) 4 行

“X=Y”の形式になります。

```
CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}
CMAKE_C_COMPILER:STRING=%{Compiler:Executable:C}
CMAKE_PREFIX_PATH:STRING=%{Qt:QT_INSTALL_PREFIX}
QT_QMAKE_EXECUTABLE:STRING=%{Qt:qmakeExecutable}
```

Enverment(EM7) ※次ページ 55 行

“XXX=YYYYY”の形式になります。

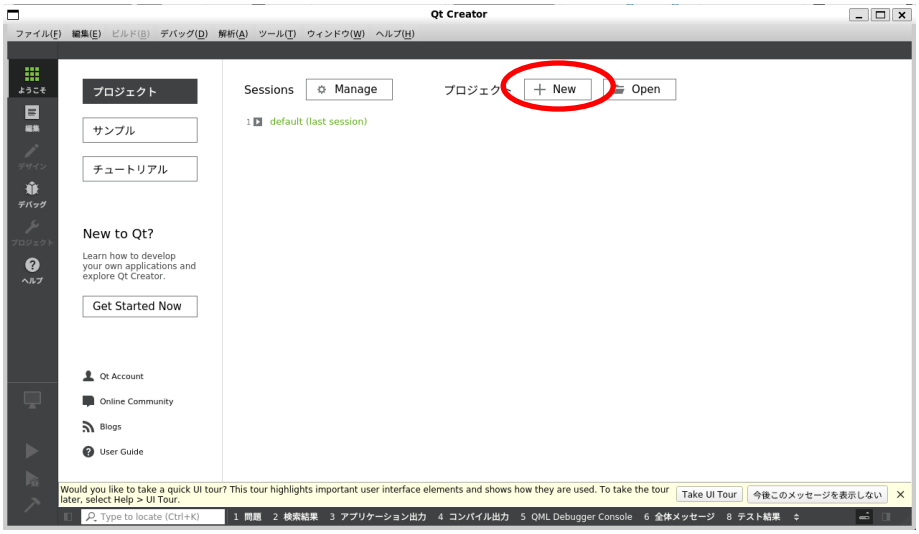
マニュアルでは表示領域の都合上、長い行は折り返されて表示されておりますが、設定時は途中で改行が入らないようにご注意ください。行の始まりは必ず XXX=になります。

```
AR=arm-poky-linux-gnueabi-ar
ARCH=arm
AS=arm-poky-linux-gnueabi-as
CC=arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
CCACHE_PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/../x86_64-pokysdk-linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-uclibc:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-musl:
CFLAGS=  -O2 -pipe -g -feliminate-unused-debug-types
CONFIGURE_FLAGS=--target=arm-poky-linux-gnueabi --host=arm-poky-linux-gnueabi --build=x86_64-linux --with-libtool-sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
CONFIG_SITE=/opt/poky/2.1.2/site-config-armv7a-neon-poky-linux-gnueabi
CPP=arm-poky-linux-gnueabi-gcc -E  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
CPPFLAGS=
CROSS_COMPILE=arm-poky-linux-gnueabi-
CXX=arm-poky-linux-gnueabi-g++  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
CXXFLAGS= -O2 -pipe -g -feliminate-unused-debug-types
GDB=arm-poky-linux-gnueabi-gdb
KCFLAGS=--sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
LD=arm-poky-linux-gnueabi-ld  --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
LDFLAGS=-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
LD_LIBRARY_PATH=/media/sf_D_DRIVE/temp/env:/media/sf_D_DRIVE/temp/env/qtcreator
M4=m4
NM=arm-poky-linux-gnueabi-nm
OBJCOPY=arm-poky-linux-gnueabi-objcopy
OBJDUMP=arm-poky-linux-gnueabi-objdump
OECORE_ACLOCAL_OPTS=-I /opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/share/aclocal
OECORE_DISTRO_VERSION=2.1.2
OECORE_NATIVE_SYSROOT=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux
OECORE_SDK_VERSION=2.1.2
OECORE_TARGET_SYSROOT=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
OE_QMAKE_AR=arm-poky-linux-gnueabi-ar
OE_QMAKE_CC=arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
OE_QMAKE_CFLAGS=  -O2 -pipe -feliminate-unused-debug-types
OE_QMAKE_CXX=arm-poky-linux-gnueabi-g++  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
OE_QMAKE_CXXFLAGS= -O2 -pipe -feliminate-unused-debug-types
OE_QMAKE_INCDIR_QT=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/include/qt5
OE_QMAKE_LDFLAGS=-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
OE_QMAKE_LIBDIR_QT=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/lib
OE_QMAKE_LINK=arm-poky-linux-gnueabi-g++  -march=armv7-a -mfpu=neon  -mfloat-abi=softfp --sysroot=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
OE_QMAKE_MOC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/moc
OE_QMAKE_PATH_HOST_BINS=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5
OE_QMAKE_QDBUSCPP2XML=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qdbuscpp2xml
OE_QMAKE_QDBUSXML2CPP=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qdbusxml2cpp
OE_QMAKE_QT_CONFIG=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/qconfig.pri
OE_QMAKE_RCC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/rcc
OE_QMAKE_UIC=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/uic
PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/sbin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/sbin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/../x86_64-pokysdk-linux/bin:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-uclibc:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-musl:/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
PKG_CONFIG_PATH=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/lib/pkgconfig
PKG_CONFIG_SYSROOT_DIR=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
QMAKESPEC=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/linux-oe-g++
QT_CONF_PATH=/opt/poky/2.1.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qt.conf
QT_IM_MODULE=ibus
RANLIB=arm-poky-linux-gnueabi-ranlib
SDKTARGETSYSROOT=/opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi
STRIP=arm-poky-linux-gnueabi-strip
TARGET_HARD=EM-A8
TARGET_IO=EM_A8
TARGET_PREFIX=arm-poky-linux-gnueabi-
```

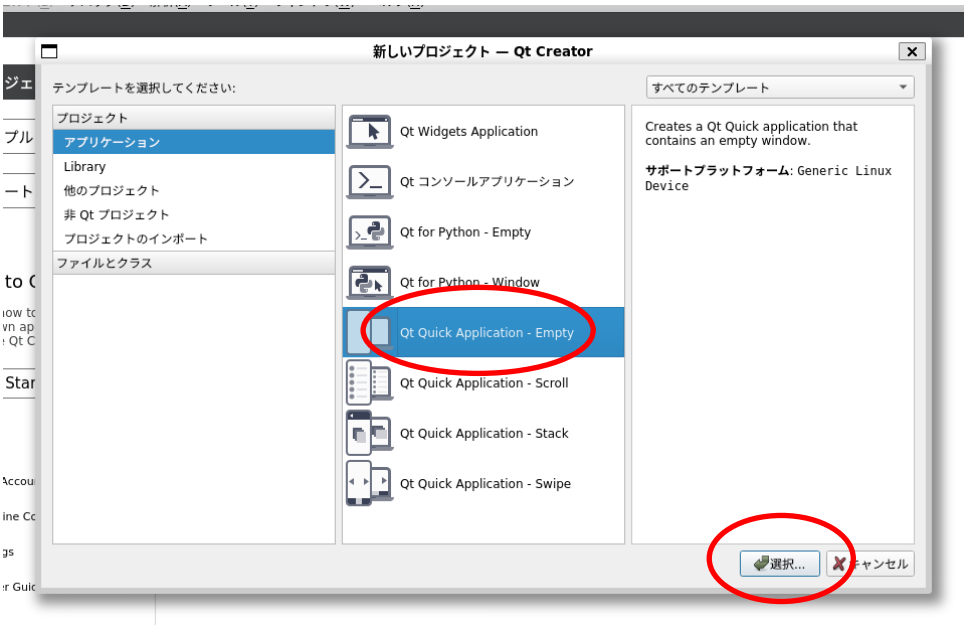
4.5　新規プロジェクト作成

ここでは、Qt Quick Application プロジェクトの例を記載しています。

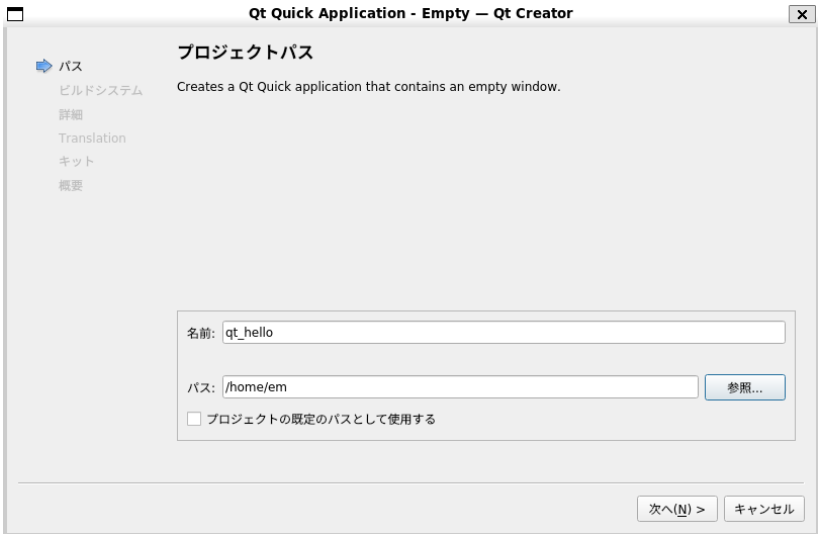
①　「＋New」をクリックします。



②　「Qt Quick Application - Empty」を選択します。



③ 任意のプロジェクト名とパスを入力します。

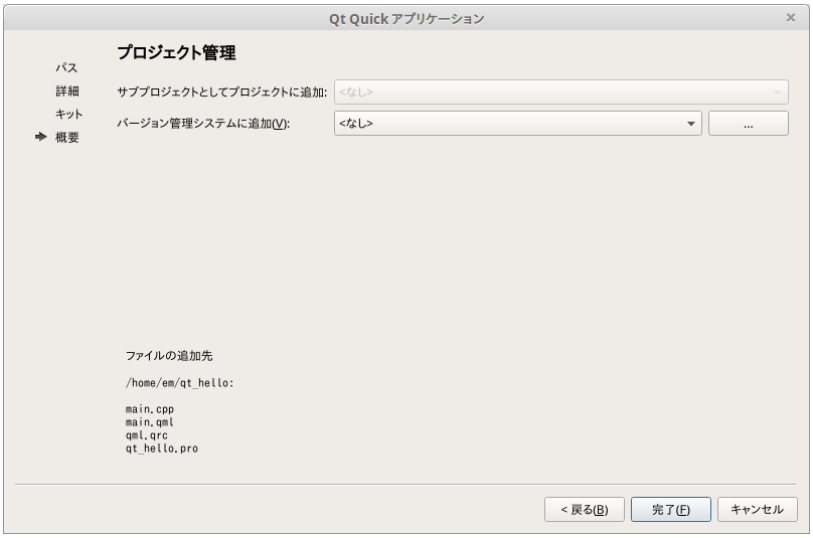


④ 引き続き使用するビルドシステムや Qt バージョン等を選択します。
今回は以下のように設定します。

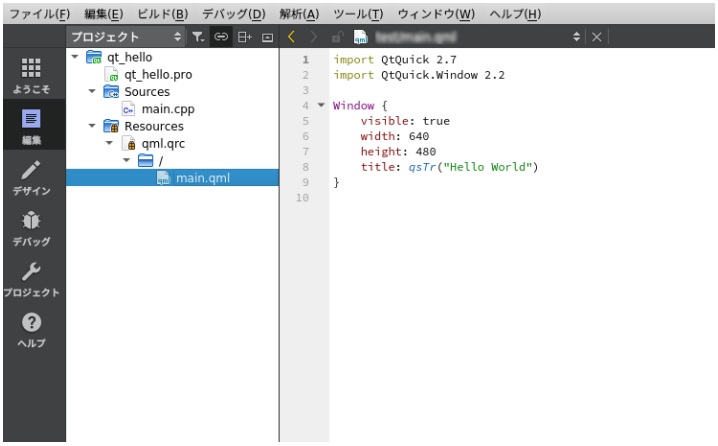
ビルドシステム	qmake
最小必要 Qt バージョン	Qt 5.7
Translation	<none>

⑤ 「4.4 キット設定」で作成したキットを選択します

⑥ ここでは、バージョン管理システムには追加しません。



プロジェクトが作成されました。



[オプション]

