TWE-NET SDK マニュアル

(2016-12.1)

モノワイヤレス株式会社

バージョン

- 2016/12/12: モノワイヤレス株式会社の記述に変更
- 2015/07/23: GIT の記載について修正
- 2014/06/20: 2014年6月版
- 2014/04/3: 2014年4月版
- 2013/11/5: 2013年11月版
  2013/9/5: 2013年9月版
- 2013/6/11 : TWE-Lite 追加サポート、βリリース

2016年12月版資料の変更点

- ソフトウェア・ライセンスについて設定します
  - 東京コスモス電機株式会社の著作表示のあるソースコードなどは、モノワイヤレス株式会社への著作権の移転が行われました。
  - 新たに「モノワイヤレスソフトウェア使用許諾バージョン1(識別子: MW-SLA-1J) を設定します。過去に配布したソースコードなどについては、当使用許諾を順次当ライセンスの明記を行います。
    - 当ライセンスの英文表記は「MONO WIRELESS SOFTWARE LIECENSE AGREEMENT VERSION 1 (MW-SLA-1E)」です。
    - https://mono-wireless.com/jp/products/TWE-NET/license.html を参照ください。
  - 著作権表示並びにライセンス記述について上記の更新が為されていないものについては、原則として MW-SLA-1J を準用します。
  - ライセンス上の不明点がある場合は、当技術サポート窓口へのご連絡をお願い致します。
- TWE-Regular/Strong の記述について、原則として削除しています。
- ToCoNet を TWE-NET と名称変更しました。機能や関数名などに変更はありません。

2013年11月版の変更点

- Eclipse のバージョンを Kepler.SR2 に変更しました。Kepler のリリースの
- make 時に Version.h の自動生成が省略されました。
  替わりにコンパイルオプションで VERSION_MAIN, VERSION_SUB, VERSION_VAR が定義され、Version.h に記述があると複数定義エラーになる場合があります。

  対応 → Version.h をインクルードしないか、Version.h の中身を空に編集して下さい。

本SDK対象モジュールについて

- 本SDKの対象モジュールは以下となります。
  - TWE-LITE (TWE-LITE DIP, TWE-LITE 2525A, MoNoStick を含む)

SDKの利用規定

- 本SDKの利用は以下を原則とします。
  - 本SDKの添付物（ソースコード、ライブラリ、資料）については、TWEシリーズ上のマイコンで実行するために利用する限り、商用利用を含み制限なく無償で利用できます。（MW-SLA-1J を参照ください）
  - 各サンプルソースコード中に別途規定が記載される場合があります。
  - SDK 添付物を流用しお客様が改変した成果物に対して開示義務は、発生しません。
  - お客様がソースコードを改変して、これをウェブ上等に公開することは可能ですが、公開時にモノワイヤレス株式会社にご一報ください。（ソースコード中の各ライセンス記述をご参照ください）
  - SDK 添付物は無保証です。
- TWE-NET (ToCoNet) ライブラリについて
  - TWE-NET ライブラリは無保証です。
  - TWE-NET ライブラリのバージョン間の互換性は保証しません。
  - モノワイヤレス株式会社は TWE-NET ライブラリのソースコード開示義務を負いません。
- 要望・問題の対応について
  - 当SDKは無保証が原則ですが、要望の依頼、問題の報告はモノワイヤレス株式会社の技術問い合わせ窓口を通じて実施します。
  - モノワイヤレス株式会社は、問題の修正を含む、期日を設定したあらゆる行動の義務を負いません。

サポートについて

- 本SDKのサポートについては、サポート契約のあるお客様に対して電子メールにて実施します。
- サポート契約の無いお客様は、当社ウェブサイト上の問い合わせ窓口経由でご連絡ください。
  - ただし返答のお約束は出来ません。

SDKの構成

- ライブラリ
  - JN5164 (TWE-Lite) 開発用MAC層ライブラリ
- ツールチェイン
  - GCC, Make など
- 統合環境
  - Eclipse Kepler (Pleiades プロジェクトより)
  - ドキュメントツール (doxygen, Graphviz ツール)
- ファームウェア書き込みツール
  - flashprogrammer (半導体ベンダ純正)
  - jenprog (Python インタプリタによる実装)
  - TWE-Programmer (TWE-Lite 専用の書き込みツール, Windows専用)
  - TWE_PGM_MODE (ToCoStick, TWE-Lite-R 用、プログラムモード設定ツール, Windows専用)
- スニファ
  - Wireshark によるパケットモニタ
- サンプルコード

SDKの対応範囲

- 本SDKは TWE-NET ライブラリによる無線送受信アプリケーションを構築するために使用します。
- 以下は対応しません
  - 標準の MAC 層 API による開発
    - 技術的には可能ですがサポート対応しません。MAC層の手続き自体煩雑で複雑な例外処理を行う必要があるためで、ToCoNetは一般の通信要求に満足させるためにMAC 機能を単純化しています。
  - ZigBee Pro など他のスタックライブラリによる開発
    - 本SDKの対象外です。

MONO WIRELESS

SDKのインストール

動作環境

- Windows10 32bit/64bit 環境に対応します。
- Eclipse Kepler を開発のフロントエンドに使用します。Eclipse Kepler が不都合無く動作する環境を用意してください。

SDKの解凍

- 既に古いバージョンをインストールされた方は、アプリケーション開発用のディレクトリのバックアップ (Wks_*) を取ってください。
- 提供された TWESDK_(VERINFO).zip を C:¥TWESDK に解凍します
  - (VERINFO) はバージョン識別文字列です。リリース日付が使用されます。

  ※ 必ず**最上位ディレクトリ (C:¥)** に展開してください。
  Windows のディレクトリ名を含めた最大ファイル名長の制限があり、一部ファイルが展開されず解凍エラーになる場合が有ります。

  ※ 展開先のドライブは C:¥ でなくてもかまいません。
- 更新版と記載されているアーカイブを入手した場合は、アーカイブ中の指示に従ってください。

パスの設定

- 通常は不要*ですが、Cygwin などでコマンドライン作業する場合はパスの設定をします。
  - C:¥TWESDK¥Tools¥ba-elf-ba2¥bin (/cygdrive/c/TWESDK/Tools/ba-elf-ba2/bin) の追加が必要です。
  - コマンドラインツールで必要なのは GNU make (make.exe) です。GNU make が実行されるようにシステムの PATH 設定を行ってください。他の make ツールが起動するとエラーになります。

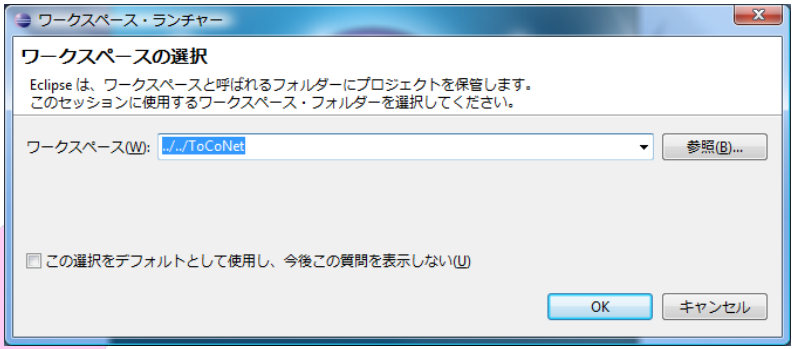
* 提供される eclipse のサンプルでは以下の設定が行われていますので、eclipse 上のプロジェクトではGNU make など必要なツールが呼び出されるようになっています。

[プロジェクトプロパティ]→[C/C++ ビルド]→[環境]→PATH

Eclipse の起動

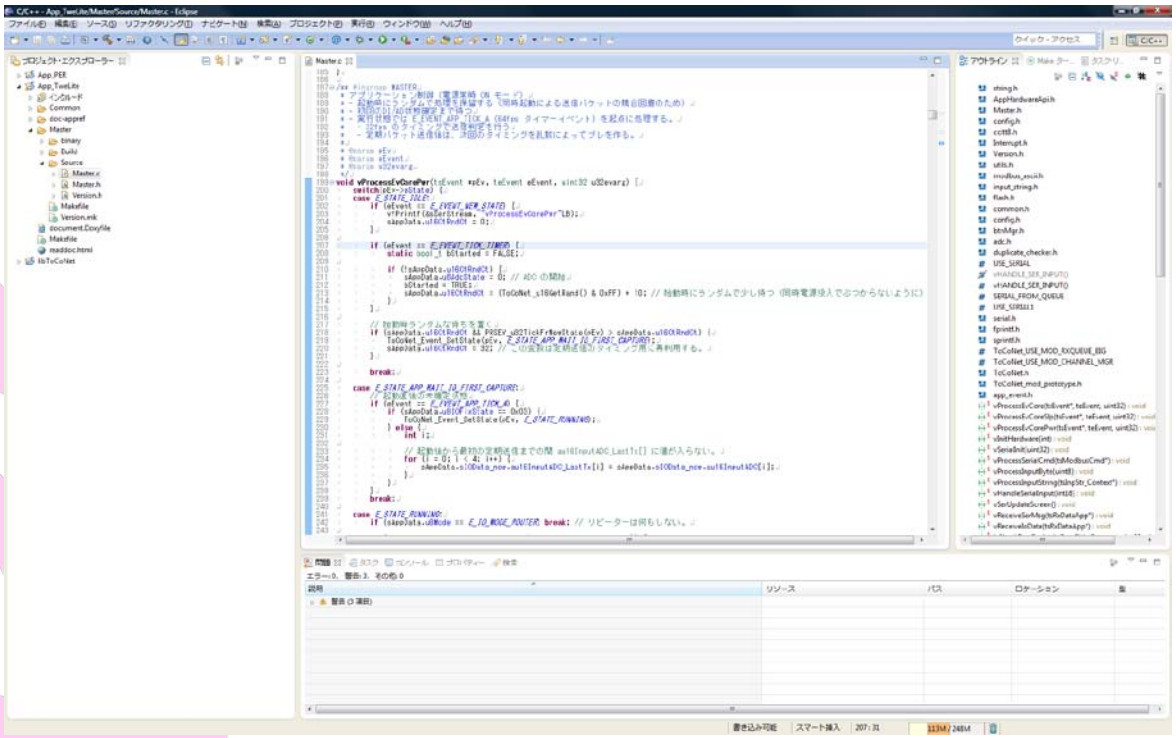
- C:¥TWESDK¥pleiades¥eclipse¥eclipse.exe を起動します。
  - C:¥TWESDK¥eclipse.cmd を実行しても構いません。

- 起動後ワークスペースの場所を聞かれますので ../../Wks_ToCoNet と指定してください (C:¥TWESDK¥Wks_ToCoNet でも可)。



Eclipse Kepler

- Pleiades プロジェクトにて日本語化された Eclipse Kepler SR2 です。
- ToCoNet ワークスペースや添付のプロジェクトには諸設定が行われています。
- 開発及びビルドに Eclipse は必須ではありませんが、本資料では Eclipse を用いた開発を紹介します。

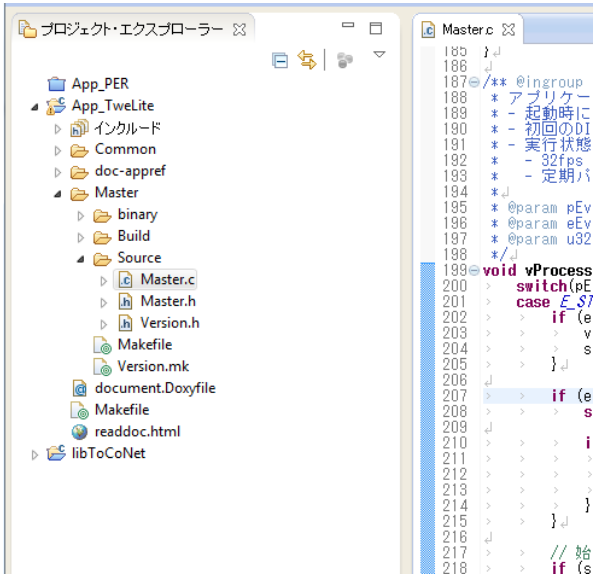


MONO WIRELESS

ビルドしてみる

ビルドするプロジェクトを選択

- プロジェクトエクスプローラにて、対象のプロジェクトのファイルを選択します。ここでは Master.c を開いています。
  ※ 青色のフォルダは、プロジェクトが開かれていない状態です。右クリック⇒[プロジェクトを開く]を選択してください。

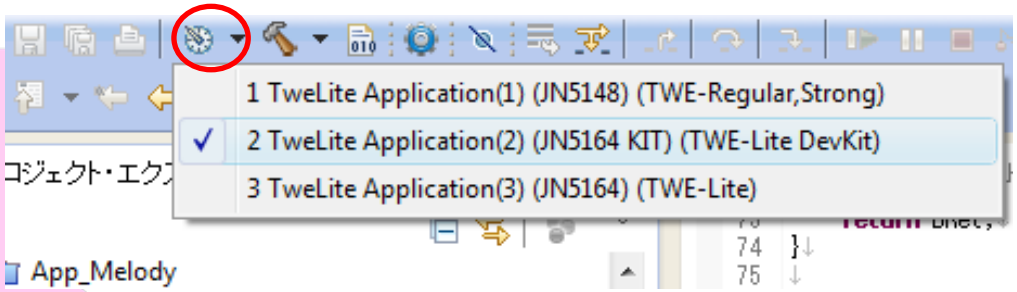


- ビルドする前にはクリーン（コンパイル中の中間ファイルを削除）します。メニューより [プロジェクト]>[クリーン...] を選択してください。
- 右図のようなダイアログが出現します。[以下で選択したプロジェクトをクリーン] をチェックし、対象プロジェクト App_TweLite を選択し [OK] ボタンを押します。

ビルドする構成を選択

- サンプルプロジェクトには、TWEモジュールに利用する半導体のモデルに応じた構成など、いくつかの構成が事前定義されています。
- 構成を選択するには、メニューより[プロジェクト]>[構成のビルド]>[アクティブにする]より目的の構成を選択するか、構成選択アイコン右の▼をクリックします。

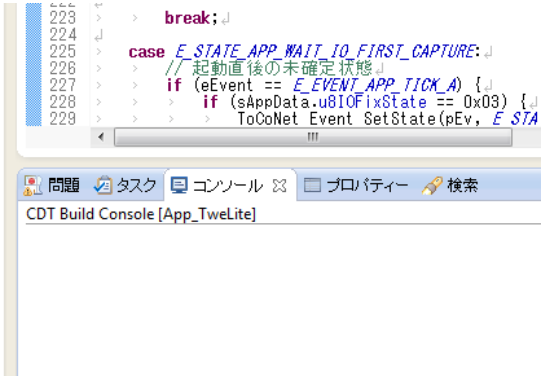
構成選択アイコン

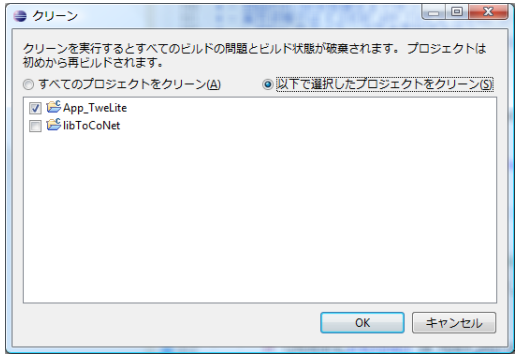


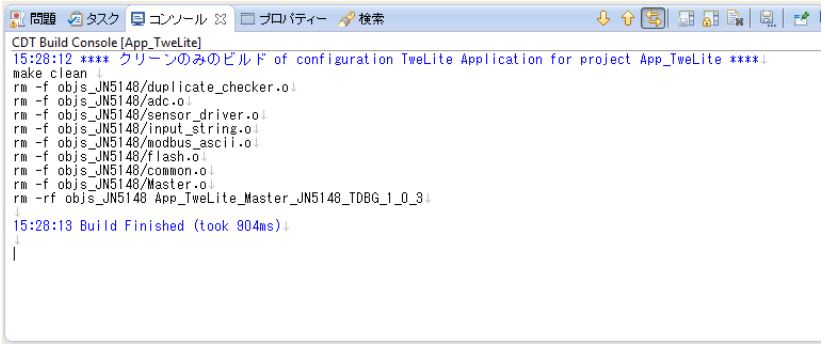
クリーン

ビルドする前には念のためクリーン(コンパイル中の中間ファイルを削除)します。

- クリーン中の動作を確認するため、コンソールタブを見えるようにして置きます。
- メニューより [プロジェクト]>[クリーン...] を選択してください。
- 右図のようなダイアログが出現します。[以下で選択したプロジェクトをクリーン] をチェックし、対象プロジェクト App_TweLite を選択します。
- [OK] ボタンを押します。





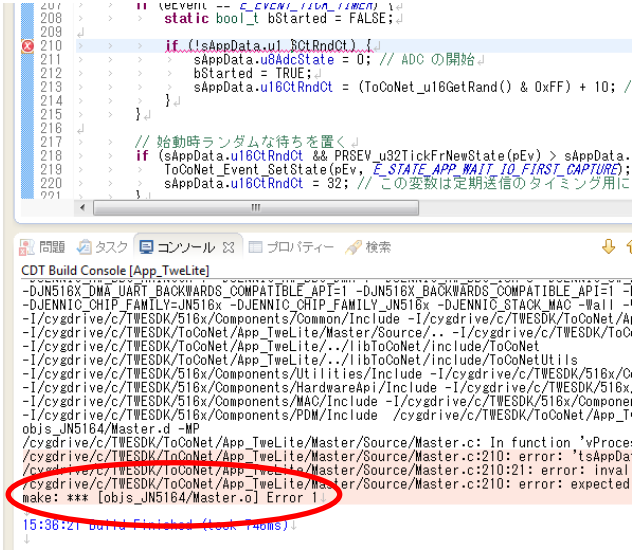


ビルド

- 続けてビルドアイコンを押します。
  ※ ここで▼によりビルドする構成を選択出来ます。
- コンソールにビルド経過が表示されます。無事にビルドが終了すれば、bin ファイルが生成されます。エラーがあるとビルドが停止します。