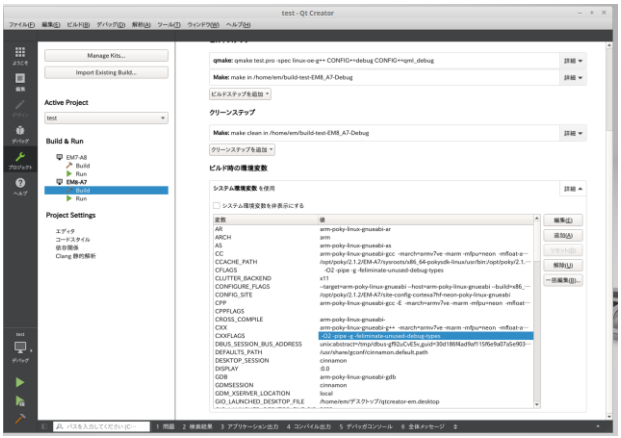
デフォルトでは、デバッグシンボルが付加された実行ファイルが生成されます。デバッグシンボルが不要な場合は、ビルド時の環境変数を以下のように変更して下さい(-g を削除)。環境変数は、プロジェクトのビルド設定から行えます。※デバッグシンボルが付加された実行ファイルは通常より大きくなります。

デバッグシンボル有り

CFLAGS	-O2 -pipe **-g** -feliminate-unused-debug-types
CXXFLAGS	-O2 -pipe **-g** -feliminate-unused-debug-types

デバッグシンボル無し

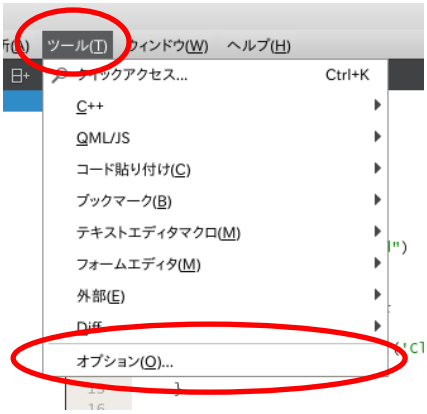
CFLAGS	-O2 -pipe -feliminate-unused-debug-types
CXXFLAGS	-O2 -pipe -feliminate-unused-debug-types

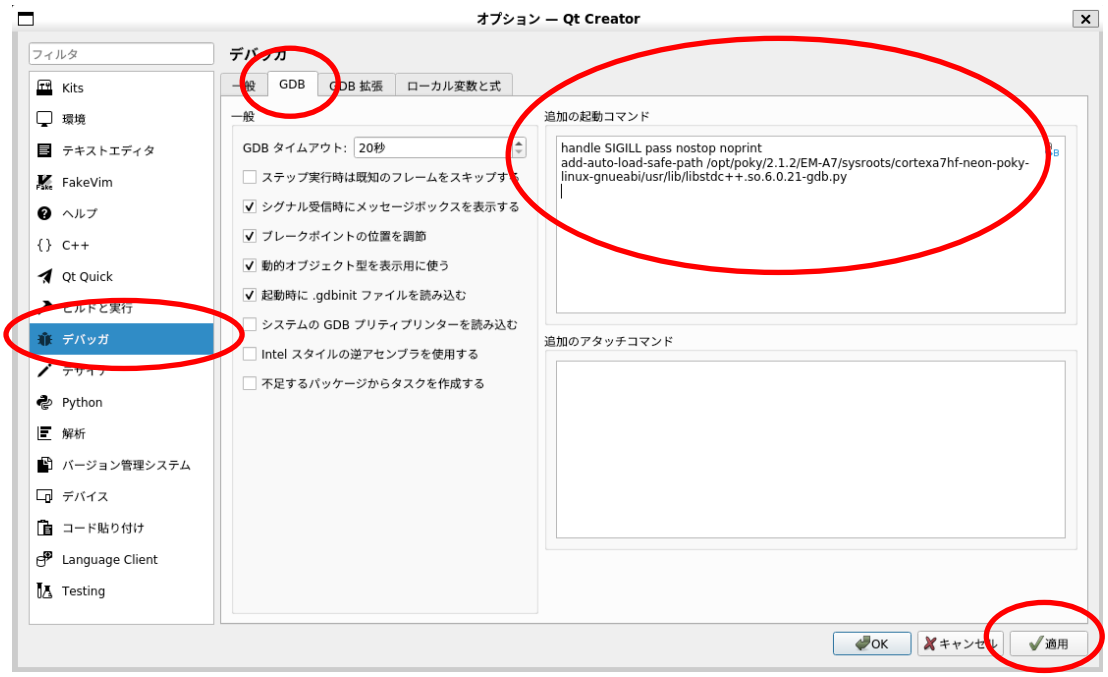


4.6　リモートデバッグ設定

ここでは、リモートデバッグの設定を行っています。
QtCreator で以下の設定を行います。

① [ツール]-[オプション]-[デバッガ]-[GDB]タブ-[追加の起動コマンド]を表示します。

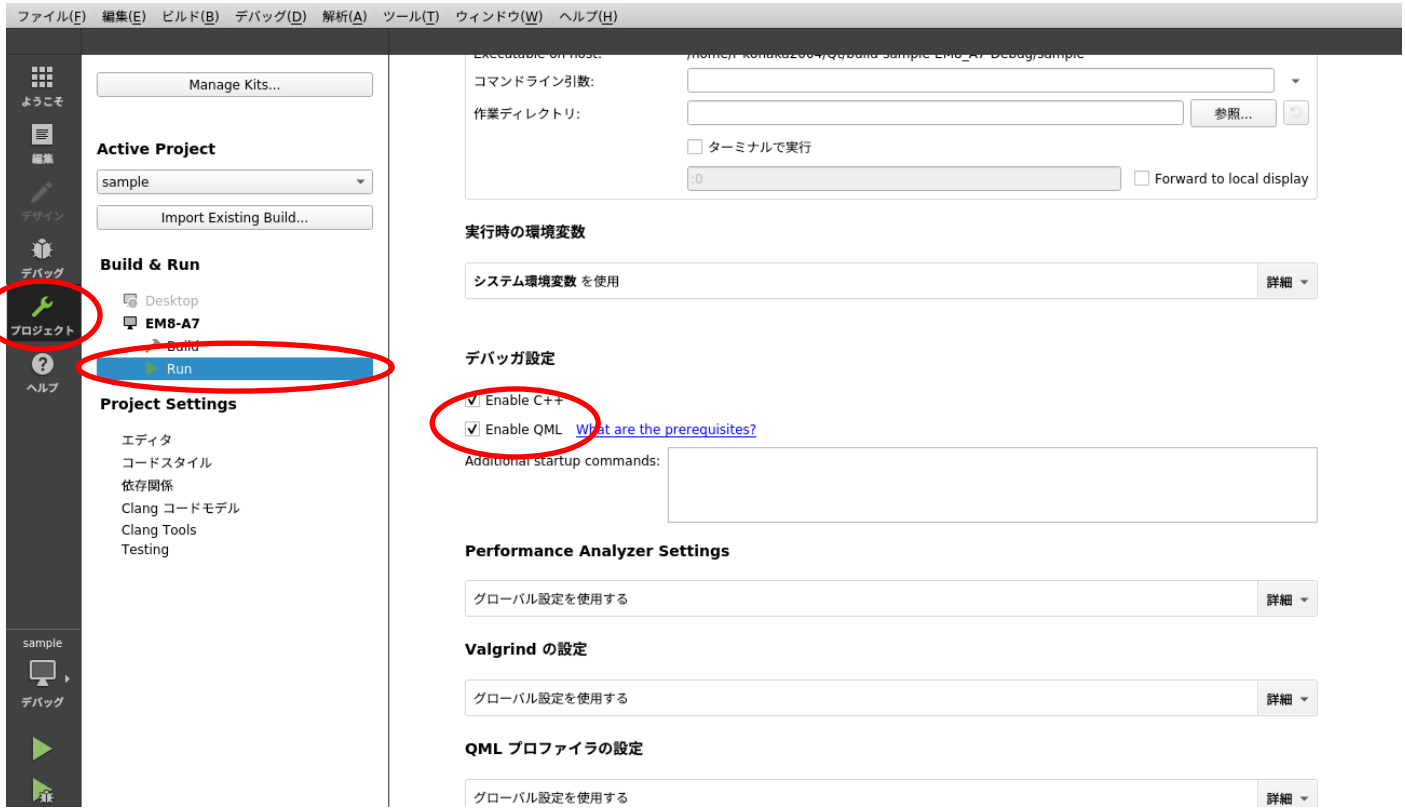




② [追加の起動コマンド]に以下の 2 行を追加し、OK を押します。

区分	追加コマンド
EM(G)8 EMP	handle SIGILL pass nostop noprint
	add-auto-load-safe-path /opt/poky/2.1.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/libstdc++.so.6.0.21-gdb.py
EMG7	handle SIGILL pass nostop noprint
	add-auto-load-safe-path /opt/poky/2.1.2/sysroots/armv7a-neon-poky-linux-gnueabi/usr/lib/libstdc++.so.6.0.21-gdb.py

③ [プロジェクト]-[Build&Run]-[Run]-[デバッガ設定]-QML を有効化にチェックします。

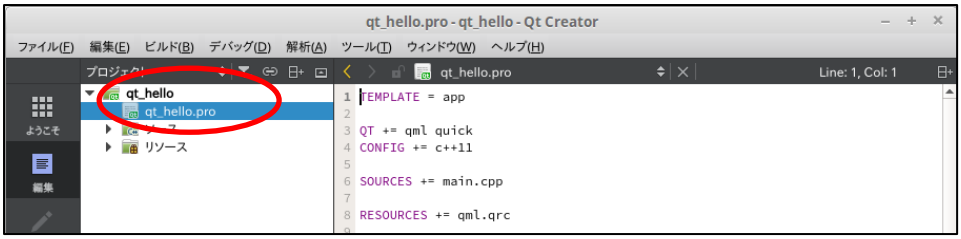


④ [プロジェクト]-[Build&Run]-[Build]-[ビルドステップ]-[qmake 詳細]-QML デバッグとプロファイルを有効にするにチェックします。

4.7　プロジェクト設定

QtCreator で実行した時に EM に転送されるパスを設定します。
ここでは「/mnt/user/」に転送されるように設定しています。
※読み取り専用にフォルダに転送するには、書き込み保護を解除する必要があります。解除方法は、「2 章 書き込み保護設定」を参照下さい。

①　プロジェクトファイル（*.pro）をダブルクリックして表示します。



②　プロジェクトファイル（*.pro）に以下を追加します。

```
INSTALLS += target
target.path = /mnt/user/
QMAKE_CXXFLAGS += -DQT_NO_OPENGL_ES_3
```

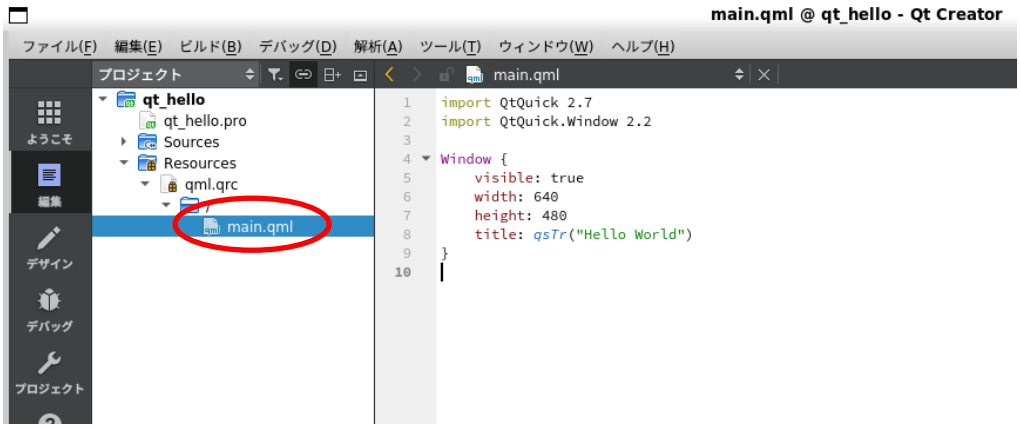
デフォルト部分はコメントアウトします。

```
27 # Default rules for deployment.
28 #qnx: target.path = /tmp/$${TARGET}/bin
29 #else: unix:!android: target.path = /opt/$${TARGET}/bin
30 #!isEmpty(target.path): INSTALLS += target
31
32 INSTALLS += target
33 target.path = /mnt/user/
34 QMAKE_CXXFLAGS += -DQT_NO_OPENGL_ES_3
35
```

4.8　QMLファイル編集

「Qt Quick アプリケーション」では、画面構成は QML ファイルに記述します。
ここでは、フルスクリーン設定と画面上に表示する文字列を設定しています。

① main.qml をダブルクリックして表示します。



② 以下を追加します。

```
visibility: "FullScreen"

Text {
        text: "Hello EM"
        anchors.centerIn: parent
        font.pixelSize: 32
        color: "black"
 }
```

```
Window {
    visible: true
    width: 640
    height: 480
    title: qsTr("Hello World")

    visibility: "FullScreen"

    Text {
            text: "Hello EM"
            anchors.centerIn: parent
            font.pixelSize: 32
            color: "black"
     }
}
```

4.9　ネットワーク設定

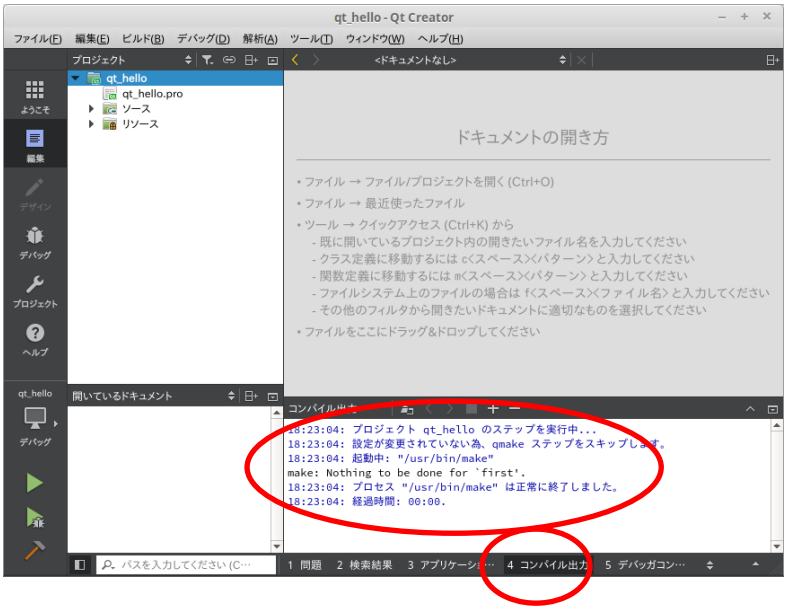
※ 設定方法は「1.2　PC と EM 本体の接続」を参照下さい。

4.10 ビルド

作成したプロジェクトをビルドします。

① 「ビルド」をクリックして下さい。

ビルドが完了しました。



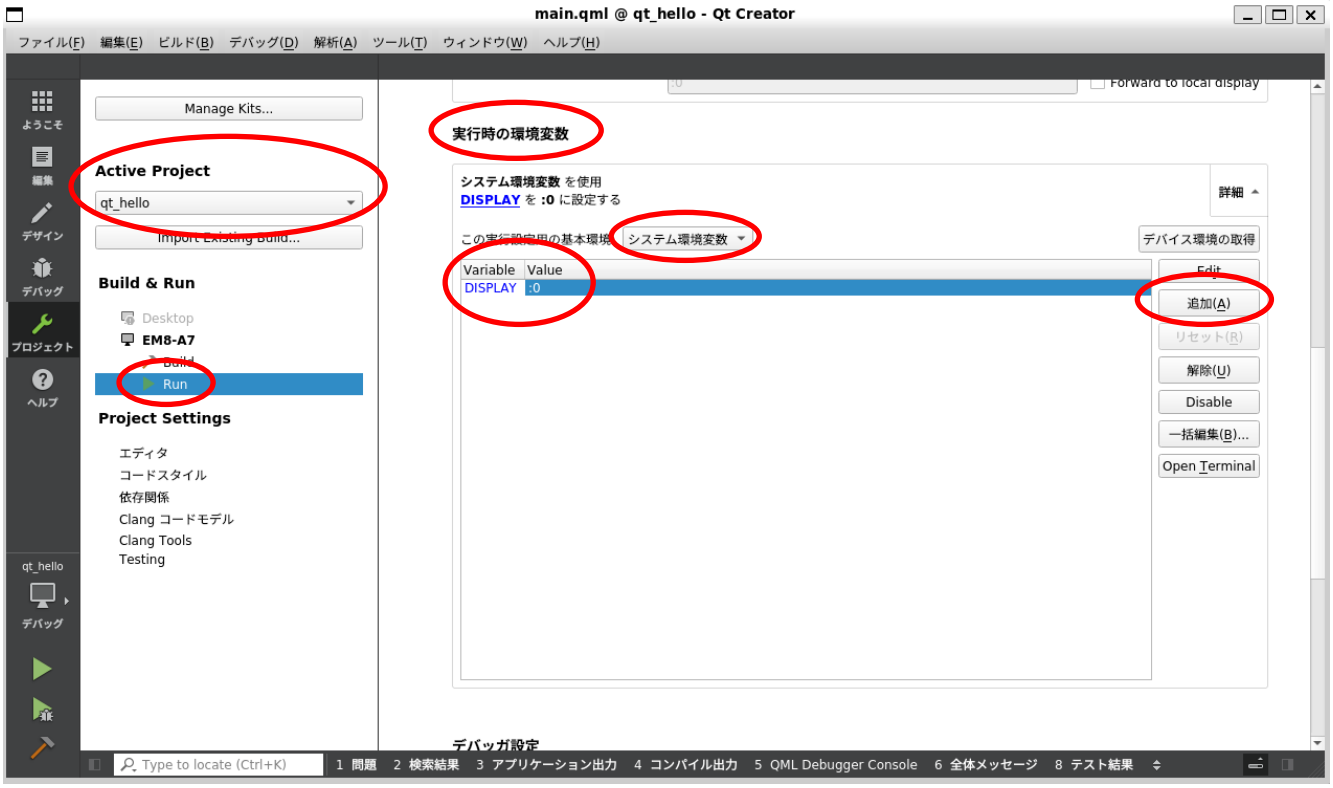
4.11 実行

実行時の環境変数

実行時の環境変数を設定します。

① プロジェクトを開き、Active Project で対象のプロジェクトを選択します。
② 左側「RUN」項目の「実行時の環境変数」で「追加」ボタンを押下し、以下のように設定します。

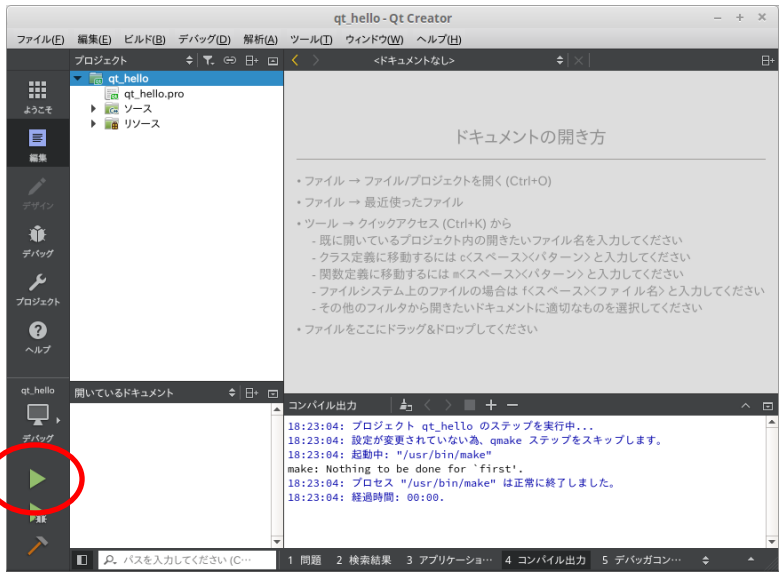
この実行設定用の基本環境	システム環境変数
Variable	DISPLAY
Value	:0



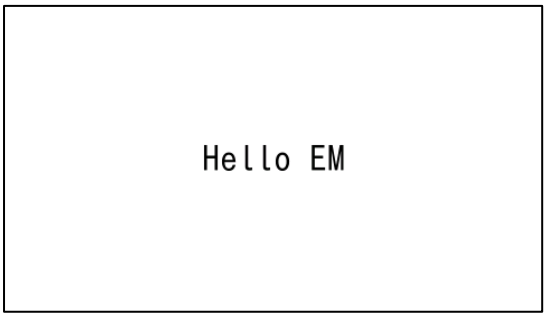
実行

実行は EM 上で行います。
「実行」すると、実行ファイルが EM に転送され、実行されます。

① 「実行」をクリックして下さい。



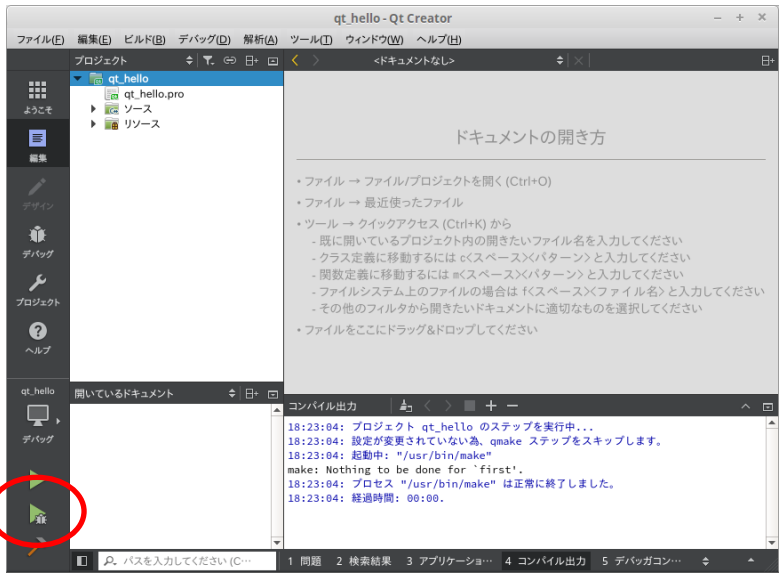
以下のような画面が表示されます。



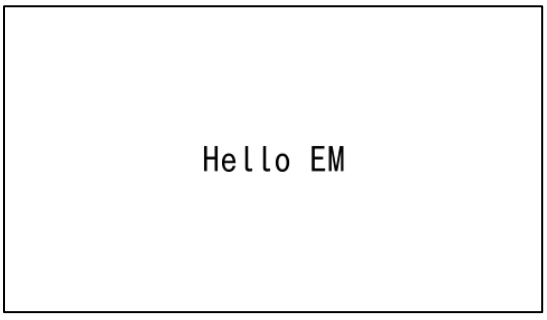
4.12 リモートデバッグ

リモートデバッグは EM 上で行います。
「デバッグ開始」すると、実行ファイルが EM に転送され、デバッグ開始されます。

① 「デバッグ開始」をクリックして下さい。



以下のような画面が表示されます。



5章　起動画面変更

実機の起動画面を変更する方法を記載します。

5.1　画像ファイル作成

画像ファイルは以下の形式でご用意下さい。

形式	非圧縮 24bit ビットマップ
サイズ	実機の解像度と同じ

5.2　転送

5.2.1　　コンソールを接続する

※　接続方法は「1.2　PC と EM 本体の接続」を参照下さい。

5.2.2　　画像ファイルを転送する

画像ファイルを EM へ転送します。

※　転送方法は「1.2　PC と EM 本体の接続」を参照下さい。

転送完了後、以下のコマンドで実機の起動画面専用エリア(/dev/mtd6)に画像ファイルを書き込みます。

※ここでは、/mnt/user/abc.bmp が画像ファイルへのパスです。

```
# psplash -w /mnt/user/abc.bmp /dev/mtd6
```

上記の手順で起動画面が変更されます。

専用エリアに書き込み後、転送した画像ファイルは削除可能です。

6章　自動起動設定

実機の起動時に自動的に動作するアプリケーションを変更する方法を記載します。
初期設定では、起動時にタスクバー、デスクトップ、EMG ランチャーが起動します。無効にする場合は、起動スクリプトを変更して下さい。