ビルドアイコン

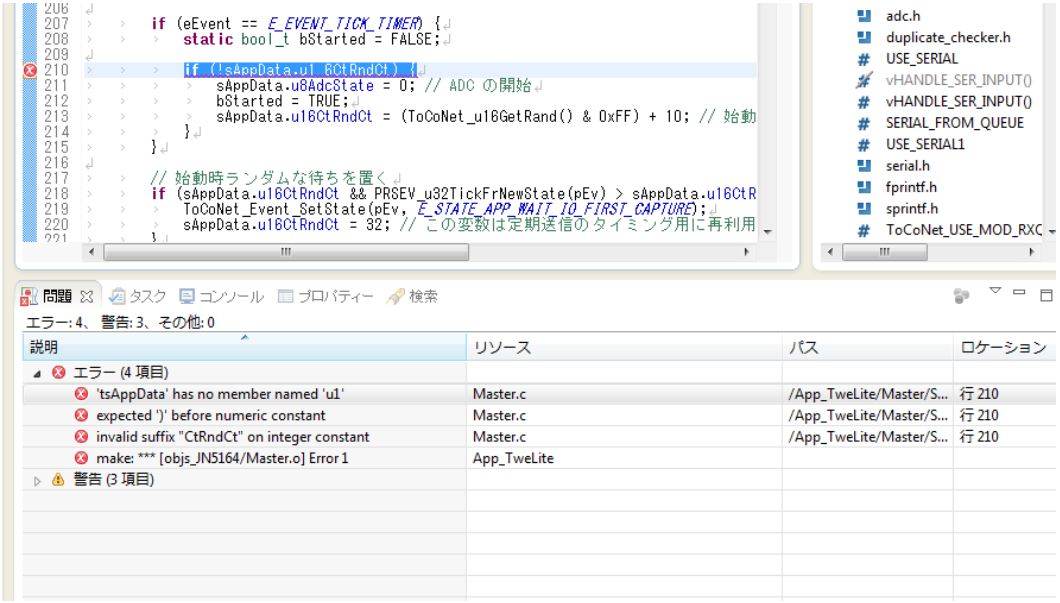
エラー発生



無事に終わった

エラーなどの表示

- ビルド時のエラーなどは[問題]タブで確認できます。
- ツール等のエラー、ワーニングが出る場合が有りますが、どうしても消せないものもあります。詳しくは【Eclipseについて】の章を参照してください。

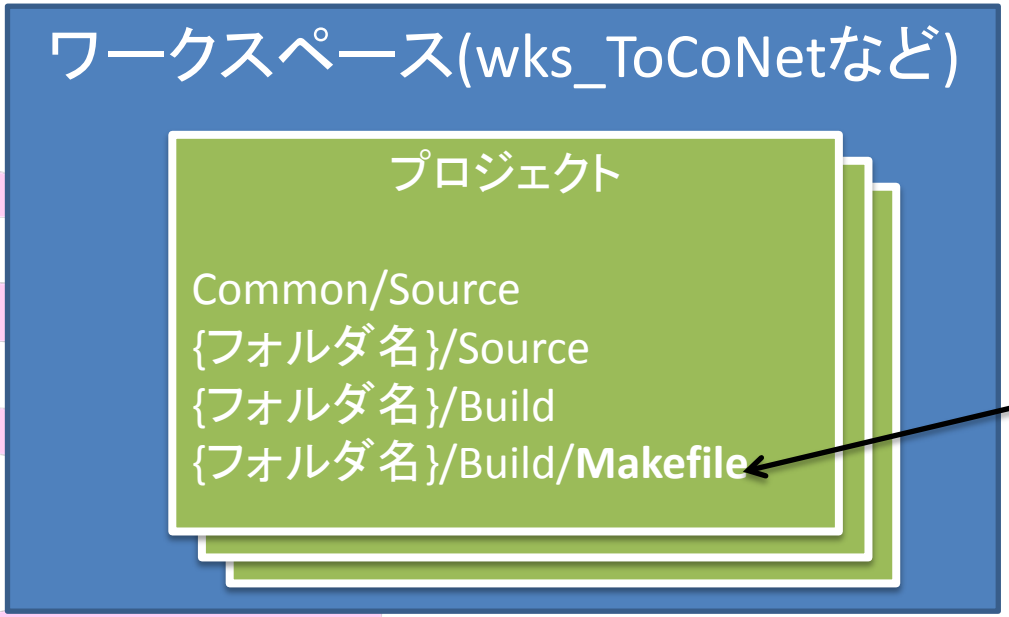


MONO WIRELESS

Eclipse について

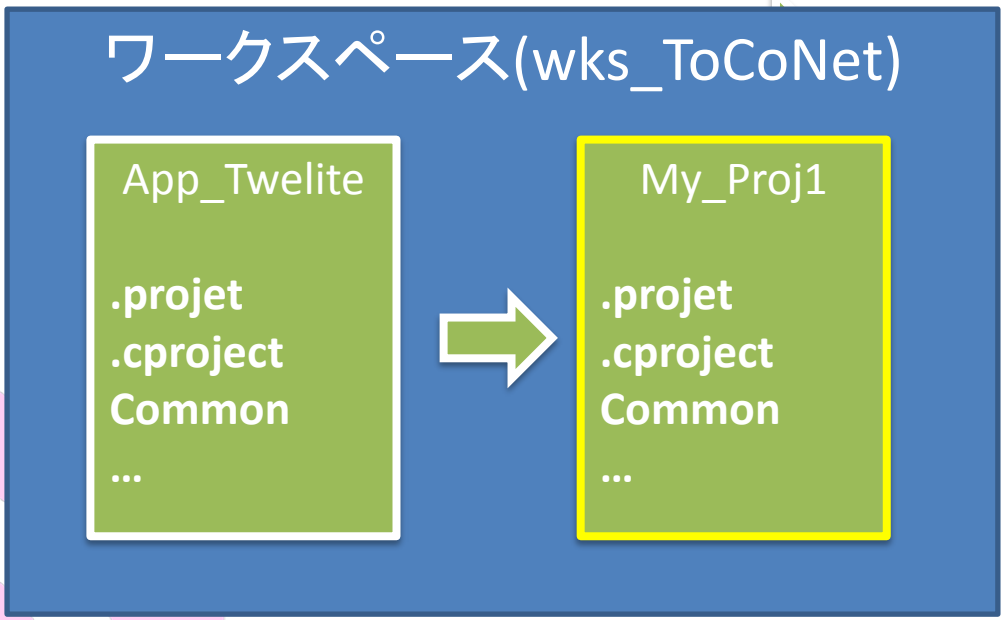
Eclipseのファイル構造

- 本SDKではEclipseをCソースコード編集エディタ、および、make呼び出しのフロントエンドとして用いています。コマンドラインでビルドしても結果は同じです。
  - ワークスペース：各プロジェクトをまとめた定義。
    - 本SDK専用の定義も行われているので新たにワークスペースを作成する時は既存のワークスペースをコピーして使用してください。
  - プロジェクト：ソースコードやビルド定義をまとめたもの。
    - 本SDKでは下図に示したフォルダ階層が基本になっており、特にBuildディレクトリの階層を変更できません。(MakefileはBuildディレクトリからの相対パスで管理されているため)



新しいプロジェクトの追加(1)

- 新しいプロジェクトは、必ず既存のサンプルプロジェクトのディレクトリを丸ごとをコピーして利用します。



ディレクトリをまるごとコピー

新しいプロジェクトの追加(2)

- {プロジェクトディレクトリ}/.project の <name>, <project> の部分をテキストエディタで編集し新しいプロジェクト名に変更してください。後述のインポート時にインポートできないことが有ります。

My_Proj1

.projet
.cproject
Common
…

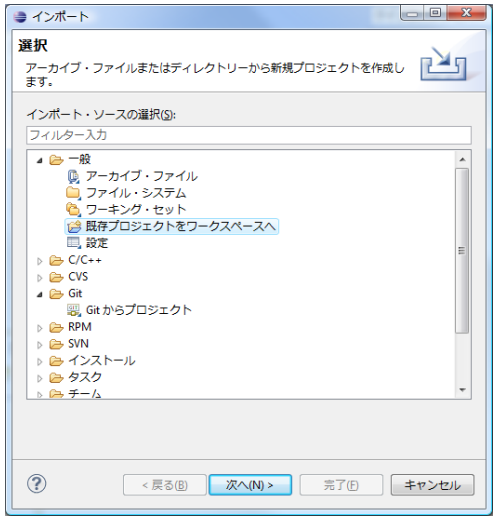
```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
	<name>App_TweLite</name>
	<comment></comment>
	<projects>
		<project>App_TweLite</project>
	</projects>
…
```

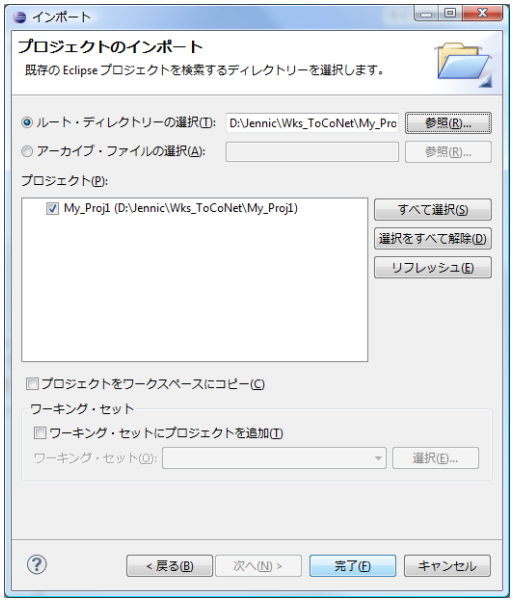
```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
	<name>My_Proj1</name>
	<comment></comment>
	<projects>
		<project>My_Proj1</project>
	</projects>
…
```

新しいプロジェクトの追加(3)

- Eclipse を立ち上げて、対象のワークスペースを開きます。
- メニューより [ファイル]→[インポート] を選択します。
- インポートダイアログ・選択より [一般]→[既存プロジェクトをワークスペースへ]を選択します。
- インポートダイアログ・プロジェクトへのインポートよりルートディレクトリの選択します。
- プロジェクト欄にインポート対象のプロジェクト名が表示されている事を確認して、[完了] ボタンを押します。
- プロジェクトが追加されたことを確認してください。

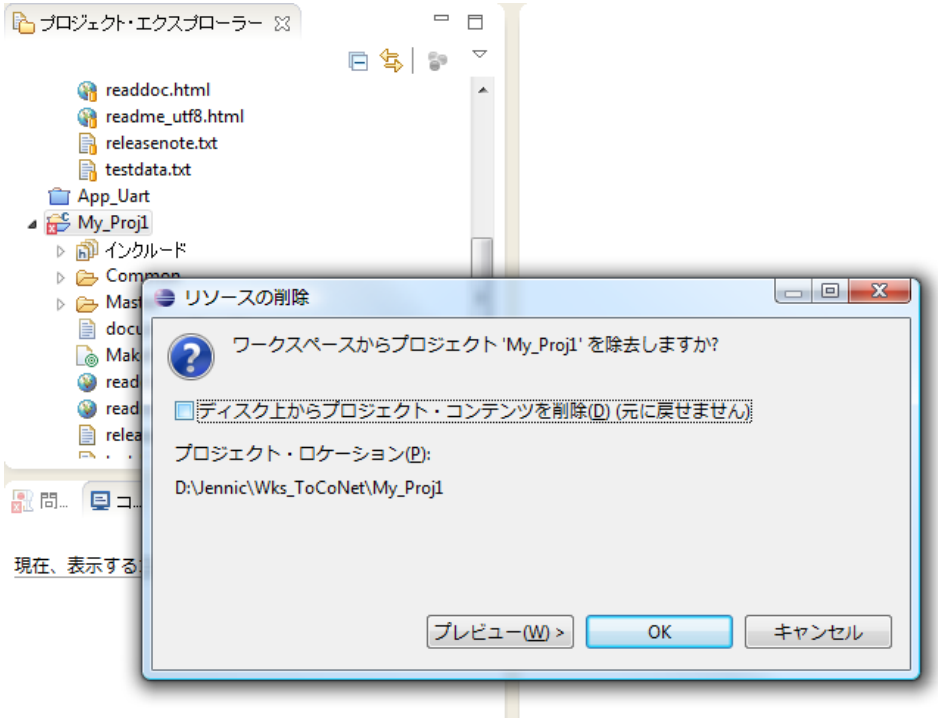
※ Git を利用する場合は、[Git からプロジェクト]を選択します。





プロジェクトの削除

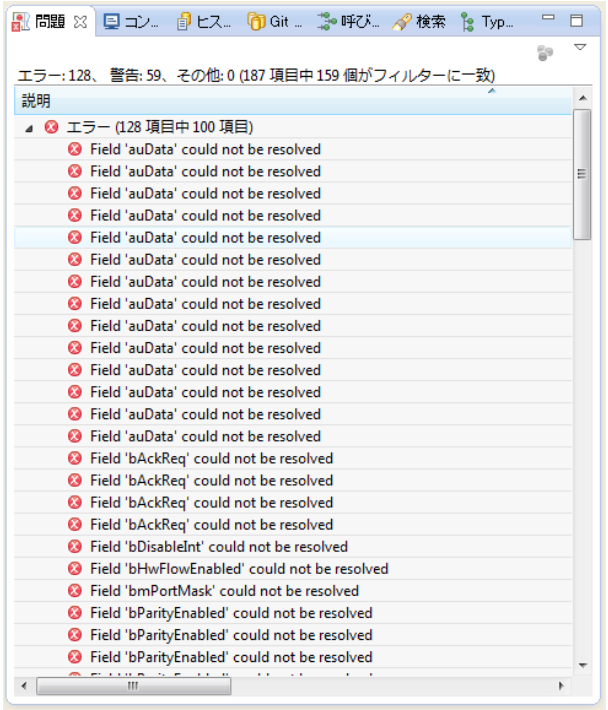
- プロジェクトエクスプローラから対象プロジェクトを選択し、[DEL]キーを押します。
- [リソースの削除]ダイアログが出現しますので [OK] を押してください。
  - [ディスク上からプロジェクトコンテンツを削除] を選択しなければ、ディレクトリはそのまま残ります。



※ Eclipse 上でのワークスペースの操作に問題が起きたような場合（コード解釈にどうしても問題が有る、ファイルのアイコン表示がおかしいなど）に、プロジェクトの削除と再追加を行えば回復する場合が有ります。

Eclipse のコード解釈 (1)

- XXX not resolved といったエラーが表示される場合は eclipse のコード解釈の問題が考えられます。
- 前述のビルド手順で、コンソールログにエラーが表示されていなければ、**コンパイルは無事に終了**しています。

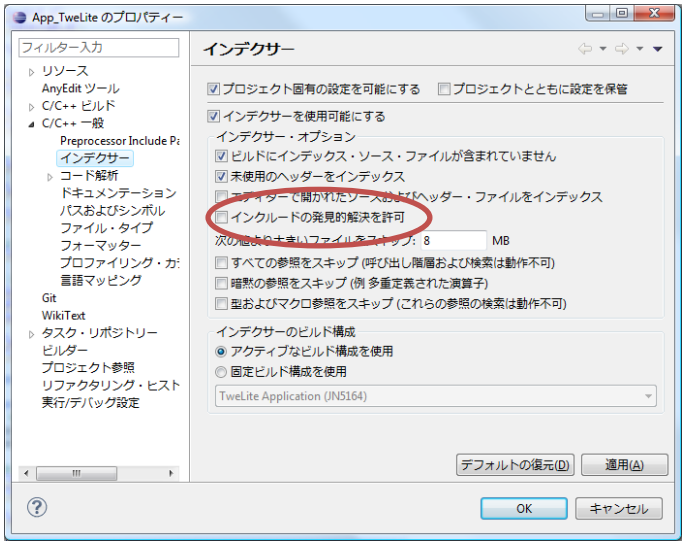


Eclipse のコード解釈 (2)

- コード解釈が上手くいかない場合
  - [プロジェクト＞プロパティ＞C/C++ インデクサー] で[プロジェクト固有の設定を可能にする]をチェックします。
  - [インクルードの発見的解決の許可] のチェックを外します。
  - プロジェクトのビルド＞クリーンを実行し、再ビルドします。
  - [インクルードの発見的解決の許可] をチェックします。