6.1　仕様

EM は、起動時に以下のスクリプトを実行します。

項目	仕様
起動スクリプト	/home/root/.config/lxsession/LXDE/autostart

デフォルトは以下の処理が登録されています。

処理	内容
@lxpanel --profile LXDE	タスクバー
@pcmanfm --desktop --profile LXDE	デスクトップ
/usr/bin/emsystem/autostart	EM オートスタート

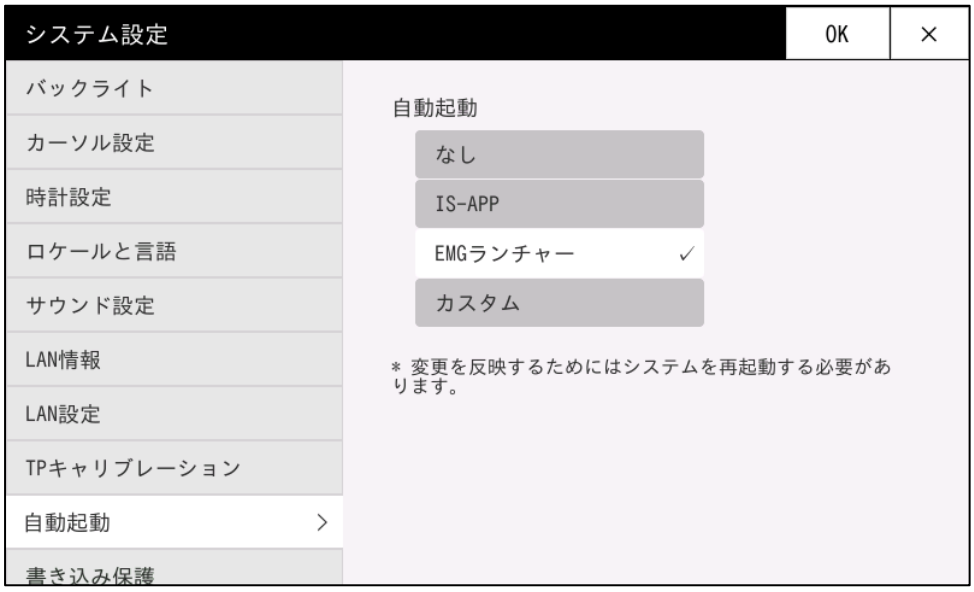
EM オートスタートは、システム設定ツールの「自動起動」で設定した項目が実行されます。デフォルトは EMG ランチャーが起動します。

EMG ランチャー

6.2　任意のプログラムを実行する方法

システム設定ツールの「自動起動」で「カスタム」を選択して下さい。
カスタムを選択すると、「/mnt/user/startup.sh」が実行されます。
vi エディタ（テキストエディタ）などで「/mnt/user/startup.sh」を編集して下さい。
システム設定ツールの「自動起動」の詳細は、別紙「ツールマニュアル」を参照下さい。

システム設定



6.3　デスクトップ、タスクバーの非表示

デスクトップ、タスクバーを非表示にする場合は以下の手順で修正を行ってください。変更するには、事前に書き込み保護を解除する必要があります。解除方法は、「2 章　書き込み保護設定」を参照下さい。

6.3.1　　コンソールを接続する

※　接続方法は「1.2　PC と EM 本体の接続」を参照下さい。

6.3.2　　起動スクリプトを修正する

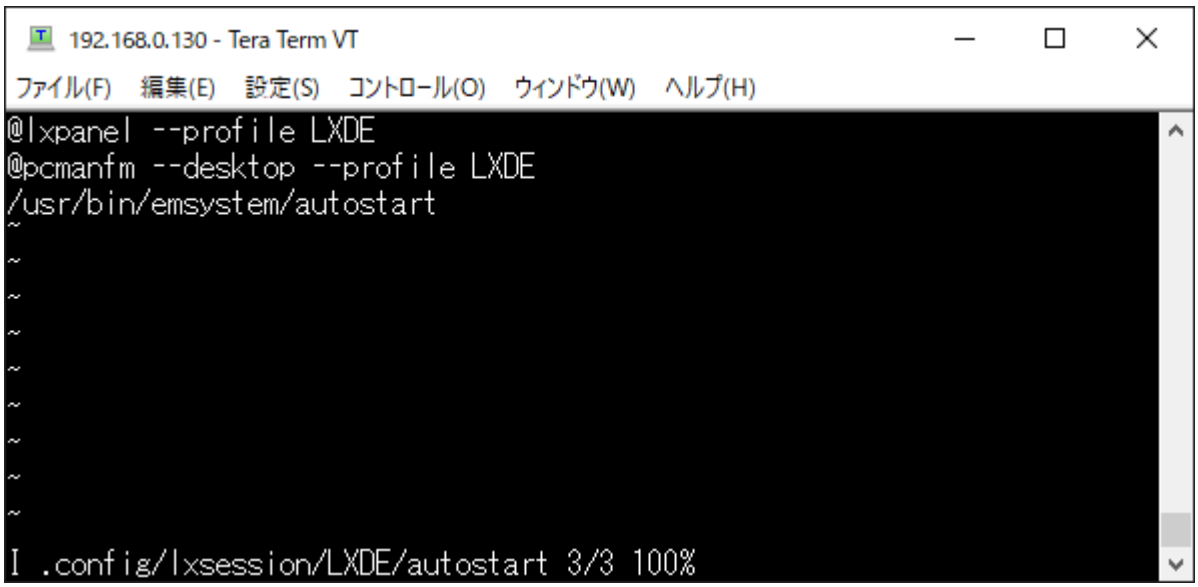
起動スクリプトを修正します。

EM のコンソールを操作します。

③　以下のコマンドで起動スクリプトを vi エディタ（テキストエディタ）で開きます。

```
# vi /home/root/.config/lxsession/LXDE/autostart
```

Vi エディタ（テキストエディタ）で起動スクリプトを編集します。



④ コマンドモードから編集モードに移行します。キーボードの「i」キーを押してください。

⑤ 起動スクリプトを編集します。

処理を無効にする場合は、無効にする行の先頭に「#」を追加して下さい。

例：タスクバーを無効にする場合

```
#@lxpanel --profile LXDE
@pcmanfm --desktop --profile LXDE
/usr/bin/emsystem/autostart
```

例：タスクバー、デスクトップを無効にする場合

```
#@lxpanel --profile LXDE
#@pcmanfm --desktop --profile LXDE
/usr/bin/emsystem/autostart
```

⑥ 編集モードを終了してコマンドモードに戻ります。キーボードの「Esc」キーを押してください。

⑦ 編集を保存して終了します。キーボードで「:wq」と入力して「Enter」キーを押してください。

⑧ 以下のコマンドで再起動を行ってください。

```
# reboot
```

再表示する場合は、「#」を削除して下さい。

```
@lxpanel --profile LXDE
@pcmanfm --desktop --profile LXDE
/usr/bin/emsystem/autostart
```

7章　EM 仕様

ファイルマップなどの仕様について記載します。

7.1　OS仕様

7.1.1　ブートローダ

EM(G)8-4／EM(G)8-5／EMP-7W／EMG7-7W／EMG7-10／EMG7-12

項目	仕様
ブートローダ	u-boot 2015.04

EM(G)8-7W／EM(G)8-10W

項目	仕様
ブートローダ	u-boot 2016.03

7.1.2　Linux カーネル

項目	仕様
カーネル	Linux 4.1.15

7.1.3　デスクトップ環境

項目	仕様
デスクトップ環境	XWindow システム(LXDE)

7.1.4　ユーザアカウント

「1.3 EM ユーザアカウント設定」を参照ください。

7.1.5　スタートメニュー

スタートメニューには以下の項目が登録されています。

項目		仕様
アクセサリ	Note	Leafpad
	ターミナル	Terminal
	イメージビューワ	Image 表示
インターネット	VNC サーバ	VNC サーバ
システムツール	LXTerminal	Terminal
	タスクマネージャ	タスク表示
	ファイルマネージャ PCManFM	ファイルマネージャ
ユニバーサルアクセス	Florence Virtual Keyboard	ソフトウェアキーボード
設定	LXSession のデフォルトアプリケーション	自動起動アプリケーション及び起動の設定
	Openbox Configuration Manager	Window の表示及び動作設定
	emglauncher	ランチャーアプリケーション(emg_launcher)
	emgetting	EM の設定ツール
	isappsetting	IS-APP の設定ツール
	キーボードとマウス	キーボードとマウスの設定
	デスクトップの設定	デスクトップ表示の設定
	デスクトップセッションの設定	自動起動の設定
	モニタの設定	ディスプレイの設定
	ルックアンドフィールの設定	ウィジェット・アイコン・フォント設定
	入力メソッド	入力ロケールの設定
実行		コマンド実行

7.2　ファイルマップ

フォルダ構成	用途	マウント位置	FileSystem/ Devicefile	Type	サイズ ※4	R/W
/bin	RootFileSystem	/	/dev/mtd3	ubifs	340M	RO※1
/boot						
/home						
/lib						
/media						
/mnt						
/sbin						
/usr						
/var						
/www						
/etc						
/dev	devfs	/dev	udev	devtmpfs	(RAM)	RW
/proc	procfs	/proc	proc	proc	(RAM)	RW
/sys	sysfs	/sys	sysfs	sysfs	(RAM)	RW
/run	一時ファイル	/run	tmpfs	tmpfs	(RAM)	RW
/tmp	一時ファイル	/tmp	tmpfs	tmpfs	(RAM)	RW
/var/volatile	一時ファイル	/var/volatile	tmpfs	tmpfs	(RAM)	RW
/mnt/user	ユーザ領域 1	/mnt/user	/dev/mtd4	ubifs	90MB	RW ※2
/mnt/user2 ※3	ユーザ領域 2 ※3	/mnt/user2	/dev/mtd5	ubifs	65MB ※3	RW ※2

※1 RootFileSystem は読み取り専用(RO)で出荷しております。読み取り専用エリアにファイルを書き込む場合は、書き込み保護を解除する必要があります。詳しくは、「2 章 書き込み保護設定」を参照下さい。

※2 ユーザ領域も読み取り専用(RO)に変更することが可能です。詳しくは、「2.1 ユーザ領域の書き込み保護領域の設定」を参照下さい。

※3 EM(G)8-4-SS / EM(G)8-5-SS / EM(G)8-7W-SS / EM(G)8-10W-SS / EMP-7W-SS の場合は高速起動用領域に予約される為使用できません。

※4 一部システムの管理用に使用される為、ファイルシステムのディスク容量に表示されるサイズは少なくなります。

7.3　ルートファイルシステム

パッケージ・ライセンス

EM にインストールされているパッケージ並びに、そのライセンス条項は、DVD-ROM（開発環境一式）内の licenses.tar.gz を参照下さい。

7.4　ドライバ

7.4.1　　LAN 仕様

ポート	デバイス	IP	備考
有線 LAN	eth0	dhcp/static	10/100BASE-TX ※IP の初期値は 192.168.0.130 / 24

LAN 設定ファイル

ファイル	説明
/etc/resolv.conf	DNS サーバ設定
/etc/network/interfaces	ネットワーク設定
/etc/network/lan0.conf	eth0 設定

［ご注意］
本システムでは、USB-Ether (usb0) で「192.168.10.*」のネットワークを使用しております。
IP アドレスを設定される場合、競合にご注意ください。
※「192.168.10.*」に設定された場合、正常に通信が行えません。

7.4.2　　タッチパネル仕様

Linux 標準の InputDriver に準じています。

項目	説明
仕様	LinuxInputSubsystem および tslib によるサポート
サンプリング周期	50 回/秒
デバイス	/dev/input/touchscreen0
キャリブレーションツール （静電容量オフセット）	EMG8 ※EMG8-10W を除く /usr/bin/tpoffset EMG8-10W 無し（オートキャリブレーション） EMG7 /usr/bin/Calibration
キャリブレーションツール （座標キャリブレーション）	EM8 / EMP /usr/bin/xinput_calibrator

7.4.3 サウンド仕様

項目	説明
ドライバ	ALSA ドライバ 1.1.0
仕様	LinuxALSA ドライバ仕様に準拠
対応形式	非圧縮の WAV ファイル
実行方法	対象機種：EM(G)8-7W、EM(G)8-10W
	alsaplayer Volume：0.0～1.0 (0.0=消音、0.35=35%、1.0=100%) ※ボリュームは実行ごとに指定して下さい。 例): alsaplayer --startvolume *<Volume>* --enqueue sample.wav
	対象機種：EMG7-7W、EMG7-10、EMG7-12
	aplay 例): aplay -D hw:0,0 sample.wav | arecord -D hw:0,0 -f S16_LE -r 44100 -c 2 -d 5 record.wav

7.4.4 USB 仕様

USB ホスト

ホストドライバ: EHCI HCD (USB2.0)

クラスドライバ	説明
USB HUB	USB ハブ
USB Mass Storage	USB メモリ
USB HID	USB マウスおよびキーボード

USB デバイス

USB Gadget Drivers: USB-Ether

Gadget	説明
USB-Ether	本機の USB ポートを PC と接続した時、PC 側からは Ethernet Adapter として認識されます。 ※PC へ USB デバイスドライバのインストールが必要です。詳しくは「1.2.2 USB デバイスポートでのネットワーク設定」を参照下さい。 ※本機と PC は 1 対 1 の接続でご使用下さい。 デバイス：usb0 IP アドレス：192.168.10.130 LAN 設定ファイル：/etc/network/glan0.conf

7.4.5 LCD 仕様

Linux 標準のフレームバッファをサポートします。

デバイスファイル：/dev/fb0

機能		仕様
open		Linux フレームバッファドライバの仕様に準拠
close		
ioctl	FBIOGET_FSCREENINFO	固定画面情報を取得します。 戻り値　0：成功 0 以外:失敗 入力　なし 出力　fb_fix_screeninfo*型変数：取得した固定画面情報
	FBIOGET_VSCREENINFO	変動画面情報を取得します。 戻り値　0:成功 0 以外:失敗 入力　なし 出力　fb_var_screeninfo*型変数：取得した変動画面情報
	FBIOPUT_VSCREENINFO	変動画面情報を設定します。 戻り値　0:成功 0 以外：失敗 入力　fb_var_screeninfo*型変数：設定する変動画面情報

mmap	Linux の mmap の仕様に準拠

注１） 画面情報から１画面分のメモリを mmap を使用してバッファメモリをマッピングして、画面描画を行
うことが出来ます。
但し、システム描画との排他制御が行われませんので、システム側の描画が行われることにより、意図した画面描画が行われない可能性があります。
（システム側の描画で上書きされてしまいます。）
この場合、システム側の描画が行われないようタスクバーを非表示にする等、考慮する必要があります。

7.4.6 バックライト仕様

ファイルパス：/sys/class/backlight/backlight/

バックライトの制御は、以下のファイルにより行います。

ファイル	Read/Write	説明
bl_power	R/W	バックライト ON/OFF 取得・設定 0： 点灯 1： 消灯
brightness	R/W	輝度取得・設定 1～8（8 段階）0 は OFF
bl_autooffenable	R/W	バックライト自動消灯 有効無効取得・設定 0:無効 1:有効
bl_autoofftime	R/W	バックライト自動消灯 時間取得・設定 1～65535（秒）

注１） バックライト自動消灯とは、最後のタッチ入力からバックライト自動消灯時間経過してもタッチ入力がなかった場合、自動的にバックライトを消灯することです。

注２） バックライトが自動消灯された場合、タッチ入力があるかもしくは、アプリケーションから
bl_power
を 0 で実行することにより、バックライトは点灯します。

注３） タッチ入力以外（bl_power など）でバックライトを点灯した場合は、バックライト自動消灯は動作しません。最後のタッチ入力から自動消灯時間を経過した場合に消灯します。

注４） 輝度・バックライト自動消灯有効無効・バックライト自動消灯時間の各パラメータは、設定時に自動的に保存されます。よって、次回ブート時には最後に設定されたパラメータで起動されます。

7.4.7　SIO 仕様

ポート割り当て

EM(G)8-4／EM(G)8-5／EM(G)8-7W／EM(G)8-10W

ポート	デバイスファイル	I/F	備考
SIO1	/dev/com1	RS232C	/dev/ttymxc4 へのリンク
SIO2	/dev/com2	RS422／RS485	/dev/ttymxc2 へのリンク I/F は DIPSW による切り替え

EMP-7W

ポート	デバイスファイル	I/F	備考
SIO2	/dev/ttymxc2	RS422	

EMG7-7W

ポート	デバイスファイル	I/F	備考
SIO1	/dev/com1	RS232C	/dev/ttymxc0 へのリンク

EMG7-10／EMG7-12

ポート	デバイスファイル	I/F	備考
SIO1	/dev/com1	RS232C	/dev/ttymxc0 へのリンク
SIO2	/dev/com2	RS485	/dev/ttymxc3 へのリンク

共通仕様

機能	仕様
全般	Linux シリアルドライバ仕様に準拠

RS422 仕様

初期化処理内で通信モードを RS422 に設定して下さい。
データ送信時は RTS を ON にして、送信後に OFF にして下さい。
拡張コントロールコードは、DVD-ROM(開発環境一式)内の「software」-「ioctrl_include」の「seedsware_ext_ioctl.h」に定義しております。include してご使用下さい

機能		仕様
全般		Linux シリアルドライバ仕様に準拠
ioctl	IOCTL_UART_MODE_RS422	通信モードを RS422 に設定します。 戻り値　0:成功 -1：失敗

		入力　0
	IOCTL_UART_ASSERT_DE	RTS を ON にします。 戻り値　0:成功 -1：失敗 入力　NULL
	IOCTL_UART_DEASSERT_DE	RTS を OFF にします。 戻り値　0:成功 -1：失敗 入力　NULL

RS485 仕様

初期化処理内で通信モードを RS485 に設定して下さい。

拡張コントロールコードは、DVD-ROM(開発環境一式)内の「software」-「ioctrl_include」の「seedsware_ext_ioctl.h」に定義しております。include してご使用下さい。

機能		仕様
全般		Linux シリアルドライバ仕様に準拠
ioctl	IOCTL_UART_MODE_RS485	通信モードを RS485 に設定します。 戻り値　0:成功 -1：失敗 入力　0
	TIOCSRS485	RS485 動作詳細を設定します。 戻り値　0:成功 0 以外：失敗 入力　rs485_config 型ポインタ 出力　なし
	TIOCGRS485	RS485 動作詳細を取得します。 戻り値　0:成功 0 以外:失敗 入力　なし 出力　rs485_config 型ポインタ
flags	SER_RS485_ENABLED	ドライバでの RS485 動作を許可します
	SER_RS485_RTS_ON_SEND	送信時 RTS を ON し、送信後 OFF します
	SER_RS485_RTS_AFTER_SEND	送信時 RTS を OFF し、送信後 ON します
	SER_RS485_RX_DURING_TX	送信時、受信を有効にします

注）終端抵抗が必要な場合、以下の手順で終端抵抗を有効にすることが可能です。

EM(G)8-4／EM(G)8-5／EM(G)8-7W／EM(G)8-10W の場合

シリアルポート設定用 4 連 DipSW の SW1 を ON することにより、終点抵抗が有効となります。

EMG7-10／EMG7-12 の場合

/sys/class/gpio/status_terminate/value に 1 を書き込むことで有効となります。

コマンドで有効にする場合　echo 1 >/sys/class/gpio/status_terminate/value

7.4.8 RTC 仕様

機能	仕様
設定可能日時	2000/1/1 00:00:00 ～ 2037/12/31 23:59:59
閏年	2000～2037 年の範囲で対応
バッテリ切れ時動作	ドライバ初期化時(=OS 起動時)にバッテリ切れを検出 検出時は 1970/1/1 00:00:00 にデバイスをリセットする

デバイスファイル：/dev/rtc0

本ドライバは、外部 RTC を制御するためのドライバです。

外部 RTC とは、電源 OFF されてもバッテリで動作する RTC です。

機能	仕様
open	Linux 標準 RTC ドライバの仕様に準拠
close	
ioctl	

注１）通常の日付・時刻設定は、Linux 標準インタフェースで行えます。
設定した日付・時刻を外部 RTC に反映させるために、hwsync 関数を実行して下さい。
hwsync 関数を実行しない場合、電源 OFF で無効となります。

7.4.9　状態表示 LED 仕様

EMG7-7W / EMG7-10 / EMG7-12

対象機種
EMG7-7W
EMG7-10
EMG7-12

本体正面の状態表示 LED は以下のように動作します。

状態	LED 表示色
電源 OFF	消灯
ブートローダ動作時	橙
OS 起動時	橙
通常時	緑
バックライト消灯時	緑点滅（0.5 秒周期　自動実行）
バックライト不良時	赤点滅（0.5 秒周期　自動実行）

また、以下のファイルを読み出すことにより LED の状態を取得でき、書き込むことにより LED を設定することができます。

EMG7-7W

ファイル	Read/Write	説明
/sys/class/leds/status_led_green/brightness	R/W	LED 緑色 1： 点灯 0： 消灯
/sys/class/gpio/status_led_red/brightness	R/W	LED 赤色 1： 点灯 0： 消灯

EMG7-10/ EMG7-12

ファイル	Read/Write	説明
/sys/class/leds/status_led_green/brightness	R/W	LED 緑色 1： 点灯 0： 消灯
/sys/class/gpio/status_led_red/brightness	R/W	LED 赤色 1： 点灯 0： 消灯

注１） システムでも LED を制御していますが、上記ファイルにより LED の設定を行った場合、その設定が反映されます。
また、アプリケーションより LED 設定されても、バックライト消灯・バックライト不良が発生した場合、緑点滅・赤点滅となります。

EMP-7W

対象機種
EMP-7W

状態	LED 表示色
電源 OFF	消灯
ブートローダ動作時	緑
OS 起動時	緑
通常時	緑

また、以下のファイルを読み出すことにより LED の状態を取得でき、書き込むことにより LED を設定することができます。

ファイル	Read/Write	説明
/sys/class/leds/led_green/brightness	R/W	LED 緑色 1： 点灯 0： 消灯
/sys/class/leds/led_red/brightness	R/W	LED 赤色 1： 点灯 0： 消灯

注１） システムでも LED を制御していますが、上記ファイルにより LED の設定を行った場合、その設定が反映されます。

7.4.10　SRAM 仕様

対象機種
EMG7-7W
EMG7-10
EMG7-12
EM(G)8-4
EM(G)8-5

EMG7-7W / EMG7-10/ EMG7-12 は、/dev/mem のメモリデバイスとして SRAM をアクセスすることができます。
EM(G)8-4 / EM(G)8-5 は、/dev/sram1 の SPI デバイスとして SRAM をアクセスすることができます。

デバイスファイル：/dev/mem（EMG7-7W / EMG7-10/ EMG7-12）

機能	仕様
open	Linux 標準 mem ドライバの仕様に準拠
close	
mmap	
read	
write	
ioctl	

デバイスファイル：/dev/sram1（EM(G)8-4 / EM(G)8-5）
拡張コントロールコードは、DVD-ROM(開発環境一式)内の「software」-「ioctrl_include」の「seedsware_ext_ioctl.h」に定義しております。include してご使用下さい。

機能		仕様
open		Linux 標準ドライバの仕様に準拠
close		
read		
write		
mmap		関数呼び出し仕様は、Linux 標準 mmap 同等 SRAM へのインタフェースが SPI であるため、ドライバで内部メモリを確保し、疑似的な mmap 仕様をサポートします。
ioctl	IOCTL_SRAM_GET_SIZE	SRAM サイズを取得します。 戻り値　0:成功 0 以外:失敗 入力　なし 出力　int 型変数ポインタ　SRAM のサイズ

	IOCTL_SRAM_FLUSH_M2D	mmap で取得した全内部メモリデータ領域を SRAM に書き込みます。 戻り値　0:成功 　　0 以外:失敗 入力　なし 出力　なし
	IOCTL_SRAM_FLUSH_M2D_PAR	mmap で取得した全内部メモリデータ領域を io_sram_t で指定された領域を SRAM に書き込みます。 戻り値　0:成功 　　0 以外:失敗 入力　io_sram_t 型ポインタ（オフセット・サイズ） 出力　なし
	IOCTL_SRAM_FLUSH_D2M	全 SRAM データを mmap で取得した内部メモリデータ領域に読み出します。 戻り値　0:成功 　　0 以外:失敗 入力　なし 出力　なし
	IOCTL_SRAM_FLUSH_D2M_PART	指定された SRAM 領域データを mmap で取得した内部メモリデータ領域に読み出します。 戻り値　0:成功 　　0 以外:失敗 入力　io_sram_t 型ポインタ（オフセット・サイズ） 出力　なし
	IOCTL_SRAM_SEEK	read･write 関数でアクセスするオフセットを設定します。 戻り値　0:成功 　　0 以外:失敗 入力　int 型ポインタ　オフセット：0～SRAM 最大値-1 出力　なし