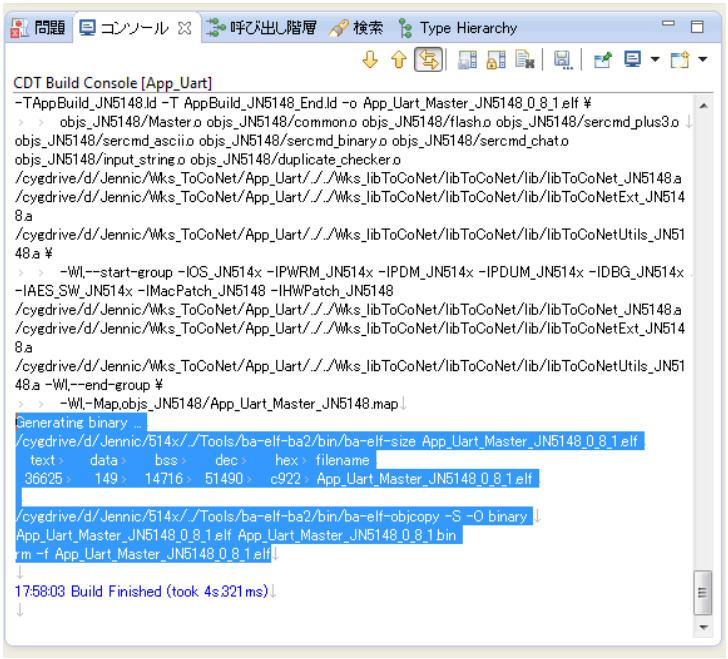
※ どうしても解釈が上手くいかない場合は、エラーを表示させないようにします。
[ウインドウ→設定→C/C++→コード解析] の Sytax and Semantic Errors のチェックを全部はずします。

※ Eclipse のコード解釈は複雑な振る舞いをします。完全に解釈できない場合もあります。



Eclipse 特有のエラーについて

- ビルドは成功していても出現するエラーが存在します。代表的なエラーを挙げます。
  - Program “ba-elf-gcc” not found in PATH
  - Program “gcc” not found in PATH
- これらのエラーは無視してください。
  - コンソールにてビルドログを確認して .elf ファイル .bin ファイルが生成されればビルドが完了しています。



MONO WIRELESS

Eclipse
SDK添付ワークスペース

Wks_ToCoNet (1)

- 本ワークスペースは ToCoNet を用いたアプリケーションやサンプルを格納しています。通常はこのワークスペースから始めてください。
  http://mono-wireless.com/jp/products/Software_download/index.html
  - App_Twelite (標準アプリケーション)
    - デジタル４入力4出力、アナログ４入力、PWM4出力、UART、I2C
  - App_Uart (UART通信アプリケーション)
    - UART 通信に特化、チャット・透過・書式、暗号化対応
  - App_IO (リモコン通信専用アプリ)
    - デジタル１２入力（子機）、アナログ１２出力（子機）、暗号化対応
  - App_Tag (無線タグアプリ)
    - ネットワークには参加せずブロードキャストでデータ送信のみを行う超省電力センサーノードを実現するためのサンプル。
    - マルチホップ中継可能です。

Wks_ToCoNet (2)

- Samp_PingPong (ごく単純な送受信サンプル)
  - 片側から Ping (ピン) を送信すると、受信した無線モジュールから Pong(ポン) を返す。UART で制御します。
  - 添付の readme_utf8.txt をご覧ください。
- Samp_PER (パケットエラー測定ツール)
  - UART制御、または評価開発キット上の液晶デバイスとボタンで実行するパケットエラー率の測定アプリケーション
  - 添付の readme_utf8.txt をご覧ください。
- Samp_Wayback (上り下り両対応子機のサンプル)
  - ネットワークに参加し、上り下り通信を行う子機を実現するためのサンプル。
  - 中継器を追加すると LayerTree 方式によりマルチホップ中継可能です。
- Samp_I2C (I2C デバイスの取り扱い)
  - I2C の各種デバイスのアクセス方法。
- App_Melody (標準アプリケーション改造例)
  - PWM出力によるメロディ演奏App_Audio (オーディオ伝送例、Push To Talk)
  - ADよりアナログ入力を得て、データ送信(ADPCM, 8bit 無圧縮, 10bit 無圧縮)、受信側では PWM として出力

Wks_libToCoNet

- 本ワークスペースは以下を格納します。Wks_ToCoNet のビルドには必須のディレクトリです。
  - ToCoNet のライブラリ・ヘッダファイル (include, lib)
  - ToCoNetUtils (ToCoNet ユーティリティ)
    - ToCoNet の各ユーティリティを格納しています。振る舞いの理解の目的、また動作の改造のために再コンパイルできるプロジェクト定義を用意しています。
      - ボタン入力判定
      - バイト列のFIFOキュー
      - シリアルポート
      - 乱数生成
      - シリアルポート, fprintf
      - 固定小数点演算 (fixmath)
      - 各種ハードAPIの手続き
      - など

Wks_empty

- 空のワークスペース。
  - 本SDKに関連する各設定が含まれます。
  - 既存の Wks_ToCoNet を複製して利用することを推奨します。

Wks_MAC

- 802.15.4 MAC層 API を用いたサンプル。
  過去のSDKに含まれていた一部のサンプルを格納する目的で格納しています。通常はこのワークスペースは使用しないでください。
  - SimpleTag_3
    - シンプルな構成の無線タグのデモ。
  - Signal_Transmit
    - アナログ信号伝送のデモ。
  - PingPong
    - シンプルな送受信を行うデモ。

MONO WIRELESS

Eclipse
GIT を用いたプロジェクト

Git のプロジェクト作成 (1)

※ Git を利用しなくても開発は可能です。

- Git はリーナス・トーバルズによって開発された、バージョン管理システムで、Eclipse とも統合されています。バージョン管理システムでの利点は、過去に編集したファイルとの差分や比較、またバージョンごとにファイル群を管理出来る事です。
- サーバを構築すれば複数開発者が利用する事も可能ですが、ここでは1人の開発者がPC上にデータベース（レポジトリ）を作成して版管理する例を紹介します。
- 先ほど追加した My_Proj1 プロジェクトを GIT の管理対象としてプロジェクトに追加する例を解説します。GIT のデータベースはプロジェクトと同じディレクトリに管理する方法（ローカルレポジトリ）で解説します。
- 予め、追加した My_Proj1 プロジェクトを Eclipse から削除してください。

- GIT をインストールし、実行可能なパスを設定しておきます。GITは以下からダウンロードします。(TWESDK/Tools/archive にも収録しています)
  http://msysgit.github.io/

Git のプロジェクト作成 (2)

- Git Bash を起動します。(Git インストール時に追加)
- My_Proj1 があるフォルダーに移動します

```
# cd /c/TWESDK/Wks_ToCoNet/My_Proj1
```

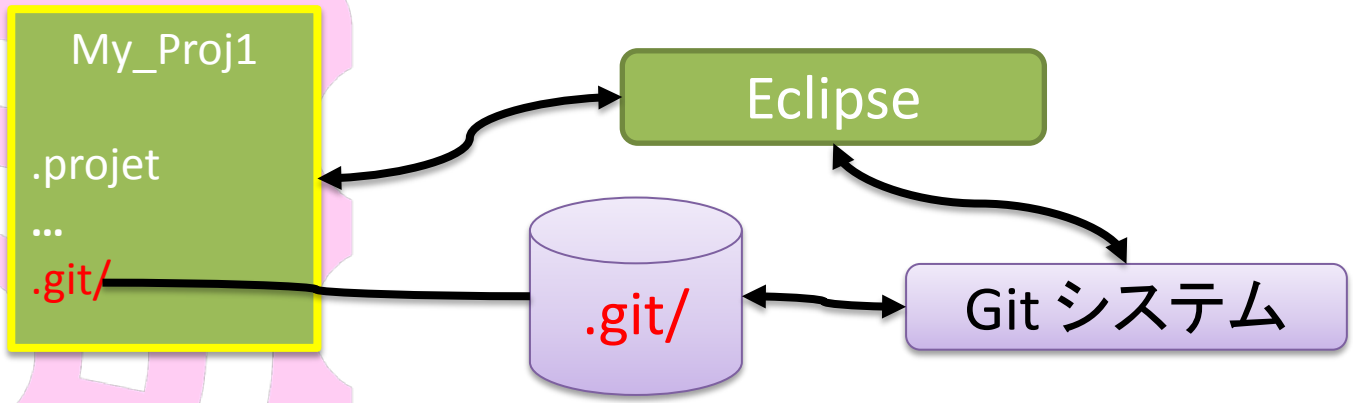
- 予めビルド時の中間ファイルなどを消去しておきます。
- レポジトリを作成します。プロジェクトフォルダに .git/ が作成されます。

```
# git init
Initialized empty Git repository in c:/TWESDK/Wks_ToCoNet/My_Proj1/.git/
```

- ファイルを追加します。Eclipse 上からの操作できますが、一旦全ファイルを GIT の管理対象として追加しておきます。

```
# git add -A
```

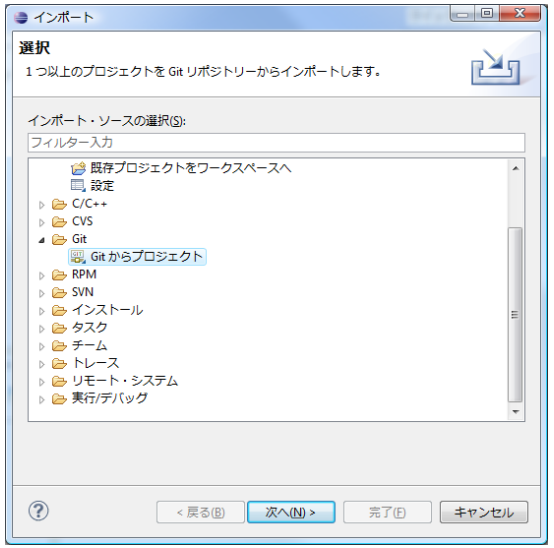
改行コードが CRLF から LF に変更される旨のワーニングなどが出ますが無視してもかまいません。

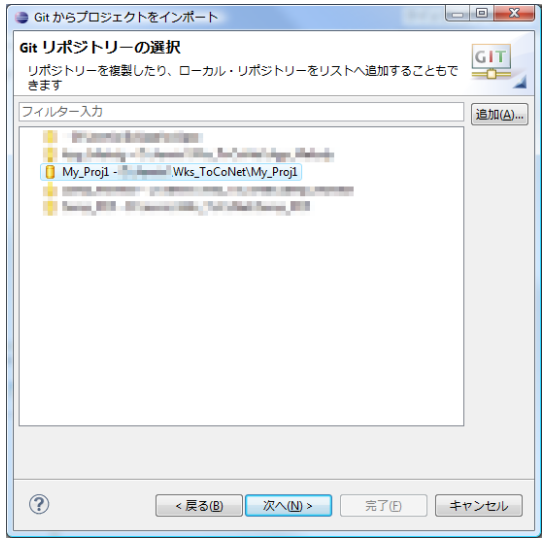


レポジトリはプロジェクトフォルダ/.gitに格納され、Eclipse は Git システム経由でバージョン管理を行います。
プロジェクトディレクトリ丸ごとをバックアップすれば、Git レポジトリも同時にバックアップされます。

Git のプロジェクト作成 (3)

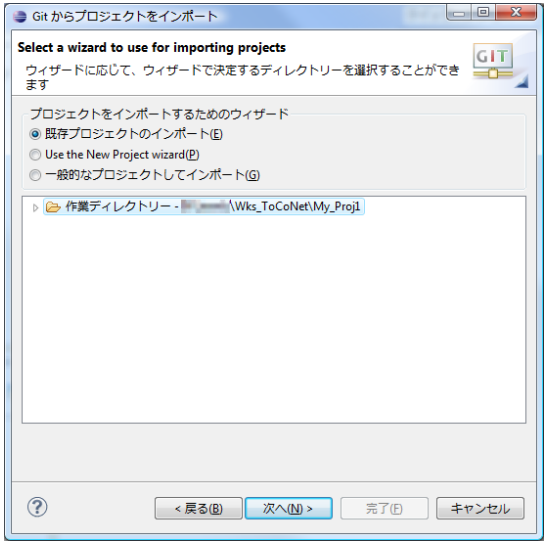
- Eclipse を起動し、対象のワークスペースを開きます。
- メニューより [ファイル]→[インポート] を選択します。
- [インポート・選択]ダイアログより [一般]→[Git からプロジェクト]を選択します。
- [Gitからプロジェクトをインポート・Select Repository Source] ダイアログよりローカルを選択し[次へ]ボタンを押す。
- [Gitからプロジェクトをインポート・GITリポジトリーの選択] ダイアログ上で [追加] ボタンを押す。
- 出現した[ローカル・ファイル・システム上の Git リポジトリの検索および選択] ダイアログ上で対象のプロジェクトのディレクトリを指定する。対象のプロジェクトがリストされるので、これをチェックした状態で [完了] ボタンを押す。
- 再び[Gitからプロジェクトをインポート・GITリポジトリーの選択]ダイアログ上で、対象のリポジトリを選択して [次へ] を押す。

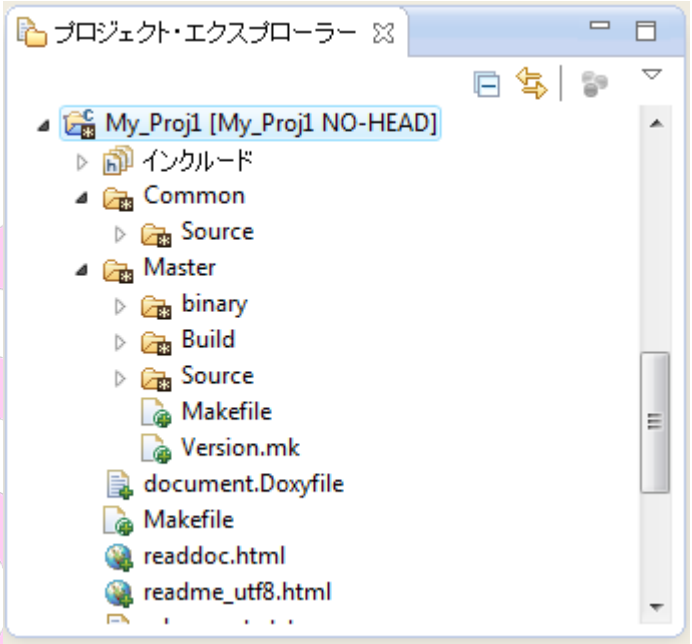


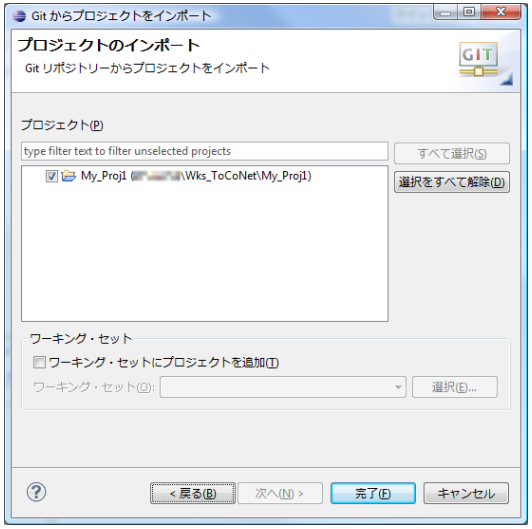


Git のプロジェクト作成 (4)

- [Gitからプロジェクトをインポート・Select a wizard to use for importing projects] ダイアログ上で、[既存プロジェクトをインポート] をチェックし [次へ] ボタンを押す。
- [Gitからプロジェクトをインポート・プロジェクトのインポート] ダイアログ上で、対象のプロジェクトがチェックされている事を確認して [完了] ボタンを押す。
- プロジェクトエクスプローラ上に Git のアイコン付きのプロジェクトが追加されます。

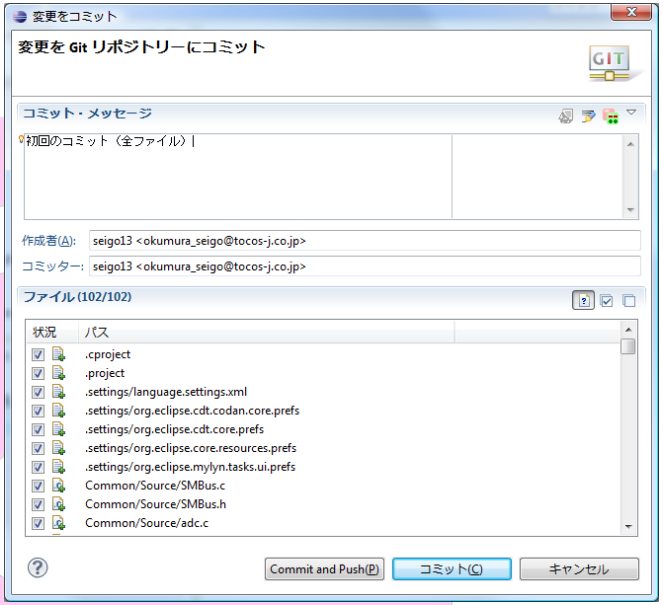


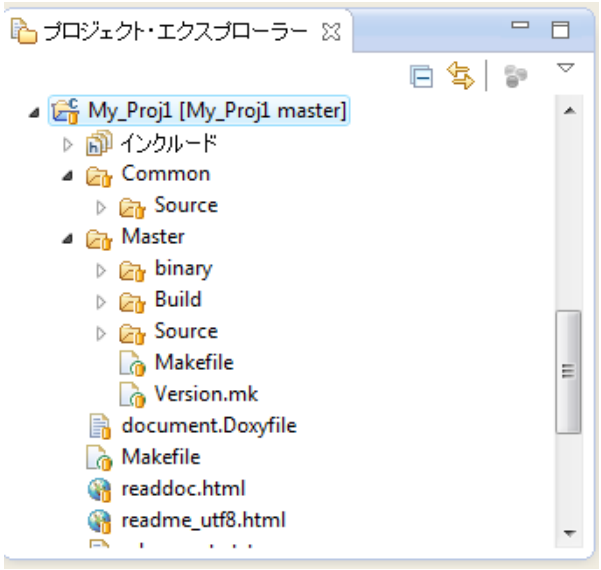




Git の操作 (コミット：登録)

- 先ほどまでの手順で Eclipse にインポートしたプロジェクトでは、コマンドラインから操作し Git に追加対象としたファイルが登録されています。
- Git のレポジトリに正式にファイルを保存するには以下の操作を行います。
  - My_Proj1 プロジェクトフォルダを右クリックし、出現したコンテキストメニューより [チーム]→[コミット] を選択します。
  - [変更をコミット] ダイアログ上で、コミット・メッセージに記載し [コミット] ボタンを押します。
  - プロジェクトエクスプローラ上でのアイコン表示がレポジトリ登録済みのアイコンに替わります。
- 変更の区切りがつくたびにコミットしていきます。変更履歴がコミットのたびに記録され、過去にコミットしたファイルと、現在編集中のファイルの比較など、開発の助けになる機能が提供されます。





Git の操作 (その他)

- 履歴状況をみたい
  → プロジェクトフォルダを右クリック、[チーム]→[ヒストリーに表示する]
- 直前のコミットとの差をみたい
  → 対象ファイルを右クリック、[比較]→[HEAD改訂]
- 新しく作成・追加したファイル(アイコンに？マークが付いている)を Git の管理対象にしたい
  → 対象ファイルを右クリック、[チーム]→[索引に追加]
- 版管理したい
  → プロジェクトフォルダを右クリック、[チーム]→[拡張]→[タグ...]
- 何故かコミット出来ないファイルが有る
  .gitignore ファイルを編集する、コマンドラインで強制コミットする。
- データベースのバックアップしたい
  プロジェクトフォルダを丸ごとバックアップします。

※ 上記では版管理のための最低限の機能のみを紹介しています。詳細は Git の解説を参照してください。

MONO WIRELESS

ファームウェアを書き込む

ツールの格納場所

- TWE-Progurammer (GUI プログラマ)
  - C:¥TWESDK¥Tools¥TWE-Programmer
- jenprog (CUI プログラマ)
  - C:¥TWESDK¥Tools¥jenprog
- TWE_PGM_MODE (CUI ToCoStick, TWE-Lite-R 用リセットツール)
  - C:¥TWESDK¥Tools¥TWE-RstPrg

シリアルポートの準備

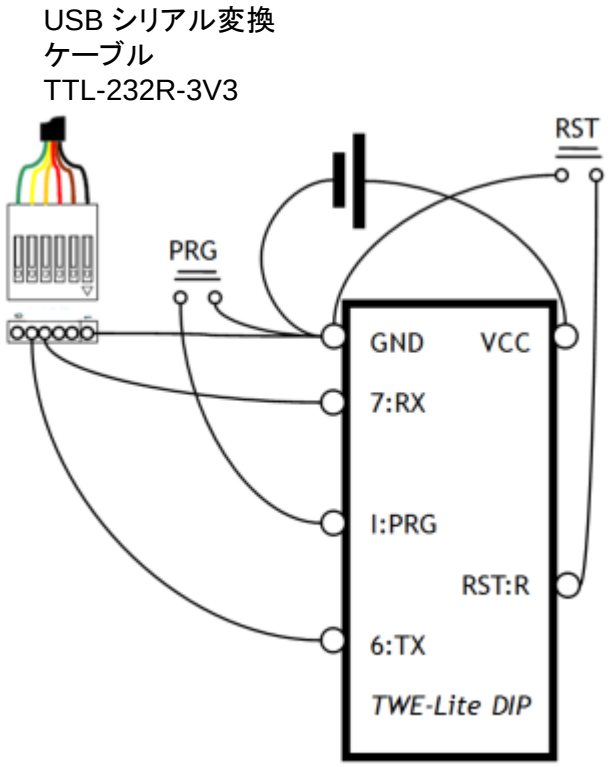
- ファームウェア書き込みには PC と TWE モジュールをシリアル接続する必要が有ります。
  - 評価開発キット (USBシリアルケーブル TTL-232R-3V3添付)
  - TWE-Lite-R
  - TTL-232R-3V3 などの USB シリアルケーブルを用意する
- TWE-Lite-R や TTL-232R-3V3は、PC上で動作する各種OS（Windows, Linux, Mac OS X）に対応したFTDI社 FT-232R が搭載されています。デバイスドライバは以下より入手してください。
  http://www.ftdichip.com/
  （Windows7 以降はOSに含まれていますので、原則ドライバのインストールは不要です。）
- ※ デバイスドライバを導入する際には TWE モジュールを接続しない状態で PC にケーブルを接続してください。

シリアルポートデバイスの確認

- TWE-Lite-RをUSBケーブルでUSBポートに接続します。この時点ではTWE-Lite DIPは接続しないでください。
  - Windowsでは、COMポートとして割り当てられます。
    デバイスマネージャを立ち上げて確認してください。
  - Linuxでは /dev/ttyUSB? に割り当てられます。
    ? は 0から順に割り当てられます。
  - Mac OS Xでは /dev/tty.usbserial* となります。
    * はデバイスのシリアル番号になります。
- 調べたシリアルポートの動作確認はターミナルなどのアプリケーションを使用し、ループバックテスト（折り返しテスト）で行う事ができます。TWE-Lite-Rのリセットボタン近くのスルーホールのTXとRXをクリップなどでショート接続した状態でターミナルに入力した文字がそのまま表示されれば正常です。
- パソコン上でターミナルソフトを立ち上げます。通信条件は 115200bps 8N1（8bit パリティ無し ストップビット1）です。
  ※ WindowsXP まで添付されていた HyperTerminal は推奨しません。ここでは TeraTerm を推奨します。(Linux, Mac OS X では screen, minicom が利用可能です)

配線

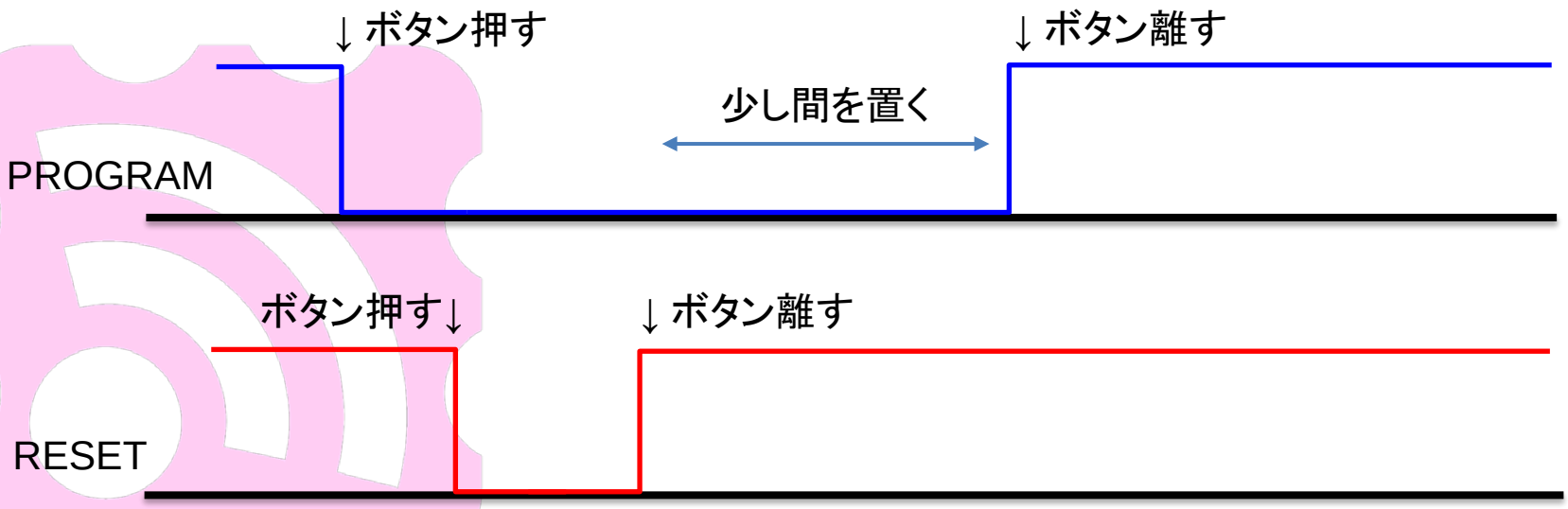
- TWEモジュールにファームウェアの書き込むには、以下の信号線を利用します。
  - SPIMISO (PROGRAM)
  - RESET
  - DIO6 (UART0 TXD)
  - DIO7 (UART0 RXD)
- 評価開発キットやTWE-Lite-Rでは、基板上に予め配線されています。開発用で新たに配線する時は右図を参考にしてください。
- 配線を全て確認したら、**TWEモジュールに電源供給を行わない状態**で USB ケーブルを接続します。またTWEモジュールの電源供給を断ってから USB ケーブルを取り外します。



TWE-Lite 配線例

プログラムモード

- TWEモジュールをプログラムモードに設定するには、
  1. PROGRAM(SPIMISO)=LO(GNDレベル) に設定して TWE モジュールを電源投入、またはリセット
  2. 動作が落ち着くまで SPIMISO を保持。
     メカのボタンで動作させるときは0.5秒以上、マイコンのIOから制御する時は 0.1秒以上を目安とします。
  3. SPIMISO を HI に戻す



書き込みユーティリティ

- TWE-Lite DIP にソフトウエアを書き込むためにTWEProgrammer (Windows, GUI版, TWE Lite 専用) またはjenprogユーティリティ(Windows,Linux,OS X, TWE全モデル対応)を使用します。
  - コマンドラインで実行するため、ユーティリティの実行形式のパスを設定する事を推奨します。
- ユーティリティの格納場所
  - TWE Programmer (Tools/tweprog/TWEProgrammer.exe)
    - .NET 4.0 の導入が必要です
      - (弊社確認の範囲では Windows7/8 では、そのまま動作します)
    - http://mono-wireless.com/jp/tech/misc/LiteProg/
  - jenprog ユーティリティ (Tools/jenprog)
    - http://mono-wireless.com/jp/tech/misc/jenprog/
    - Windows上ではそのまま動作します。以下の実行形式をコマンドラインから実行します。
      C:¥TWESDK¥Tools¥jenprog¥bin¥jenprog.exe
    - Linux, Mac OS Xを使用する場合はPython環境（Python2.7およびPyserial2.6）が必要です。Python環境は各OSにあらかじめ導入されている場合もありますが、必要に応じてセットアップしてください。

TWE Programmer (1)

- TWE Programmer は TWE-Lite 専用に開発された Windows 環境用のプログラマです。.bin 形式のTWE用アプリケーションソフトウェア(ファームウェア)を簡単に書き込めます。

1. TWE Programmer (C:¥TWESDK¥Tools¥tweprog¥TWE-Programmer.exe) を起動します。(起動前にシリアルポートを使用するターミナルソフトは終了してください。COMポートが占有されていると、COMポートが列挙されません)
2. COMポートを選択します。
   - 自動でTWE無線モジュールとの通信を確認し、成功すれば [TWE-Lite シリアル番号] を表示します。
   - 失敗したときは、TWEモジュールをプログラムモードに設定して [接続再チェック] ボタンを押してください。

TWE Programmer (2)

1. 前頁参照
2. 前頁参照
3. [ソフトウェアを選択して書き込む]ボタンを押して、書き込みたいソフトウェアファイル(.bin) を選択するか、.binファイルをドラッグ &ドロップします。
   - COMポートが既に選択された状態で、.binファイルを選択するとすぐにプログラムを開始します。
4. プログラムが終了すると、ウインドウの背景が水色（成功）またはピンク（失敗）になります。再度書き込みたいときは[(再)書き込み]ボタンを押してください。

※ TWE Programmer は、起動時コマンド引数にソフトウェアファイルを指定できます。Windows のアプリケーション関連付けで .bin を関連付けると .bin ファイルをダブルクリックするだけで TWE-Lite プログラマが起動し .bin ファイルが選択状態になります。

TWE Programmer (3)

※ TWE Programmer は、起動時コマンド引数にソフトウェアファイルを指定できます。Windows のアプリケーション関連付けで .bin を関連付けると .bin ファイルをダブルクリックするだけで TWE-Lite プログラマが起動し .bin ファイルが選択状態になります。

※ TWE Programmer 起動時に対応USBデバイス(TWE-Lite-R)が接続されている場合は自動認識して、該当のCOMポートの選択、TWEモジュールのリセット＆プログラムモードの設定を行います。

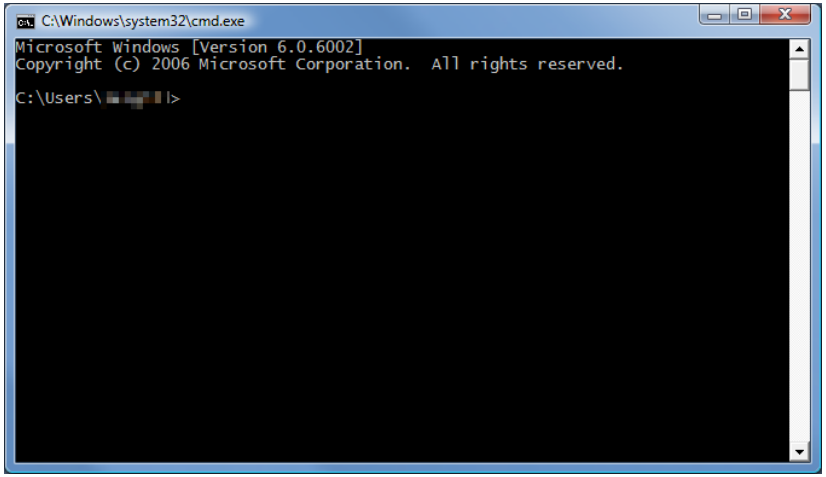
- COMポート選択時にもリセット＆プログラムモード遷移を行います。
- 複数接続されている場合は、いずれかの COM ポートが選択されます。

※ COMポートは書き込み時のみ占有します。

※ COMポート選択用のコンボボックスの [COMポートの再スキャン] を選択すると、新たに COM ポートを列挙します。

jenprog(1)

- ユーティリティはコマンドラインから使用します。
  - Windows 標準の cmd.exe
  - Cygwin bash など
- Windows cmd.exe の起動方法
  - スタートメニューより[プログラムとファイルの検索]に cmd と入力すると、cmd.exe が検索されます。(WindowsVista, 7)
  - cmd.exe をクリックします。



MONO WIRELESS

cmd.exe コマンドラインの操作法

```
C:¥Users¥MyName>
C:¥Users¥MyName> C:¥TW[TAB] → TWESDK に補完される
C:¥Users¥NyName> C:¥TWESDK¥T[TAB] → Tools に補完される
…
C:¥Users¥MyName> C:¥TWESDK¥Tools¥jenprog¥bin¥jenprog.exe[Enter]
*** jenprog ver 1.3.1 ***
Found ports:                                    ← jenprog 書き込みユーティリティをパラメータなしで起動
COM3                                              バージョンと利用可能なCOMポートが列挙される。

C:¥Users¥MyName> cd ¥TWESDK¥Tools[Enter]
C:¥TWESDK>                                      ← cd コマンドにより C:¥TWESDK に移動
C:¥TWESDK> cd Tools¥jenprog¥bin[Enter]
C:¥TWESDK¥Tools¥jenprog¥bin> jenprog[Enter] ← jenprog だけで起動できる。(.exe も省略可)
*** jenprog ver 1.3.1 ***
…

C:¥TWESDK> Tools¥jenprog¥bin¥jenprog -t COM3
    Wks_ToCoNet¥AppTwelite¥Master¥Build¥App_TweLite_Master_JN5148_1_3_6.bin[Enter]
                                                → 実際の書き込みコマンド例(2行に分かれているが実際は1行)
```

代表的な cmd.exe のコマンド

cd	現在のディレクトリを変更する
dir	ファイル一覧
copy	ファイルをコピーする
del	ファイルを消去する
{コマンド} /?	使用方法を表示する

jenprog(2) – 接続確認

- TWE モジュールが正しくプログラムモードで起動しているか確認してみます（通常の書き込みでは省略してもかまいません）。
- ユーティリティを引数なしで起動すると利用可能な COM ポートが列挙されます (Windows のみ)。

```
C:¥> C:¥TWESDK¥Tools¥jenprog¥bin¥jenprog
 *** jenprog ver 1.3 ***
   COM3,¥Device¥VCP0     ← Windows では利用可能な COM ポートが列挙される
```

- ユーティリティに –t {COMポート名} を指定すると、接続された TWE モジュールの各種情報が表示されます。

```
C:¥> C:¥TWESDK¥Tools¥jenprog¥bin¥jenprog -t COM3
*** jenprog ver 1.3 ***
   flash   : JN516x Internal Flash
   chip id : 0x10008686
   mac addr: 0x001bc50126300002
```

  - 注：ここでエラーが表示される場合は、プログラムモードの再設定、問題が有る場合はUSBケーブルを接続するところまで戻ってください。

jenprog(3) – 接続確認

- ユーティリティにシリアルポートのデバイス名とファイル名を与えると書き込み動作となります。
  - 書き込むファームウェアは、カレントフォルダ(C:¥)にあり、ファイル名は App_TweLite_Master_JN5164_X_Y_Z.bin とします。

```
C:¥> C:¥TWESDK¥Tools¥jenprog¥bin¥jenprog -t COM3 App_TweLite_Master_JN5164_X_Y_Z.bin
 *** jenprog ver 1.3 ***
   file info: 04 03 0008
   writing...
   0%..10%..20%..30%..40%..50%..60%..70%..80%..90%..done - 7.86 kb/s
   done
OK: firmware is successfully programmed.
```