注１）read･write 関数は、SPI インタフェースを使用して、SRAM に直接読み込み書き込みを行います。
　　read･write 関数を使用する前に、IOCTL_SRAM_SEEK によりオフセットを設定して下さい。
　　read･write 関数実行により、オフセットは変化しません。
　　読み書きしたいオフセットが変わる場合､必ず IOCTL_SRAM_SEE によりオフセットを設定して下さい。

注２）ｵﾌｾｯﾄと read･write 関数で要求できる最小・最大サイズは、以下の通りです。
　　最小サイズ:1　最大サイズ：SRAM サイズーオフセット

注３）mmap の要求サイズは、SRAM サイズとして下さい。

注４）mmap 関数は、内部メモリを確保するのみです。
　　内部メモリは不定データですので、IOCTL_SRAM_FLUSH_D2M を実行して内部メモリに
　　SRAM データを読み出して下さい。

（ア）　mmap で確保した領域のポインタを使用して、アプリケーションから読み書きを実行できます。
　　ポインタを使用した読み書きでは、SRAM には書き込みは行われません。

IOCTL_SRAM_FLUSH_D2M･IOCTL_SRAM_FLUSH_D2M_PART を使用して、内部メモリデータを
SRAM に書き込む処理を行って下さい。

（イ） IOCTL_SRAM_FLUSH_D2M･IOCTL_SRAM_FLUSH_D2M_PART が実行されなかった場合、電源 OFF で内部メモリデータが破棄されます。よって、SRAM データは IOCTL_SRAM_FLUSH_D2M･IOCTL_SRAM_FLUSH_D2M_PART が最後に実行された時の状態のままとなります。

注５）バッテリ切れ時の SRAM 内の値は不定データになります。

7.4.11 BUZZER 仕様

デバイスファイル：/dev/buzzer

拡張コントロールコードは、DVD-ROM(開発環境一式)内の「software」-「ioctrl_include」の「seedsware_ext_ioctl.h」に定義しております。include してご使用下さい。

機能		仕様
open		デバイスのオープン,クローズ
close		
ioctl	IOCTL_PLAY	ブザー鳴動を開始します。 返り値　常に 0 入力　　なし 出力　　なし
	IOCTL_STOP	ブザーの鳴動を停止します。 返り値　常に 0 入力　　なし 出力　　なし
	IOCTL_BEEP	指定時間の間、ブザーを鳴動します。 返り値　0:成功 　　0 以外:失敗 入力　　int 型変数：鳴動時間（100ms 単位） 出力　　なし
	IOCTL_SET_BEEP_INTERVAL	ブザーの周波数を設定します。 返り値　0:成功 　　0 以外:失敗 入力　　int 型変数：設定する周波数（1～30000Hz） 出力　　なし
	IOCTL_GET_BEEP_INTERVAL	ブザーの設定周波数を取得します。 返り値　0:成功 　　0 以外:失敗 入力　　なし 出力　　int 型変数：設定した周波数（Hz）
	IOCTL_GET_BUZZER_STATE	ブザーの状態を取得します。 返り値　0:成功 　　0 以外:失敗

		入力　　なし 出力　　0：停止 　　1：鳴動

注１）　ブザーの鳴動は、要求されたコマンドが随時で処理されます。
そのため、IOCTL_BEEP でブザー鳴動状態でも、次に再度 IOCTL_BEEP 異なった鳴動時間で要求された場合、後に実行された IOCTL_BEEP の鳴動時間で再設定されます。

注２）　システムのタッチ音を有効にしている場合、ドライバのブザー鳴動とタッチ音は、排他的な処理となっていませんので、タッチ音発生のために意図した時間ブザーが鳴動しない可能性があります。
このような場合、システムのタッチ音を無効にして、アプリケーションレベルで
本ドライバを使用して、タッチ音を発生させるようなプログラムを作成して下さい。

7.4.12　DIO 仕様

製品の DIO インタフェースにスイッチ、LED などを接続することが出来ます。
機種によって DIO API を使い制御する場合と、通常の GPIO を使用して制御する場合があります。

EM(G)8-4 / EM(G)8-5

対象機種
EM(G)8-4
EM(G)8-5

対象機種の場合は、動作モードの切替、入出力操作は DIO API を通じて行うことが可能です。
DIO API については、「7.6.1　DIO API」を参照下さい。

動作モードは、以下の 3 種類が存在します。

DIO SCAN ありモード
4 点の DOUT を SCAN ライン、6 点を DIN とし、24 点の DIN として動作します。
24 点の DIN の状態は、通常の DIN 入力として DIO API を使用して取得することができます。
本モードでは、取得時に SCAN 処理を実施します。
また、8 点の DOUT も DIO API を使用して制御可能です。

DIN 数	DOUT 数
24	8

DIO SCAN なしモード

12 点の DOUT と 6 点の DIN として、動作します。

12 点の DOUT と 6 点の DIN は、DIO API を使用して制御可能です。

DIN 数	DOUT 数
6	12

シートキーモード

4 点の DOUT を SCAN ライン・6 点の DIN を RETURN ラインとして 24 点のシートキーとして動作します。

1 回の処理で 4 ラインの処理を実施し、処理間隔は 100ms です。

24 点は、キーコードマッピングされており、キー入力として認識されます。

キー入力は、キーコードとステータスの ON・OFF となります。

キーリピートは、サポートしません。

また、8 点の DOUT も DIO API を使用して制御可能です。

DIN 数	DOUT 数
-	8

EM(G)8-7W / EM(G)8-10W

対象機種
EM(G)8-7W
EM(G)8-10W

通常の GPIO を使用して、DOUT4 点/DIN4 点を使用することが可能です。

正論理(On : 1 / Off : 0)で動作します。

/sys/class/gpio/gpio<*GPIO 番号*>/value」を Read/Write することでアクセス可能です。

DOUT4 点

DOUT1	DOUT2	DOUT3	DOUT4
GPIO4_25(gpio121)	GPIO4_26(gpio122)	GPIO4_27(gpio123)	GPIO4_28(gpio124)

DIN4 点

DIN1	DIN2	DIN3	DIN4
GPIO4_17(gpio113)	GPIO4_18(gpio114)	GPIO4_19(gpio115)	GPIO4_20(gpio116)

7.4.13　KPP 仕様

対象機種
EMP-7W

デバイスファイル：/dev/input/event0

機能		仕様
open		Linux 標準ドライバの仕様に準拠 デバイスのリード（キーイベントの取得）時に以下のように指定すると、キーイベントを１つ取得することができます。 【入力】 int 型：ファイルディスクリプタ struct input_event 型のポインタ：イベントを受け取るバッファ size_t 型　：イベントを受け取るバッファサイズ 【出力】 キーイベントがあれば、struct input_event 型のポインタが示すバッファに１つキーイベント取得
Close		
Read		

ioctl	EVIOCSREP	オートキーリピートの情報を設定します。
		【返り値】
		0 以上：成功
		0 より小さい：失敗
		【入力】
		int 型：ファイルディスクリプタ
		unsigned long 型：コントロールコード
		struct kbd_repeat 型ポインタ：
		オートリピート情報を設定するバッファポインタ
		【出力】
		なし
		struct kbd_repeat {
		　int delay;　キー押下からオートリピート処理開始までの時間
		　int period;　オートリピート処理間隔
		};
		デフォルト設定
		　delay：200ms
		　period：33ms

		オートリピートを停止させたい場合、delay・periodを0にして設定して下さい。 また、デフォルト設定以下の値は、指定しないで下さい。
	EVIOCGREP	オートキーリピートの情報を取得します。 【返り値】 0以上：成功 0より小さい：失敗 【入力】 int型：ファイルディスクリプタ unsigned long型：コントロールコード struct kbd_repeat型ポインタ： オートリピート情報を設定するバッファポインタ 【出力】 struct kbd_repeat型ポインタの示すバッファにオートキーリピート情報取得
	IOCTL_GETKPPBUZZER_ENBALE	キー押下時、ブザー鼓動させるかどうかの設定を取得します。 【返り値】 0以上：成功 0より小さい：失敗 【入力】 int型：ファイルディスクリプタ unsigned long型：コントロールコード bool型ポインタ： 設定情報を取得するバッファのポインタ 【出力】 設定情報を取得するバッファの示すポインタに設定情報取得 ｔrue：キー押下でブザー鼓動 ｆalse：ブザー鼓動なし オートリピート処理でのキーイベントについては、ブザー鼓動しません。
	IOCTL_SETKPPBUZZER_ENBALE	キー押下時、ブザー鼓動させるかどうかを設定します。 【返り値】 0以上：成功 0より小さい：失敗 【入力】

		int 型：ファイルディスクリプタ unsigned long 型：コントロールコード bool 型ポインタ： 設定情報を指定するバッファのポインタ true：キー押下でブザー鼓動 false：ブザー鼓動なし出力 【出力】 なし オートリピート処理でのキーイベントについては、ブザー鼓動しません。
	IOCTL_GETBACKLIGHTON_SENDEVENT_ENBALE	バックライト消灯時､キー押下でバックライトONイベントを送信するかどうかの設定を取得します。 【返り値】 0 以上：成功 0 より小さい：失敗 【入力】 int 型：ファイルディスクリプタ unsigned long 型：コントロールコード bool 型ポインタ： 設定情報を取得するバッファのポインタ 【出力】 設定情報を取得するバッファの示すポインタに設定情報取得 true：キー押下時、バックライト ON イベントを送信する。 false：バックライト ON イベント送信しない。
	IOCTL_SETBACKLIGHTON_SENDEVENT_ENBALE	バックライト消灯時､キー押下でバックライトONイベントを送信するかどうか設定します。 【返り値】 0 以上：成功 0 より小さい：失敗 【入力】 int 型：ファイルディスクリプタ unsigned long 型：コントロールコード bool 型ポインタ： 設定情報を指定するバッファのポインタ true：キー押下時、バックライト ON イベントを送信する。

		false：バックライト ON イベント送信 しない。 【出力】 なし
	IOCTL_GETBACKLIGHTOFF_NOKEYEVENT	バックライト消灯時、キーイベントを送信するかどうかの設定を取得します。 （バックライト消灯時の誤操作防止のため） 【返り値】 0 以上：成功 0 より小さい：失敗 【入力】 int 型：ファイルディスクリプタ unsigned long 型：コントロールコード bool 型ポインタ： 設定情報を取得するバッファのポインタ 【出力】 設定情報を取得するバッファの示すポインタに設定情報取得 true：バックライト消灯時、キーイベントを送信しない。 false：バックライト消灯時、キーイベントを送信する。
	IOCTL_SETBACKLIGHTOFF_NOKEYEVENT	バックライト消灯時、キーイベントを送信するかどうかを設定します。 （バックライト消灯時の誤操作防止のため） 【返り値】 0 以上：成功 0 より小さい：失敗 【入力】 int 型：ファイルディスクリプタ unsigned long 型：コントロールコード bool 型ポインタ:設定情報を指定するバッファのポインタ true：バックライト消灯時、キーイベン トを送信しない。 false：バックライト消灯時、キーイベン トを送信する。 【出力】 なし

7.4.14　スイッチ仕様

対象機種
EMP-7W

デバイスファイル：/sys/class/gpio/Emergency_Stop/value

機能	仕様
open	Linux 標準ドライバの仕様に準拠
close	
read	open 時の第 2 引数の flag は、O_RDONLY を指定して下さい。 デバイスファイルを open したまま、read を繰り返して実行するときは、read 前に必ず lseek(fd, 0, SEEK_SET); を実行して、seek ポインタを先頭に指定して下さい。 fd：open で取得したファイルディスクリプタ read では、スイッチの状態が“0”・“1”で取得できます。 0：OFF 1:ON

デバイスファイル：/sys/class/gpio/EnableSwitch-Push/value

機能	仕様
open	Linux 標準ドライバの仕様に準拠
close	
read	open 時の第 2 引数の flag は、O_RDONLY を指定して下さい。 デバイスファイルを open したまま、read を繰り返して実行するときは、read 前に必ず lseek(fd, 0, SEEK_SET); を実行して、seek ポインタを先頭に指定して下さい。 fd：open で取得したファイルディスクリプタ read では、スイッチの状態が“0”・“1”で取得できます。