  - 注：書き込み中は電気的ノイズなどに影響されやすいため、接続ケーブルや TWEモジュールが接続された回路に触れたり動かしたりしないようにしてください。万が一、書き込みエラーが起きた場合はもう一度書き込み手順をやり直してください。

jenprog(4) – 他のパラメータ

- -v　　ベリファイ
  - 書き込みコマンド実行時に –v を付記すると、書き込み後、TWEモジュール内のフラッシュに書き込まれた内容と比較を行う

  例: `jenprog –t COM3 –v foo.bin`
- -e　　消去
  - フラッシュの内容を消去します。

  例: `jenprog –t COM3 –e`
- -z　　比較
  - フラッシュの内容とファイルが同一であるか比較する。

  例: `jenprog –t COM3 –z foo.bin`
- -b 38400　　安全なボーレートで通信する
  - 全ての処理を 1Mbps の替わりに 38400bps で実行します。38400以外の指定はできません。

  例: `jenprog –t COM3 –b 38400 –v foo.bin`

TWE_PGM_MODE

- コマンドライン (CUI) から ToCoStick や TWE-Lite R のリセット、プログラムを制御するユーティリティです。jenprog と組み合わせて利用してください。

```
C:¥> TWE_PGM_MODE.exe 4 0            < リセットのみ
*** TWE Lite Reset/Programmer ***
*** Reset COM4 ***

C:¥> TWE_PGM_MODE.exe 4 1            < プログラム＆リセット
*** TWE Lite Reset/Programmer ***
*** ProgramMode COM4 ***
```

MONO WIRELESS

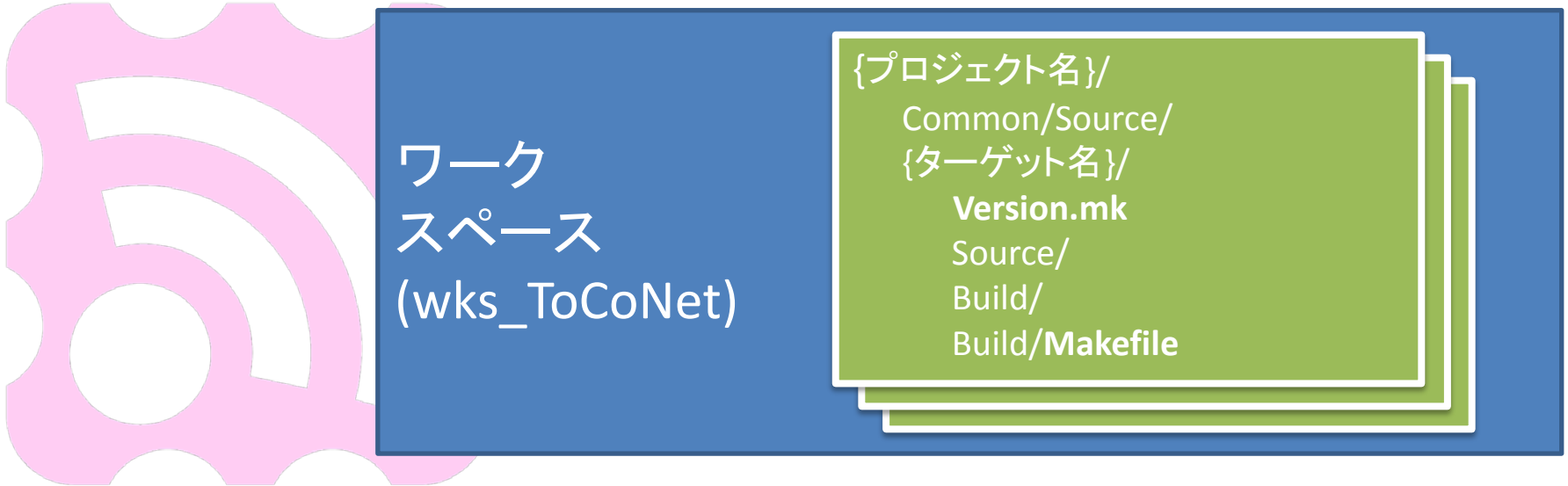
Makefile について

Makefile とは

- Makefile はプロジェクトのビルド方法を定義するファイルです。
- ビルド対象のファイルを追加したり、リンク時のライブラリを追加するような場合に編集します。
  ※ Eclipse のフォルダにファイルを追加しただけではビルド対象にはなりません。
- 通常は、既にビルド可能になっているプロジェクトから開発を進め、一から書き起こすことはありません。
- 本SDKではユーザによる Makefile の記述を最低限にするため、いくつかの事前設定されたファイルを用意しています。(TWESDK/MkFiles に格納)

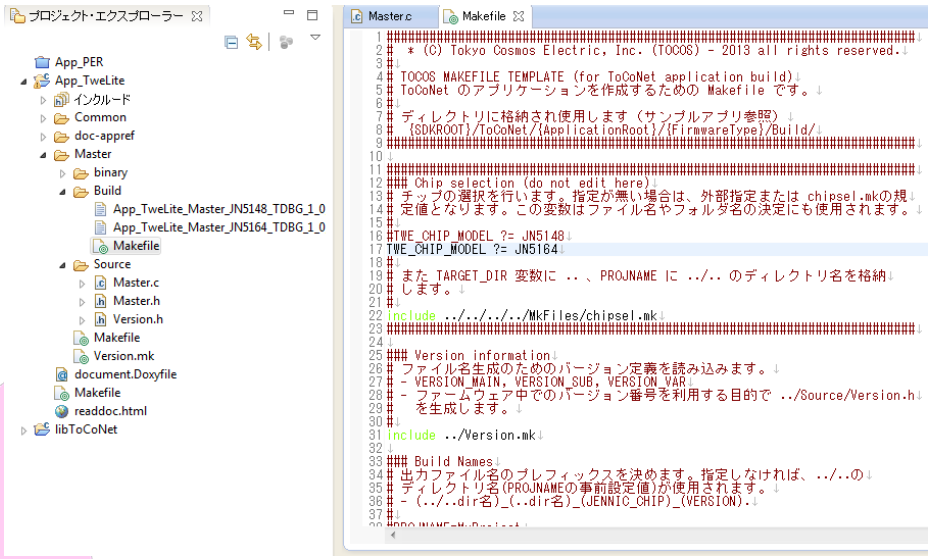
ディレクトリ構造

- Makefile ではいくつかのディレクトリ構造は固定的に運用されます。
  - Makefile は {プロジェクト名}/{ターゲット名}/Build に格納します。
    - これ以外のディレクトリではエラーになります。
    - Build ディレクトリには、生成バイナリや中間ファイルが格納されます。
  - ソースファイルは、以下に格納します。
    - {ターゲット名}/Source　　ターゲット特有のコードなど
    - Common/Source　　各ターゲット共通のコードなど
    - {ターゲット名}//Version.mkバージョンを定義ファイル



Makefile の例

- App_TweLite プロジェクトの Makefile を例にとって解説します。
- App_TweLite/Master/Build/Makefile を開いてください。



Makefile の編集

- プロジェクトにファイルを追加したい。
  ファイル名を myfile.c とすると、以下を追加します。
  - `APPSRC += myfile.c`

  ※ myfile.c は複数あるソースファイル格納ディレクトリのどこに格納してもかまいません。
- ソースファイル格納ディレクトリを増やしたい。
  Common/mysource/ に格納したい場合は、以下の行を追加します。
  - `APP_COMMON_SRC_DIR_ADD1 = ../../Common/mysource`

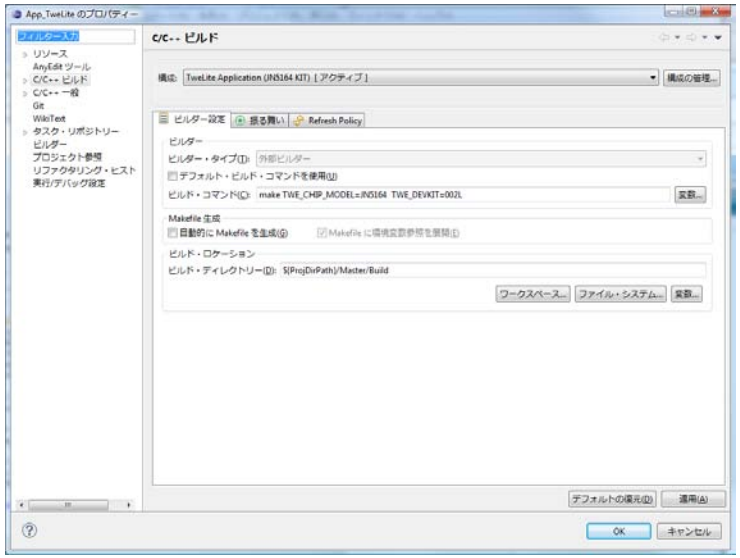
  ※ Makefile を基点に相対パスで指定します。プロジェクトの最上位ディレクトリは ../.. になります。
- C言語の定義を追加したい。
  例えば MYDEBUG をCプリプロセッサに定義したい場合は、以下の行を追加します。
  - `CFLAGS += -DMYDEBUG`
- make の引数を処理したい。
  例えば MY_OPT=ABC を make に渡された場合は、以下のように記述します。
  -
```
ifeq ($(MY_OPT),ABC)
  CFLAGS += -DMYDEBUG
endif
```
  ※ この記述は `make MY_OPT=ABC` と呼び出された場合に、Cプリプロセッサに対し `MYDEBUG` を定義します。

Eclipse から make の呼び出し

- Eclipse から make コマンドラインツールを Makefile を引数として呼び出しています。
  - 設定箇所は [プロジェクト]>[プロパティ]>[C/C++ビルド]の[ビルダー設定] です。
  - [ビルドコマンド] は make の呼び出しコマンドを指定します。

    以下の指定では TWE_CHIP_MODEL をTWE-Lite用として、また MY_OPT=ABC を指定しています。
    - `make TWE_CHIP_MODEL=JN5164 MY_OPT=ABC`
  - [ビルド・ディレクトリ] は make を実行するディレクトリを指定します。

    ${ProjDirPath} はプロジェクトの最上位ディレクトリです。以下の例では Master/Build にある Makefile を対象としてビルドすることを指定します。
    - ${ProjDirPath}/Master/Build



Makefile 解説 (1)

定義名	解説
TWE_CHIP_MODEL	TWE-Lite は JN5164 を指定します。
PROJNAME	生成ファイル名を決めます。指定が無い場合は ../.. ディレクトリ名を使用します。
APPSRC	コンパイル対象ファイルを指定します。APPSRC+=test.c のように += 演算子を使用します。
TARGET_TYPE	生成する対象を指定します。TARGET_TYPE = bin を指定した場合は実行形式、TARGET_TYPE = a を指定した場合はライブラリとなります。
TOCONET_DEBUG	1ならスタックのデバッグコードを含めます。出力ファイル名に _TDBG が付記されます。スタックデバッグコードについては、API解説を参照してください。0は含めず、バイナリサイズが若干小さくなります。
CFLAGS	コンパイル時の gcc オプションを追加します。CFLAGS += -DMY_DEBUG のように += 演算子を利用します。最適化オプションなどは事前定義されていますので指定しないようにしてください。
APP_COMMON_SRC_DIR_ADD1 .. 4	ソースおよびヘッダファイルの検索パスを指定します。APP_COMMON_SRC_DIR_ADD1 = ../mydir のようにMakefile のあるディレクトリからの相対パスを指定します。
INCFLAGS	インクルードディレクトリのみを追加指定したい場合は、INCLFAGS == -I../mydir のように相対パス指定します。

Makefile 解説 (2)

定義名	解説
ADDITIONAL_LIBS	追加でリンクしたいライブラリ (.a) を指定します。 ADDITIONAL_LIBS += ../mylib.a のように追加します。
ADDITIONAL_OBJS	追加でリンクしたいオブジェクトファイル (.o) を指定します。 ADDITIONAL_OBJS += ../myobj.o のように追加します。
LDLIBS	コンパイラ付属ライブラリ (math, sppなど) を指定します。LDLIBS += m のように追加します。この指定では-lmがリンカに渡されます。
LDFLAGS	リンカへの追加オプションです。LDFLAGS += -u dummy のように追加指定します。必要なオプションは事前に定義されています。
OBJDIR_SUFF	オブジェクト格納ディレクトリ名に設定した文字列を含める。
TARGET_SUFF	ターゲット(.bin)ファイル名に設定した文字列を含める。

Version.mk

- Version.mk ファイルは、各ターゲットの Source ディレクトリに格納し、Makefile から参照され、出力ファイル名のバージョンなどの決定に使用されます。

  以下のように指定します。

  - `VERSION_MAIN = 1` ← 主バージョン(0-255)
  - `VERSION_SUB = 3` ← 副バージョン(0-255)
  - `VERSION_VAR = 10` ← ビルドなどの派生(0-255)

- Makefile は Version.mk から以下のコンパイルオプションを追加します。

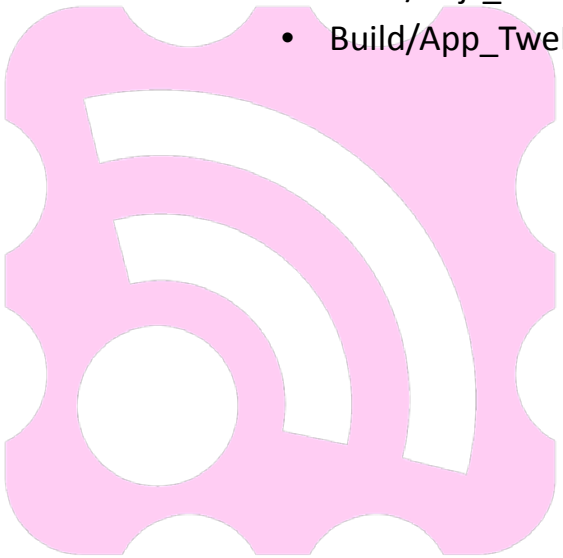
  上記の例では `-DVERSION_MAIN=1 -DVERSION_SUB=3 -DVERSION_VAR=10` を追加します。

  - ※ SDK 2013/11版までは `Version.h` が生成されましたが省略されます。ソース中で `Version.h` をインクルードしている場合は、`Version.h` を空ファイルにする、またはインクルードしないようにしてください。

ビルドディレクトリ

- ビルド生成物を格納するディレクトリは以下のように構成されます。ビルド定義ごとに、
  - 標準的な構成で TWE_CHIP_MODEL=JN5164 でビルド
    - Build/objs_JN5164/　　オブジェクトファイル
    - Build/App_TweLite_Master_JN5164_1_3_10.bin
      生成バイナリ(バージョン 1.3.10)
  - 以下を追加してビルド
    - OBJDIR_SUFF += _KIT002L
    - TARGET_SUFF += _KIT002L

    - Build/objs_JN5164_KIT002L/　　オブジェクトファイル
    - Build/App_TweLite_Master_JN5164_KIT002L_1_3_10.bin
      生成バイナリ



MONO WIRELESS

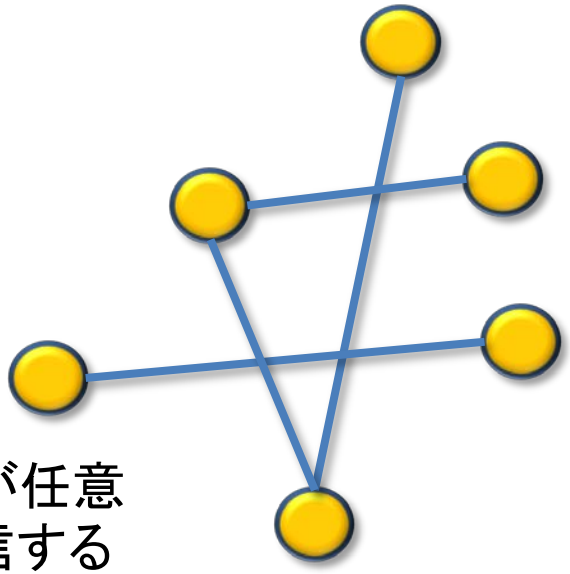
ネットワークの基本

TWE-NETの通信方式

- TWE-NET(ToCoNet)では、2種類の通信方式を記述できます。
  - ネットワーク無し
    - TWE-NETの中継ネットワークを使用せず、電波範囲内同士直接通信します。
    - ネットワークの構築や状態を意識する必要が無く、また親子関係などに縛られる事もありません。つまり電源を入れて直ぐに電波を送り直ぐに終了させると言った事が可能です。
    - 中継が必要な場合は、アプリケーションで全て同報通信で設計し、受信時にそのまま再送する簡易的な中継を実装します。
  - LayerTree ネットワーク
    - 親機と子機が有り、子機間で中継を行う中継ネットワークです。
    - LayerTree は、自身の中継深さ(親機からのホップ数)を指定する事でネットワークの構築を簡素化し、完全自動のネットワークに比べ素早いネットワーク構築が可能です。

ネットワーク無し

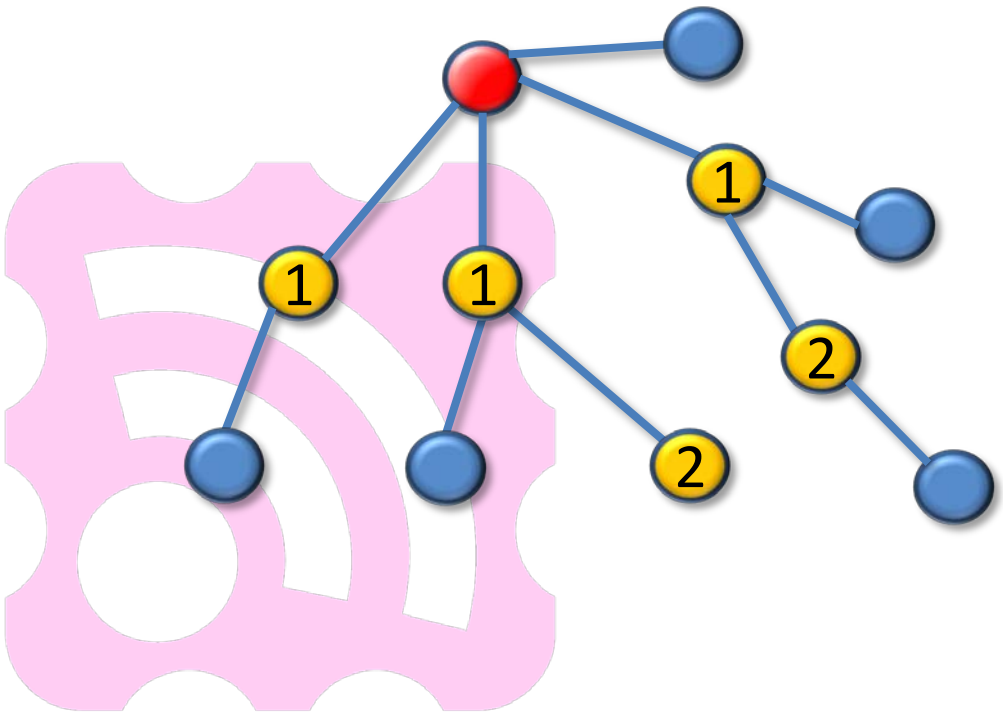
- ネットワークが存在しない場合、電波範囲内の通信は自由です。親機と子機という概念も存在しません。
- TWE-NETでは、ネットワーク無しの通信を支援するため以下の機能を有します。
  - 近隣探索：電波範囲内に存在するノードのアドレスや通信電波強度を列挙する
  - エナジースキャン：指定したチャネルのノイズレベルを返す
  - 同報通信
  - アドレスを指定したAck確認付きの通信
  - MAC層の再送に加えアプリケーションによる再送（回数・ランダム設定可能な遅延時間・再送間隔の指定可能）

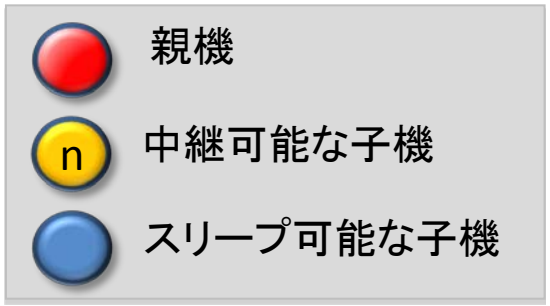


任意のノードが任意のノードに通信する

LayerTree ネットワーク機能

- 中継階層を**事前設定**する事で効率的に動作するネットワークです。
  - リニア型、ツリー型のトポロジーを組めます。
  - 例えば階層2、階層1と設定した場合、2＞1＞親機の順に配送されます。

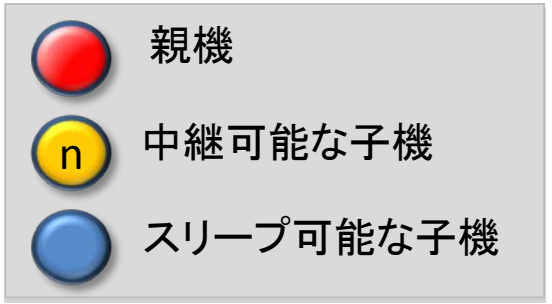




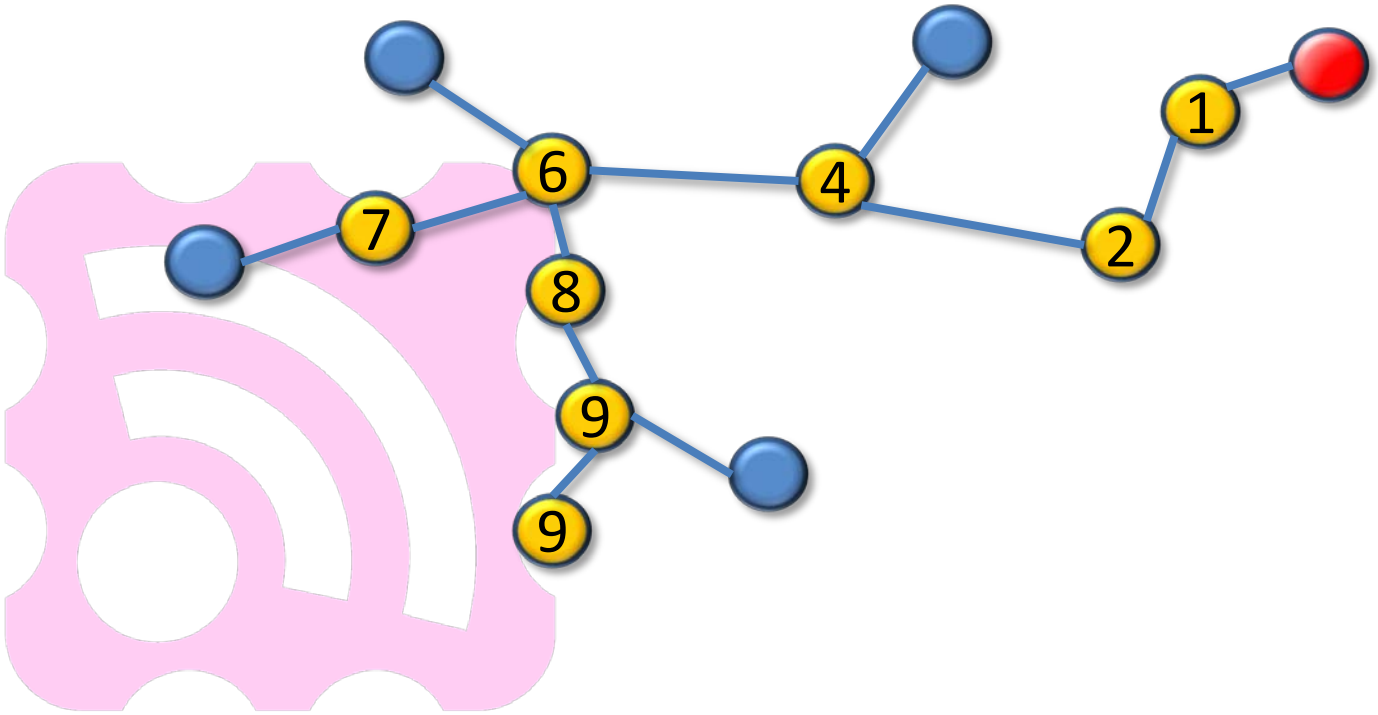
※ 中継子機の数字は、設定レイヤ数

ホップ数の多いネットワーク

- 最大63階層まで指定できます
- 昇順であれば、番号が飛んでも構いません

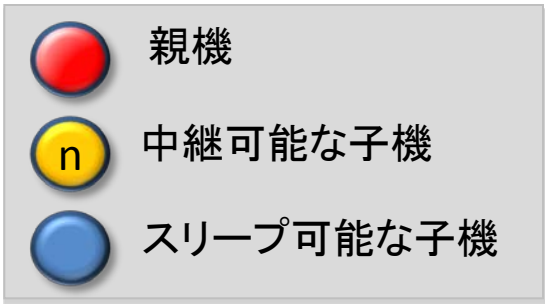


※ 中継子機の数字は、設定レイヤ数

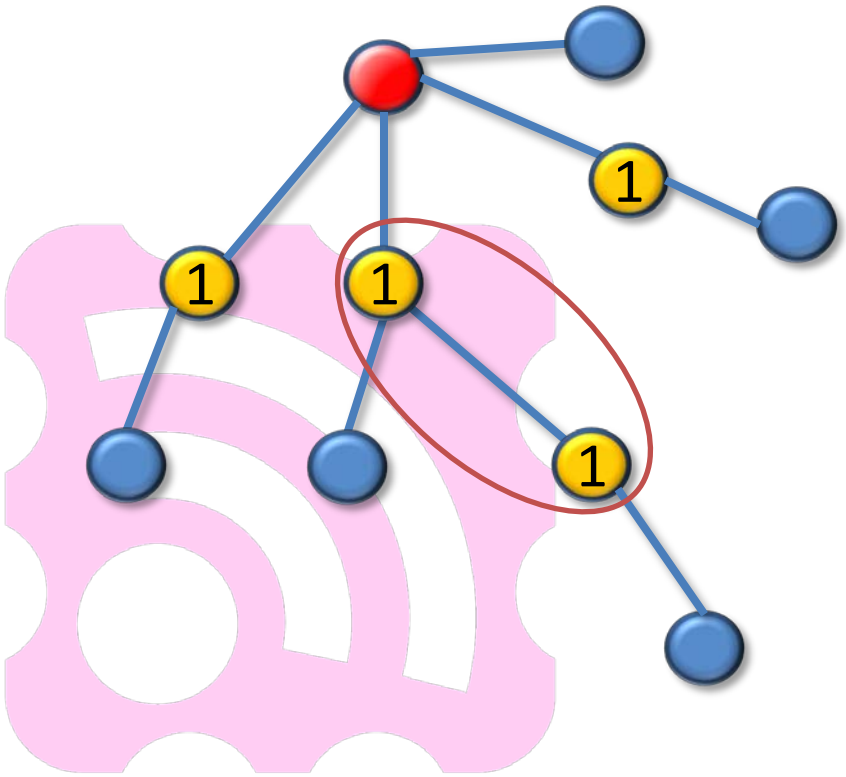


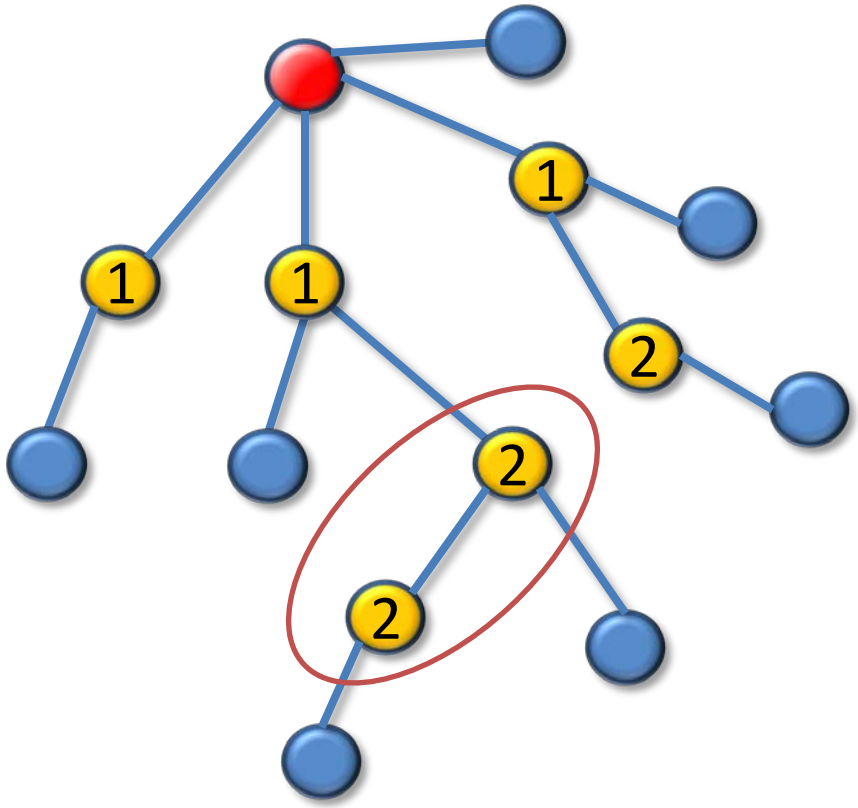
同一階層でのホップ

- 同一階層が上位階層を発見できなかった場合の救済として、既に上位階層との接続を完了したノードとの接続を行います。
- 但し、上位階層への接続を優先します。



※ 中継子機の数字は、設定レイヤ数





上位階層の決定アルゴリズム

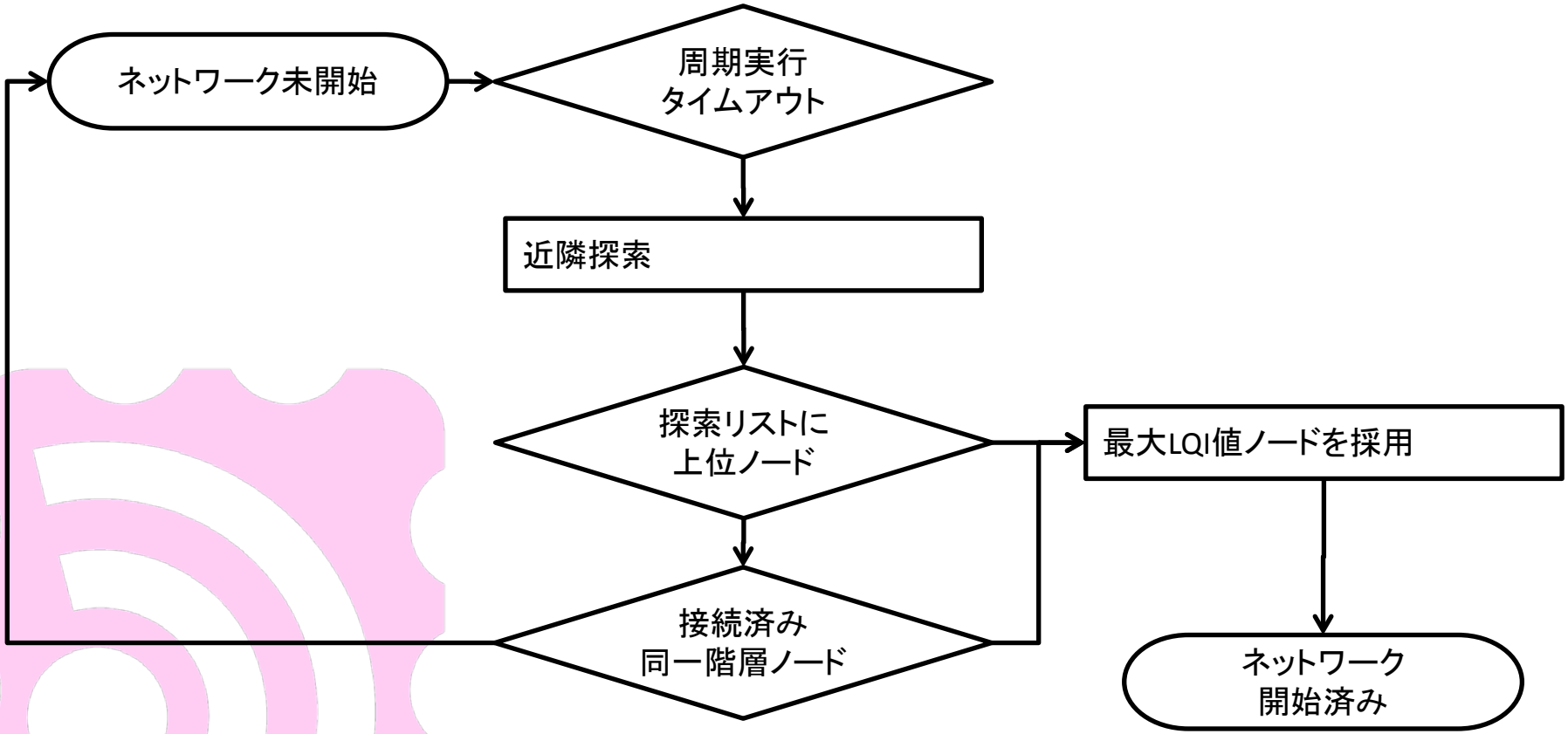
- 上位階層の探索は以下のように行われます。
  1. 近隣探索を行う。
  2. 自身より上位階層のノード中で最もLQI(電波強度)の高いノードを上位ノードとする。

- 上記で近隣ノードが発見できなかったときは、
  1. 自身と同一階層で、かつ上位階層との接続済みのノードの中で、最も LQI (電波強度) の高いノードを上位ノードとする。

- 上位ノード発見後はネットワーク開始状態となりますが、上位ノードとの通信を定期的に確認しています。
  1. 上位ノードにダミーパケットを送信する。
  2. 一定回数失敗すると、未接続状態とし新たに上位ノードを探索し直す。