	0：OFF 1:ON

デバイスファイル：/sys/class/gpio/EnableSwitch-Return/value

機能	仕様
open	Linux 標準ドライバの仕様に準拠
close	
read	open 時の第 2 引数の flag は、O_RDONLY を指定して下さい。 デバイスファイルを open したまま、read を繰り返して実行するときは、read 前に必ず lseek(fd, 0, SEEK_SET); を実行して、seek ポインタを先頭に指定して下さい。 fd：open で取得したファイルディスクリプタ read では、スイッチの状態が“0”・“1”で取得できます。 0：OFF 1:ON

デバイスファイル：/sys/class/gpio/SelectSwitch_NC/value

機能	仕様
open	Linux 標準ドライバの仕様に準拠
close	
read	open 時の第 2 引数の flag は、O_RDONLY を指定して下さい。 デバイスファイルを open したまま、read を繰り返して実行するときは、read 前に必ず lseek(fd, 0, SEEK_SET); を実行して、seek ポインタを先頭に指定して下さい。 fd：open で取得したファイルディスクリプタ read では、スイッチの状態が“0”・“1”で取得できます。 0：OFF 1:ON

デバイスファイル：/sys/class/gpio/SelectSwitch_NO/value

機能	仕様
open	Linux 標準ドライバの仕様に準拠
close	
read	open 時の第 2 引数の flag は、O_RDONLY を指定して下さい。 デバイスファイルを open したまま、read を繰り返して実行するときは、read 前に必ず lseek(fd, 0, SEEK_SET); を実行して、seek ポインタを先頭に指定して下さい。 fd：open で取得したファイルディスクリプタ read では、スイッチの状態が“0”・“1”で取得できます。 0：OFF 1:ON

7.5　アプリケーション

7.5.1　　EMG ランチャー

電源投入後、自動的に EMG ランチャーが起動します。ここからシステム設定ツールを起動したり、お客様の作成したアプリケーションを登録して起動したりすることができます。
出荷時にはファイルマネージャ/システム設定/ISApp 設定 3 つのアプリケーションが登録されています。

実行ファイル

/usr/bin/emg_launcher

基本操作

番号	項目	内容
①	タイトルバー	現在表示されている画面のタイトルを表示します。
②	メニュー画面	ボタンをタップすると対応するアプリケーションが起動します。
③	戻るボタン	前の画面に戻るボタンです。ホームメニューでは押しても動作しません。
④	ホームボタン	ホームメニューで押すと、最初のページに移動します。
⑤	終了ボタン	EMG ランチャーを終了します。

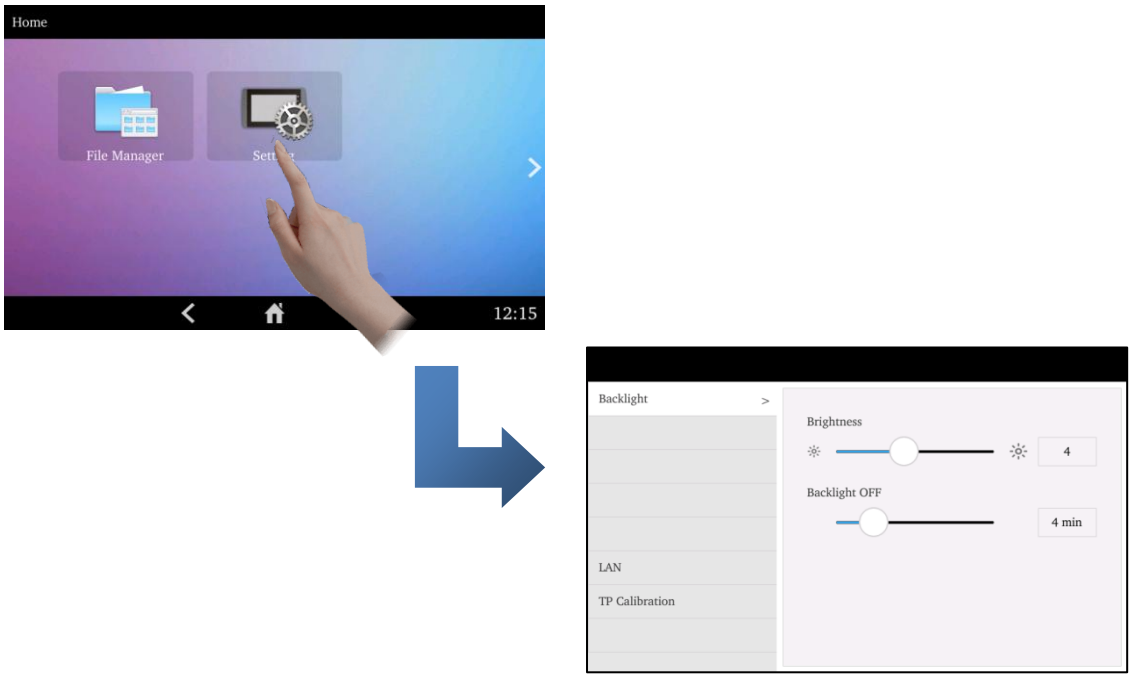
番号	項目	内容
⑥	次のページ	次のページを表示します。
⑦	時計	現在時刻を表示します。

アプリケーション一覧

アイコン	項目名(日本語)	項目名(英語)	内容
	ファイルマネージャ	File Manager	ファイルマネージャです。EMG7-Linux 内に保存されているファイルを表示・操作します。
	システム設定	Setting	本機の IP 設定や時刻設定などを行います。
	ISApp 設定	ISAppSetting	ISApp の起動方法や通信設定などを行います。

アプリケーション起動方法

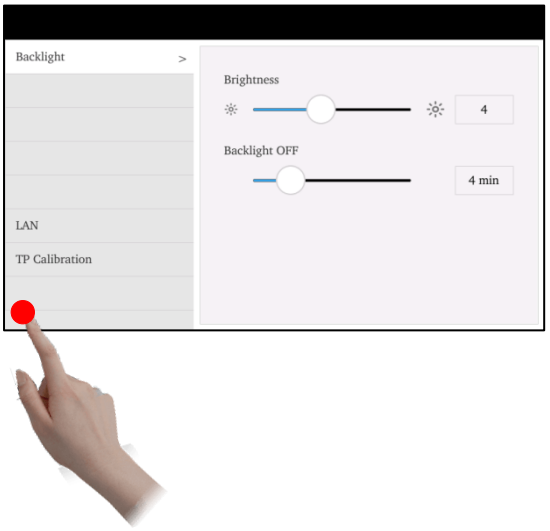
アイコンをタッチすると、登録されたアプリケーションが起動します。



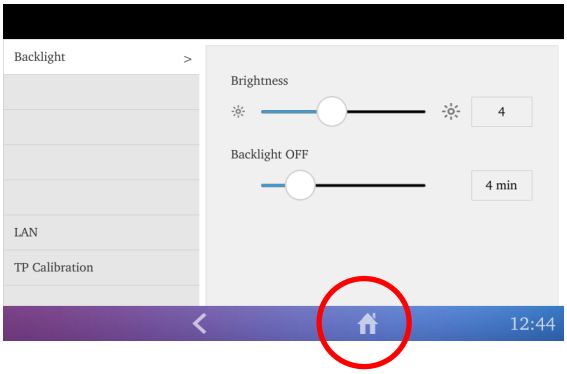
アプリケーションの終了方法

以下の操作を行うとタスクバーが表示されます。

左下隅を長押しする（２秒以上）



 をタッチすると起動中のアプリケーションが終了します。



 をタッチするとタスクバーが非表示になります。

アプリケーション登録方法

以下のフォルダにデスクトップエントリファイルを追加することで、EMG ランチャーに登録することができます。
再起動後に反映されます。

フォルダパス	/etc/emg_launcher/applications/

デスクトップエントリファイルとは、ホームメニューに表示されるボタンの情報を記載したデータファイルです。テキストエディタなどで作成します。

［共通設定］

値	説明
Name	アプリケーション名としてボタンに表示されるテキストです。（デフォルト）
Name[ja_JP]	アプリケーション名としてボタンに表示されるテキストです。（日本語ロケール時）
Exec	ボタンをタップした時に実行されるコマンドです。%f, %F, %u, %U 等の指定は無視されます。
Icon	ボタンに表示されるアイコンです。
Type	Application を指定して下さい。

※ １行目は[Desktop Entry]と記載して下さい。

※ ファイルの拡張子は「.desktop」である必要があります。

※ 文字コードは「UTF-8」で保存して下さい。

※ これ以外の値が設定されていた場合は無視されます。

例：

```
[Desktop Entry]
Name=Setting
Name[ja_JP]=システム設定
Exec=/usr/bin/emg_setting
Icon=/etc/emg_launcher/icons/menu_tablet_settings.png
Type=Application
```

アプリケーション削除方法

以下のフォルダのデスクトップエントリファイルを削除することで、EMG ランチャーから削除することができます。

再起動後に反映されます。

フォルダパス	/etc/emg_launcher/applications/

7.5.2 VNC サーバ

スタートメニューの[インターネット]-[x11VNC Server]またはコマンド入力の/usr/bin/x11vnc 起動します。
VNC クライアントがインストールされている PC 等からリモートで接続し、表示操作を行うことができます。

実行ファイル

/usr/bin/x11vnc

オプション

実行時のオプションについては以下のコマンドでご確認ください。

/usr/bin/x11vnc --help

7.5.3 VNC クライアント

コマンド入力の/usr/bin/vncviewer 起動します。
VNC サーバがインストールされている PC 等にリモートで接続し、表示操作を行うことができます。

実行ファイル

/usr/bin/vncviewer

オプション

実行時のオプションについては以下のコマンドでご確認ください。

/usr/bin/vncviewer -help

7.6 API

7.6.1 DIO API

対象機種
EM(G)8-4
EM(G)8-5

DIO インタフェースを制御するための API です。

ライブラリファイル

libem_dio.so

ヘッダファイル

em_dio.h
em_dio-c.h
em_dio-c++.h
em_types.h

ライブラリファイル、ヘッダファイルは DVD-ROM（開発環境一式）に格納されています。開発環境にコピーして下さい。

定数

動作モード

EMDIO_MODE_SHEETKEY	0	シートキーモード
EMDIO_MODE_DIO_SCAN	1	DIO SCAN ありモード
EMDIO_MODE_DIO_NON_SCAN	2	DIO SCAN なしモード

参考）各動作モードでの DIN、DOUT 数

	DIN 数	DOUT 数
DIO SCAN ありモード	24	8
DIO SCAN なしモード	6	12
シートキーモード	-	8

DOUT 出力

EMDIO_DOUT_OFF	0	オフ
EMDIO_DOUT_ON	1	オン

DIN 入力

EMDIO_DIN_OFF	0	オフ
EMDIO_DIN_ON	1	オン

エラーコード

ERROR_CODE_SUCCESS	0x00000000	成功
ERROR_CODE_INVALID_PARAMETER	0x20000001	パラメータ不正
ERROR_CODE_LIB_NOT_OPEN	0x20000002	ライブラリオープン前の関数呼び出し
ERROR_CODE_LIB_OPEN_FAILURE	0x20000011	ライブラリオープン失敗
ERROR_CODE_LIB_CLOSE_FAILURE	0x20000012	ライブラリクローズ失敗
ERROR_CODE_INVALID_MODE	0x20000013	動作モードエラー
ERROR_CODE_DRIVER_INTERNAL	0x20000021	ドライバ内部エラー

API(C++用)

＜ライブラリオープン＞

関数	int OpenLib(int mode)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード
返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_INVALID_PARAMETER ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_DRIVER_INTERNAL
機能	ライブラリをオープンし、使用できるようにします。

注意）ライブラリを操作する前に OpenLib を呼び出す必要があります。

＜ライブラリクローズ＞

関数	int CloseLib()
引数	(無し)

返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_CLOSE_FAILURE
機能	ライブラリをクローズします。

＜動作モード取得＞

関数	int GetMode(int* mode)
引数	mode: 動作モードの取得領域のポインタを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード
返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIO の動作モードを取得します。

＜DOUT 出力＞

関数	int SetDout(int num, int set)
引数	num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 set： DOUT 出力を指定する [DOUT 出力]（負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL