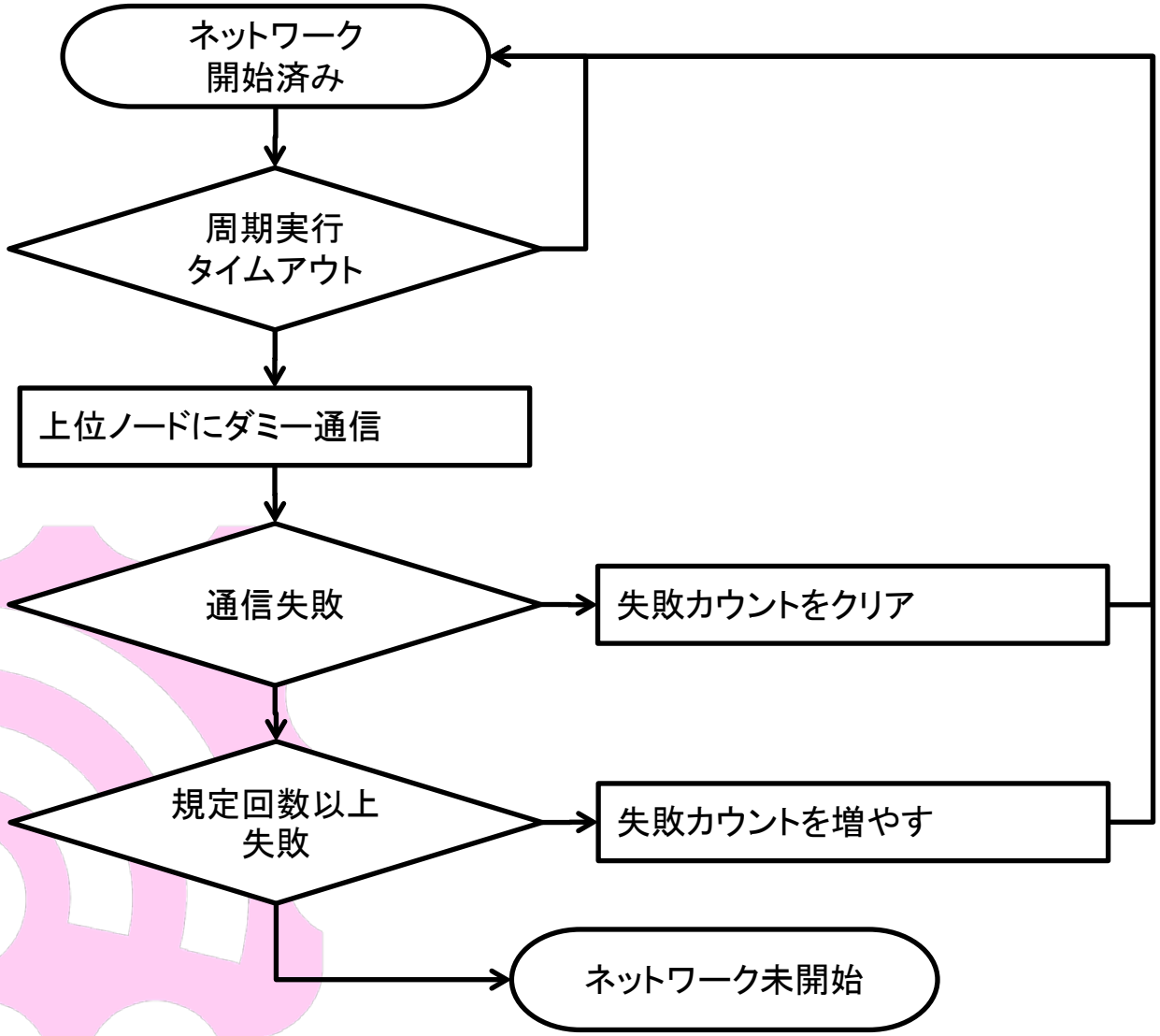
上位ノードの探索アルゴリズム(1)

- 上位ノードが未発見の状態から発見状態まで。



上位ノードの探索アルゴリズム(2)

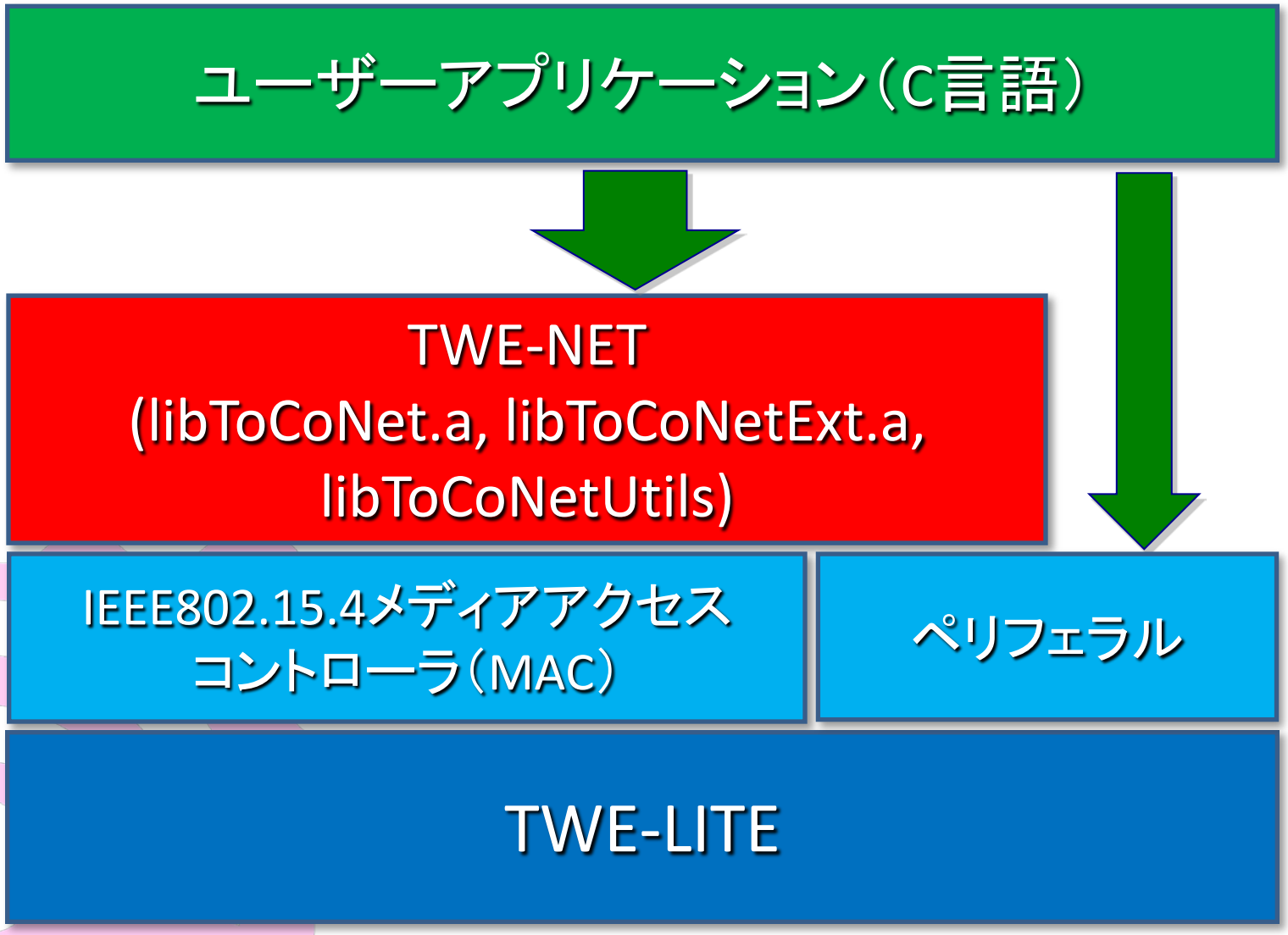
- 上位ノード発見状態で周期的に上位ノードの存在を確認する。



MONO WIRELESS

ライブラリの基本

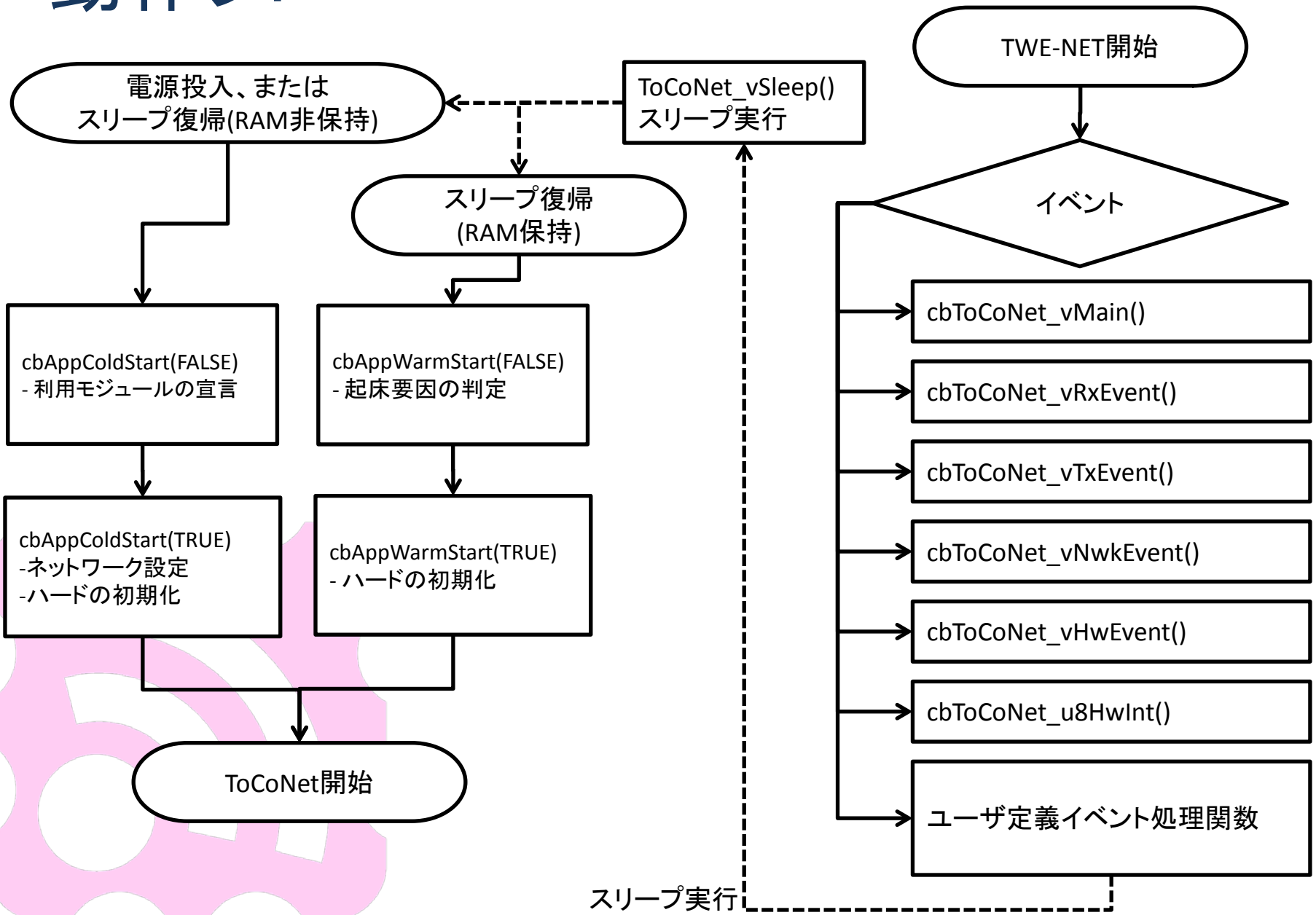
ライブラリ構造


ユーザーアプリケーション（C言語）
TWE-NET
(libToCoNet.a, libToCoNetExt.a,
libToCoNetUtils)
IEEE802.15.4メディアアクセス
コントローラ（MAC）
ペリフェラル
TWE-LITE

ライブラリ構造

- ユーザアプリケーションは TWE-NETのライブラリおよびペリフェラルAPIを介して無線モジュールを制御します。TWE-NETのライブラリは3つに分かれています。
  - libToCoNet.a : アプリケーションループの定義、イベント処理、無線ネットワーク処理など
  - libToCoNetEx.a : 始動処理・MAC層へのインタフェースなど
  - litToCoNetUtils.a : 一般的なアルゴリズムやペリフェラルの手続きなど（ソース添付）

動作フロー


電源投入、または
スリープ復帰(RAM非保持)
ToCoNet_vSleep()
スリープ実行
TWE-NET開始
スリープ復帰
(RAM保持)
イベント
cbAppColdStart(FALSE)
- 利用モジュールの宣言
cbAppWarmStart(FALSE)
- 起床要因の判定
cbToCoNet_vMain()
cbToCoNet_vRxEvent()
cbToCoNet_vTxEvent()
cbToCoNet_vNwkEvent()
cbToCoNet_vHwEvent()
cbToCoNet_u8HwInt()
cbAppColdStart(TRUE)
-ネットワーク設定
-ハードの初期化
cbAppWarmStart(TRUE)
- ハードの初期化
ユーザ定義イベント処理関数
ToCoNet開始
スリープ実行

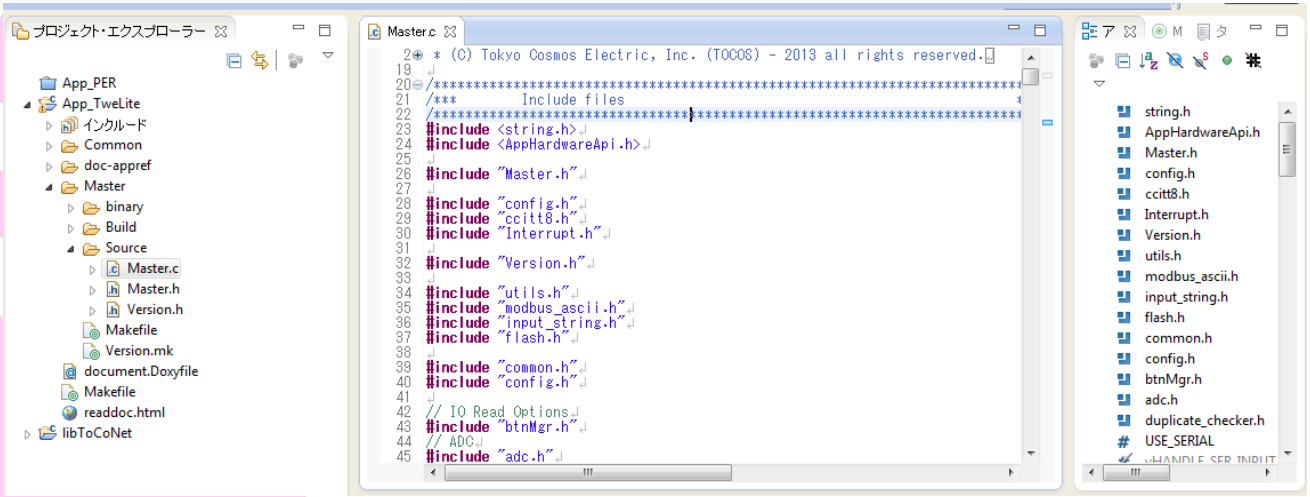
動作フロー

- TWE-NETでは、いくつかのコールバック関数に処理を記述します。
  - cbAppColdStart() – 電源投入時に呼び出される関数
  - cbAppWarmStart() – スリープ復帰時に呼び出される関数
  - cbToCoNet_vMain() – メインループ
    無限ループではなく割り込みなどの発生を起点として呼び出されます
  - cbToCoNet_vRxEvent() – 無線パケット受信時に呼び出されます
  - cbToCoNet_vTxEvent() – 無線パケット送信完了時に呼び出されます
  - cbToCoNet_vNwkEvent() – MAC層やネットワーク層の各種イベント通知
  - cbToCoNet_vHwEvent () – ペリフェラルの割り込み処理の遅延実行部。割り込み処理後に呼び出されます
  - cbToCoNet_u8HwInt() – ペリフェラルの割り込みハンドラ
  - ユーザ定義イベント処理関数 – 状態遷移によるタスク処理を記述します

- いずれのコールバック関数も、**呼び出された後に制御を返す**必要があり、非常に長い処理を記述すると全体の振る舞いに影響します。特に割り込みハンドラの記述には気を配ります。

App_TweLite

- 以下では App_TweLite のプロジェクトを参考に API の解説を行います。
- App_TweLite のディレクトリ構造は以下のようになります。
  - Master　　：アプリケーションの主要処理
  - Common　　：その他処理コード
- アプリケーションの主要処理は Master/Source/Master.c です。このファイルを開いてください。



利用するモジュールの宣言

- ToCoNet では、生成するバイナリ・メモリーサイズを適正化するため、コンパイル時に利用する機能を静的に決定しています。このため、マクロ定義および cbAppColdStart() での初期化関数呼び出しが必要です。
  - 利用する機能を #define してから、”ToCoNet.h”, “ToCoNet_mod_porototype.h” を #include します。
  - cbAppColdStart() の引数 bStart が FALSE の節で ToCoNet_REG_MOD_ALL() を呼び出します。

Master.c より

```
// Select Modules (define befor include "ToCoNet.h")
#define ToCoNet_USE_MOD_RXQUEUE_BIG
#define ToCoNet_USE_MOD_CHANNEL_MGR

// includes
#include "ToCoNet.h"
#include "ToCoNet_mod_prototype.h"

#include "app_event.h“
…
void cbAppColdStart(bool_t bStart) {
  if (!bStart) {
    // Module Registration
    ToCoNet_REG_MOD_ALL();
  } else {
…
```