機能	DOUT 出力を設定します。

＜DOUT 状態取得＞

関数	int GetDout(int num, int* val)
引数	num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 val: DOUT 状態の取得領域のポインタを指定する [DOUT 出力] （負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DOUT 状態を取得します。

＜DIN 状態一括取得＞

関数	int GetDinAll(DWORD* val)
引数	val: DIN 状態の取得領域のポインタを指定する DIO SCAN ありモード： bit31～bit24：予備（0 固定）、bit23～bit0：DIN24～DIN1 DIO SCAN なしモード： bit31～bit6：予備（0 固定）、bit5～bit0：DIN6～DIN1 [DIN 入力] （負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を一括で取得します。

注意）動作モードがシートキーモードの場合は、返り値がエラーになります。

＜DIN 状態個別取得＞

関数	int GetDin(int num, int* val)
引数	num: DIN 番号を指定する [DIN 番号] DIO SCAN ありモード：1～24 DIO SCAN なしモード：1～6 val: DIN 状態の取得領域のポインタを指定する [DIN 入力] （負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を個別で取得します。

注意）動作モードがシートキーモードの場合は、返り値がエラーになります。

＜キーコードマップ設定＞

関数	int SetKeyMap(int keyCodeMap, int num)
引数	keyCodeMap: キーコード配列のポインタを指定する num: キーコード数を指定する [キーコード数] 24(固定)
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	キーコードマップを設定します。

注意）動作モードがシートキーモードでない場合は、返り値がエラーになります。

キーコード配列

int 型	keyCodeMap[24]	キーコード配列

キーコード配置との対応

	RETURN LINE1	RETURN LINE2	RETURN LINE3	RETURN LINE4	RETURN LINE5	RETURN LINE6
SCAN LINE1	keyCodeMap[0]	keyCodeMap[1]	keyCodeMap[2]	keyCodeMap[3]	keyCodeMap[4]	keyCodeMap[5]
SCAN LINE2	keyCodeMap[6]	keyCodeMap[7]	keyCodeMap[8]	keyCodeMap[9]	keyCodeMap[10]	keyCodeMap[11]
SCAN LINE3	keyCodeMap[12]	keyCodeMap[13]	keyCodeMap[14]	keyCodeMap[15]	keyCodeMap[16]	keyCodeMap[17]
SCAN LINE4	keyCodeMap[18]	keyCodeMap[19]	keyCodeMap[20]	keyCodeMap[21]	keyCodeMap[22]	keyCodeMap[23]

＜DOUT 出力（OpenLib、CloseLib 手順不要）＞

関数	int SetDoutDirect(int mode, int num, int set)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 set： DOUT 出力を指定する [DOUT 出力]（負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	DOUT 出力を設定します。

注意）本関数は、OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DOUT 状態取得（OpenLib、CloseLib 手順不要）＞

関数	int GetDoutDirect(int mode, int num, int* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 val: DOUT 状態の取得領域のポインタを指定する [DOUT 出力]（負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DOUT 状態を取得します。

注意）本関数は、OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DIN 状態一括取得（OpenLib、CloseLib 手順不要）＞

関数	int GetDinAllDirect(int mode, DWORD* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード val: DIN 状態の取得領域のポインタを指定する DIO SCAN ありモード： bit31～bit24：予備（0 固定）、bit23～bit0：DIN24～DIN1 DIO SCAN なしモード： bit31～bit6：予備（0 固定）、bit5～bit0：DIN6～DIN1 [DIN 入力]（負論理） EMDIO_DIN_OFF：オフ

	EMDIO_DIN_ON：オン
返り値	ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を一括で取得します。

注意 1）動作モードにシートキーモードを指定した場合は、返り値がエラーになります。

注意 2）本関数は、OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DIN 状態個別取得（OpenLib、CloseLib 手順不要）＞

関数	int GetDinDirect(int mode, int num, int* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DIN 番号を指定する [DIN 番号] DIO SCAN ありモード：1～24 DIO SCAN なしモード：1～6 val: DIN 状態の取得領域のポインタを指定する [DIN 入力]（負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を個別で取得します。

注意 1）動作モードにシートキーモードを指定した場合は、返り値がエラーになります。

注意 2）本関数は、OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜ライブラリバージョン取得＞

関数	string GetLibVersion()
引数	（無し）

返り値	バージョン文字列 [ライブラリバージョン] string ver : ”1.0.0”(半角5文字)
機能	ライブラリのバージョンを取得します。

注意）本関数は、OpenLibなしで呼び出せます。

API(C用)

＜ライブラリオープン(C言語用)＞

関数	int EmDio_OpenLib(int mode)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCANありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCANなしモード
返り値	[エラーコード] 成功 : ERROR_CODE_SUCCESS 失敗 : ERROR_CODE_INVALID_PARAMETER ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_DRIVER_INTERNAL
機能	ライブラリをオープンし、使用できるようにします。

注意）ライブラリを操作する前にEmDio_OpenLibを呼び出す必要があります。

＜ライブラリクローズ(C言語用)＞

関数	int EmDio_CloseLib()
引数	(無し)
返り値	[エラーコード] 成功 : ERROR_CODE_SUCCESS 失敗 : ERROR_CODE_LIB_CLOSE_FAILURE
機能	ライブラリをクローズします。

＜動作モード取得(C 言語用)＞

関数	int EmDio_GetMode(int* mode)
引数	mode: 動作モードの取得領域のポインタを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード
返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIO の動作モードを取得します。

＜DOUT 出力(C 言語用)＞

関数	int EmDio_SetDout(int num, int set)
引数	num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 set： DOUT 出力を指定する [DOUT 出力] （負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	[エラーコード] 成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	DOUT 出力を設定します。

＜DOUT 状態取得(C 言語用)＞

関数	int EmDio_GetDout(int num, int* val)
引数	num: DOUT 番号を指定する [DOUT 番号]

	シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 val: DOUT 状態の取得領域のポインタを指定する [DOUT 出力] （負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DOUT 状態を取得します。

＜DIN 状態一括取得(C 言語用)＞

関数	int EmDio_GetDinAll(DWORD* val)
引数	val: DIN 状態の取得領域のポインタを指定する DIO SCAN ありモード： bit31～bit24：予備 (0 固定)、bit23～bit0：DIN24～DIN1 DIO SCAN なしモード： bit31～bit6：予備 (0 固定)、bit5～bit0：DIN6～DIN1 [DIN 入力] （負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を一括で取得します。

注意）動作モードがシートキーモードの場合は、返り値がエラーになります。

＜DIN 状態個別取得(C 言語用)＞

関数	int EmDio_GetDin(int num, int* val)
引数	num: DIN 番号を指定する [DIN 番号] DIO SCAN ありモード：1～24 DIO SCAN なしモード：1～6

	val: DIN 状態の取得領域のポインタを指定する [DIN 入力] （負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を個別で取得します。

注意）動作モードがシートキーモードの場合は、返り値がエラーになります。

＜キーコードマップ設定(C 言語用)＞

関数	int EmDio_SetKeyMap(int keyCodeMap, int num)
引数	keyCodeMap: キーコード配列のポインタを指定する num: キーコード数を指定する [キーコード数] 24(固定)
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_NOT_OPEN ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	キーコードマップを設定します。

注意）動作モードがシートキーモードでない場合は、返り値がエラーになります。

キーコード配列

int 型	keyCodeMap[24]	キーコード配列

キーコード配置との対応

	RETURN LINE1	RETURN LINE2	RETURN LINE3	RETURN LINE4	RETURN LINE5	RETURN LINE6
SCAN LINE1	keyCodeMap[0]	keyCodeMap[1]	keyCodeMap[2]	keyCodeMap[3]	keyCodeMap[4]	keyCodeMap[5]
SCAN LINE2	keyCodeMap[6]	keyCodeMap[7]	keyCodeMap[8]	keyCodeMap[9]	keyCodeMap[10]	keyCodeMap[11]
SCAN LINE3	keyCodeMap[12]	keyCodeMap[13]	keyCodeMap[14]	keyCodeMap[15]	keyCodeMap[16]	keyCodeMap[17]

SCAN LINE4	keyCodeMap[18]	keyCodeMap[19]	keyCodeMap[20]	keyCodeMap[21]	keyCodeMap[22]	keyCodeMap[23]

＜DOUT 出力（EmDio_OpenLib、EmDio_CloseLib 手順不要）(C 言語用)＞

関数	int EmDio_SetDoutDirect(int mode, int num, int set)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DOUT 番号を指定する [DOUT 番号] シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 set： DOUT 出力を指定する [DOUT 出力]（負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	DOUT 出力を設定します。

注意）本関数は、EmDio_OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DOUT 状態取得（EmDio_OpenLib、EmDio_CloseLib 手順不要）(C 言語用)＞

関数	int EmDio_GetDoutDirect(int mode, int num, int* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_SHEETKEY: シートキーモード EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DOUT 番号を指定する [DOUT 番号]

	シートキーモード：1～8 DIO SCAN ありモード：1～8 DIO SCAN なしモード：1～12 val: DOUT 状態の取得領域のポインタを指定する [DOUT 出力] （負論理） EMDIO_DOUT_OFF：オフ EMDIO_DOUT_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DOUT 状態を取得します。

注意）本関数は、EmDio_OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DIN 状態一括取得（EmDio_OpenLib、EmDio_CloseLib 手順不要）(C 言語用)＞

関数	int EmDio_GetDinAllDirect(int mode, DWORD* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード val: DIN 状態の取得領域のポインタを指定する DIO SCAN ありモード： bit31～bit24：予備（0 固定)、bit23～bit0：DIN24～DIN1 DIO SCAN なしモード： bit31～bit6：予備（0 固定)、bit5～bit0：DIN6～DIN1 [DIN 入力] （負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を一括で取得します。

注意 1）動作モードにシートキーモードを指定した場合は、返り値がエラーになります。

注意 2）本関数は、EmDio_OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜DIN 状態個別取得（EmDio_OpenLib、EmDio_CloseLib 手順不要）(C 言語用)＞

関数	int EmDio_GetDinDirect(int mode, int num, int* val)
引数	mode: 動作モードを指定する [動作モード] EMDIO_MODE_DIO_SCAN: DIO SCAN ありモード EMDIO_MODE_DIO_NON_SCAN: DIO SCAN なしモード num: DIN 番号を指定する [DIN 番号] DIO SCAN ありモード：1～24 DIO SCAN なしモード：1～6 val: DIN 状態の取得領域のポインタを指定する [DIN 入力]（負論理） EMDIO_DIN_OFF：オフ EMDIO_DIN_ON：オン
返り値	成功： ERROR_CODE_SUCCESS 失敗： ERROR_CODE_LIB_OPEN_FAILURE ERROR_CODE_INVALID_PARAMETER ERROR_CODE_INVALID_MODE ERROR_CODE_DRIVER_INTERNAL
機能	現在の DIN 状態を個別で取得します。

注意 1）動作モードにシートキーモードを指定した場合は、返り値がエラーになります。

注意 2）本関数は、EmDio_OpenLib なしで呼び出せます。関数内でドライバ獲得、解放を行いますので、関数呼び出しごとに処理負荷が生じることに注意してください。

＜ライブラリバージョン取得(C 言語用)＞

関数	char* EmDio_GetLibVersion()
引数	（無し）
返り値	バージョン文字列 [ライブラリバージョン] char ver[6]：”1.0.0”（半角 5 文字）
機能	ライブラリのバージョンを取得します。

注意）本関数は、EmDio_OpenLib なしで呼び出せます。

お問い合わせ

本ドキュメントに関するお問い合わせは、下記へお願い致します。

お電話でのお問い合わせ

☎ 06-6147-6645

株式会社ディ・エム・シー　大阪技術センター
受付時間：平日　9：00～17：00
※土日・祝祭日・年末年始を除く

メールでのお問い合わせ

お問い合わせフォームで受け付けています。下記からご連絡ください。

✉ www.dush.co.jp/contact/

よくあるご質問と回答集

🌐 www.dush.co.jp/support/faq/

2025 年 7 月　第 11 版
発行所　株式会社ディ・エム・シー
〒108-0074　東京都港区高輪 2-18-10 高輪泉岳寺駅前ビル 11F
TEL：(03)-6721-6731 ㈹　FAX：(03)-6721-6732
URL：https://www.dush.co.jp/