リファレンス - モジュール一覧

モジュール定義	解説
ToCoNet_REG_MOD_ENERGYSCAN	チャネルの入力レベルを計測します。入力時には送受信が不可能になります。 関数: ToCoNet_EnergyScan_bStart() イベント: *E_EVENT_TOCONET_ENERGY_SCAN_COMPLETE*
ToCoNet_USE_MOD_NBSCAN ToCoNet_USE_MOD_NBSCAN_SLAVE	近隣のモジュールを探索します。 関数: ToCoNet_NbScan_bStart(), ToCoNet_NbScan_bStartToFindRole(), ToCoNet_NbScan_bStartToFindAddr() イベント: *E_EVENT_TOCONET_NWK_SCAN_COMPLETE*
ToCoNet_USE_MOD_RAND_MT ToCoNet_USE_MOD_RAND_XOR_SHIFT	乱数生成アルゴリズムを登録します。登録しない場合は、内蔵のハードウェア乱数が使用されます。ただし、連続的に 32bit の乱数系列が必要な場合はソフトウェア乱数を使用します。(MT 法を利用する場合は、ライセンス表記が必要です)
ToCoNet_USE_MOD_NWK_LAYERTREE	レイヤーツリー型ネットワーク層を利用します。 依存モジュール: ToCoNet_USE_MOD_NBSCAN, ToCoNet_USE_MOD_NBSCAN_SLAVE
ToCoNet_USE_MOD_NWK_LAYERTREE_MININODES	同報送信専用のレイヤーツリー型ネットワークにデータ送信可能な省電力「ミニノード」。 依存モジュール: ToCoNet_USE_MOD_DUCHK
ToCoNet_USE_MOD_DUPCHK	パケットの重複チェッカ。最大40ノードまでの重複パケットを管理する。レイヤーツリー型「ミニノード」を使用する時には必須定義モジュール。
ToCoNet_REG_MOD_NWK_MESSAGE_POOL	メッセージプール機能を利用します。 依存モジュール: ToCoNet_USE_MOD_NWK_LAYERTREE
ToCoNet_USE_MOD_CHANNEL_MGR	チャネルアジリティを利用します。u32ChMask に指定したチャネルを利用して、複数チャネル駆動の通信を行います。
ToCoNet_USE_MOD_TXRXQUEUE_SMALL ToCoNet_USE_MOD_TXRXQUEUE_MID ToCoNet_USE_MOD_TXRXQUEUE_BIG	送信キューのサイズを決定します。SMALLは送信用で 3ヶ、MIDは6ヶ、BIGは20ヶのキューを確保します。パケット分割を行うような一度に多くのパケットを連続的に送信する場合はBIGを指定します。1ヶあたり約128バイトのメモリを消費し、未定義時は MID となります。

cbAppColdStart の処理

- cbAppColdStart(bool_t bStart) は、bStart == FALSE で最初に呼び出され、その後 bStart == TRUE で呼び出されます。bStart == TRUE の節でアプリケーションの初期化処理を行います。
- ここで TWE-NETの重要な初期化 (sToCoNet_AppContext) を行う必要が有ります。一部の機能はここでしか設定できません。
- ToCoNet_Event_Register_State_Machine() 関数にて、ユーザ定義イベント処理関数を登録しています。
- この処理の最後で ToCoNet_vMacStart() 関数を呼び出します。
  - API の手続きとして MAC 層の開始を明示しています。

cbAppWarmStartの処理

- cbAppWarmStart(bool_t bStart) は、スリープ復帰後に呼び出されます。
  - １度目は引数が FALSE で呼び出されます。
    - u32AHI_Init() 前の処理を記述します（スリープ復帰要因など）。
  - ２度目は引数が TRUE で呼び出されます。
    - 各種初期化（ハードウェアなど）
    - sToCoNet_AppContext やユーザ定義イベント関数の登録は不要です。

```
// 割り込み要因の判定例（スリープタイマー割り込みとDIO割り込みを有効にした場合）

if(!bStart) {
  sAppData.bWakeupByButton = FALSE;
  If(u8AHI_WakeTimerFiredStatus()) {
    ; // スリープタイマーによる起床
  } else
  if(u32AHI_DioWakeStatus() & ((1UL << PORT_INPUT1) | (1UL << PORT_INPUT2)
    | (1UL << PORT_INPUT3) | (1UL << PORT_INPUT4)) ) {
    // ボタンによる割り込み起床
    sAppData.bWakeupByButton = TRUE;
  }
}
```

リファレンス sToCoNet_AppContext (1)

定義名	解説
uint32 u32AppId (必須、始動後変更不可)	32bitのアプリケーションID。本IDでTWE-NET同士の識別を行います。 値域: ????0000, ????FFFF, 0000????, FFFF???? は設定できません。 規定値: 0xFFFFFFFF。必ずアプリケーションから設定します。
uint32 u32ChMask (モジュールによって必須)	利用するチャネル群。ch13 をマスクに加える場合は 1UL << 13 のビットを1にします。 規定値: 0x07fff800UL (ch11～26: TWE-Regular, Strong では ch26 は使用不可) 必須モジュール: CHANNEL_MGR, NBSCAN, NBSCAN_SLAVE, LAYERTREE
uint16 u16ShortAddress	モジュールのショートアドレス。指定しなければモジュールのシリアル番号から自動生成されます。0xFFFFは指定出来ません。ネットワーク層利用時は指定できません。 規定値: モジュールシリアル番号から自動設定
uint8 u8Channel (必須)	モジュールの利用チャネル。上記 u32ChMask に含まれるチャネルを指定します。 ※ モジュールToCoNet_USE_MOD_CHANNEL_MGR 利用時は設定不要ですが、アプリケーションからのチャネル変更は出来ません。 値域: 11～26 規定値: 18
uint8 u8CPUClk (変更推奨せず)	通常稼働時のCPUクロック。3:32MHz, 2:16Mhz, 1:8Mhz, 0:4Mhz を指定します。 規定値: 2:16MHz
unit8 u8TxPower	モジュールの出力 3:最大 2: -11.5db 2: -23db 0:-34.5db となります。(規定値: 3)
uint8 u8TxMacRetry	MAC層の再送回数 0..7 を指定します。(規定値: 3)
bool_t bRxOnIdle	TRUE:無線回路アイドル時も受信回路を動作させます。受信が必要な場合は必ずTRUEに設定にしますが、受信時は常に受信電流を消費します。 規定値: FALSE (ネットワーク層では TRUE 必須)

※ 国によって使用可能なチャネルが違います。(例: 日本では 11-26 対応、米国では 26 不可) 詳細はお問い合わせください。

リファレンス sToCoNet_AppContext (2)

定義名	解説
uint8 u8CCA_Retry	CCA のリトライ回数 (通常は変更しません)
uint8 u8CCA_Level	CCA アルゴリズムの開始レベル (通常は変更しません)
uint8 u8RandMode	乱数生成方法の指定。0:ハード 1:システム経過時間を元に生成 2:MT法 (規定値は 0) 3:XorShift法 規定値: 0 ※ MT法, XorShift法の利用には対応モジュール定義が必要です。
uint16 u16TickHz (変更推奨せず)	システムの Tick カウントの周期(1000で割り切れる値にすること。1000, 500, 250, 200, 100) 規定値: 250 (4ms)
bool_t bSkipBootCalib	起床時の RC タイマーのキャリブレーション値の計算を省略する。明示的にキャリブレーションを設定するには ToCoNet_u16RcCalib() API を利用します。

※ (変更不可) は、cbAppColdStart() 後に変更してはいけません。
※ 構造体の値を変更してから ToCoNet_vRfConfig() を呼び出します。

リファレンス sToCoNet_AppContext (2)
～アプリケーションIDについて～

- 指定 32bit (0xHHHHLLLL)
  - HHHH → 0x0001～0x7FFF
  - LLLL → 0x0001 ～ 0x7FFF
  - 同一のアプリケーションIDを有するデバイスがネットワークに参加できます。
- 保有モジュールに割り当てられたシリアル番号
  - 0x8XXX:YYYY
  - モジュールに割り当てられたシリアル番号(缶に記載のアドレス + 0x8000:0000) は重複が無いことが保証されます。
  - このアドレスをアプリケーションIDに利用すれば、重複が起きません。
  - YYYY が 0000 または 0xFFFF の場合は利用できません。

ユーザ定義イベント処理関数 (1)

- ユーザ定義イベント処理関数は、TWE-NETの一部として動作します。
  - 引数は tsEvent *pEv, teEvent eEvent, uint32 u32evarg
    - pEv: イベント管理構造体, eEvent: イベント種別, u32evarg:イベント引数
  - システムからは以下のイベントが伝達されます。
    - E_EVENT_START_UP : 始動時
    - E_EVENT_TICK_TIMER : 4ms 周期
    - E_EVENT_TICK_SECOND : 1秒周期
  - ToCoNet_Event_SetState() により状態遷移を行います。状態遷移後 E_EVENT_NEW_STATE イベントが発生する。このため、連続して状態遷移出来ます。
  - その他のイベント通知が必要な場合は、ToCoNet_Event_Process() 関数を用います。

    例 : ToCoNet_Event_Process(*E_EVENT_APP_TX_COMPLETE, u8CbId, sAppData.prPrsEv);*
  - ToCoNet_Event_vKeepStateOnRamHoldSleep(pEv) を呼び出しておくことで、スリープ前の状態を保持する事が出来ます。
    - 復帰後は E_EVENT_START_UP イベントが発生します。
  - イベント処理関数は必ず E_STATE_IDLE = 0 から開始します。システム組み込み状態 (teState) 内の E_STATE_RUNNING, E_STATE_FINISHED は利用できます。それ以外はユーザ定義状態として定義します。

ユーザ定義イベント処理関数 (2)

- App_TweLite では合計3つのイベント処理関数を定義しています。
  - vProcessEvCore()

    アプリケーション全体の制御。ネットワーク層の利用時を想定していますが、ネットワーク層処理が無い場合には特別な処理はありません。
  - vProcessEvCorePwr()

    常時通電モードの制御。秒64回のタイマーイベントを基点に、IO状態の変化を検知し、データの送信を行います。
  - vProcessEvCoreSlp()

    スリープ稼動モードの制御。IO状態の確定を待ってから、データ送信を行い、データ送信完了を待ってスリープします。

リファレンス
ユーザ定義イベント処理関数

定義名	解説
ToCoNet_Event_Register_State_Machine (void *pvFunc)	ユーザ定義イベント処理関数を登録します。pvFunc にはイベント処理関数の関数ポインタを指定します。戻り値は uint8 でイベント処理関数のハンドル。
ToCoNet_Event_Process (uint32 eEvent, uint32 u32evarg, void* pvFunc) ToCoNet_Event_ProcessH (uint32 eEvent, uint32 u32evarg, uint8 u8Hnd)	イベント処理関数外から指定するイベント処理関数(pvFunc, u8Hnd)でイベント(eEvent)をイベント引数(u32evarg)にて発生させます。実体は関数呼び出しで、システムに制御を戻したりせず、即座に呼び出されます。 ※ 同一イベント処理関数内からは呼び出してはいけません。
ToCoNet_Event_u32TickFrNewState (tsEvent *pEv)	現在の状態に状態遷移してからの経過時間[ms]を戻します。イベントのタイムアウトに利用します。 戻り値は uint32 で経過時間[ms]。
ToCoNet_Event_eGetState(void* pvFunc) ToCoNet_Event_eGetStateH(uint8 u8Hnd)	指定するイベント処理関数(pvFunc, u8Hnd)の状態を返します。 戻り値は teState でイベント処理関数の状態。
ToCoNet_Event_vKeepStateOnRamHoldSleep (tsEvent *pEv)	スリープ復帰後もスリープ直前の状態に復帰します。スリープ復帰後は E_EVENT_START_UP イベントが発生します。

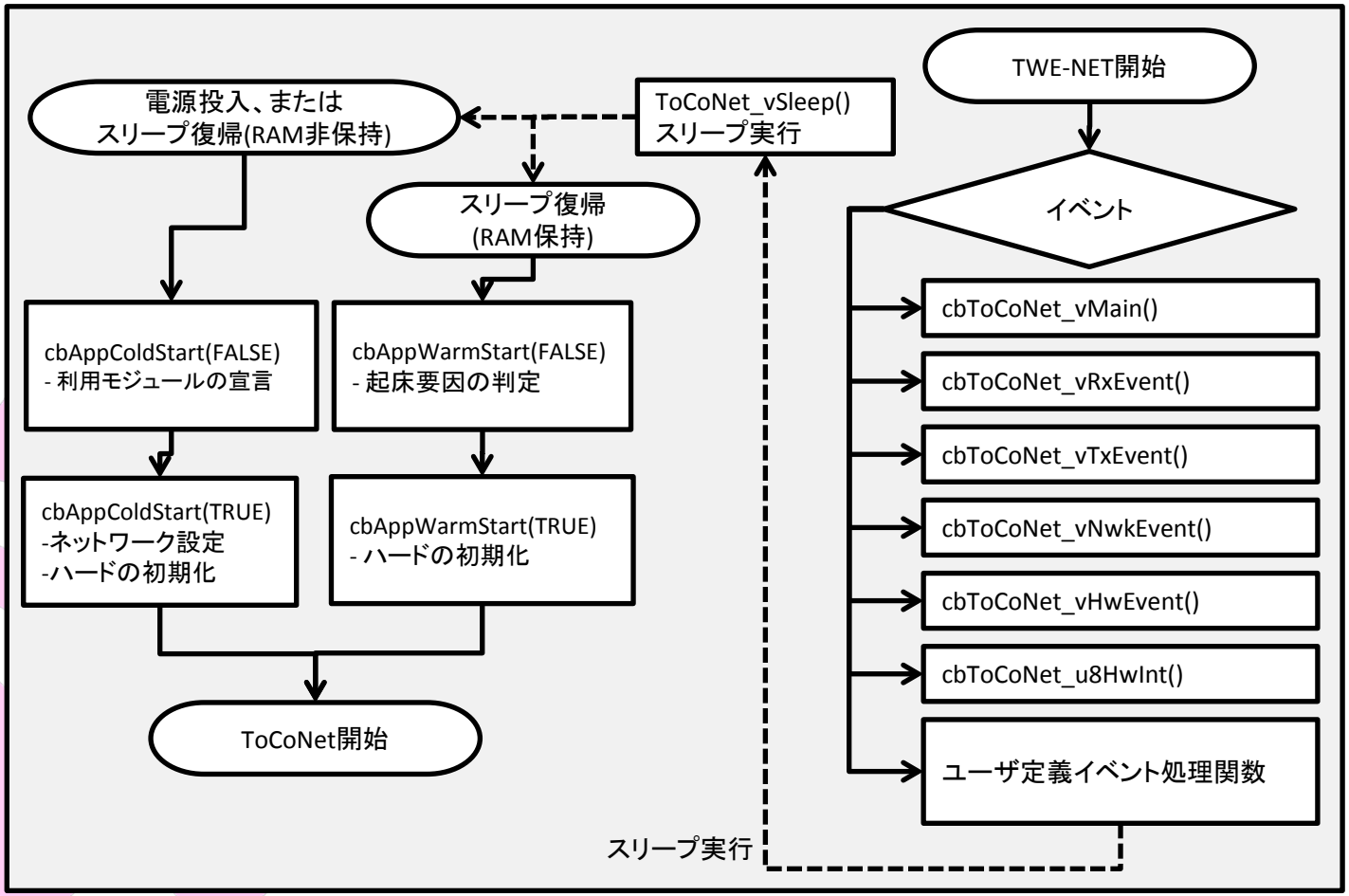
システム組み込みイベント

ユーザ定義イベント処理関数が受け取るイベントを列挙します。

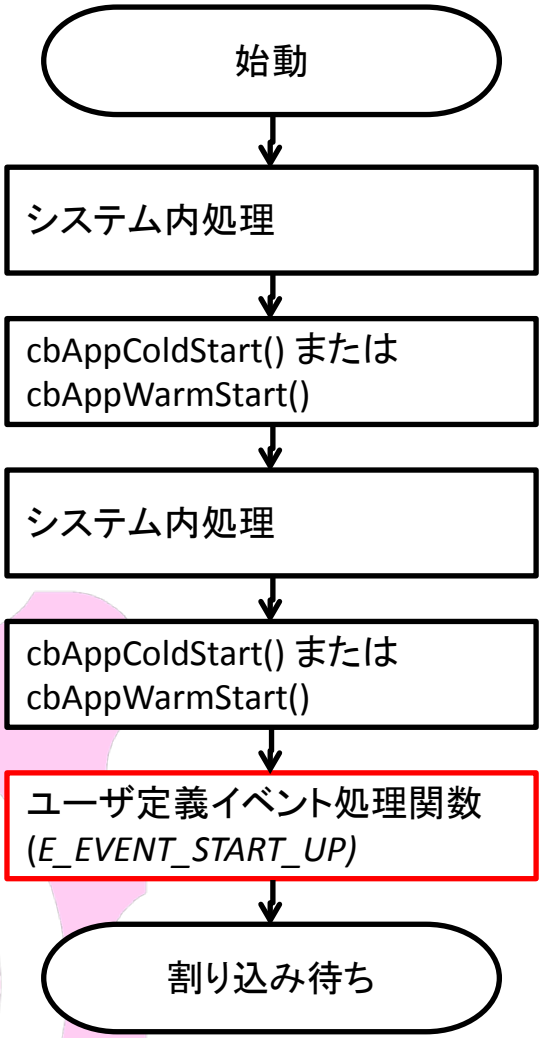
定義名	解説
E_EVENT_START_UP	システム起動（またはスリープ復帰）時に発生します。イベント引数は EVARG_START_UP_WAKEUP_MASK または (EVARG_START_UP_WAKEUP_MASK | EVARG_START_UP_WAKEUP_RAMHOLD_MASK) が与えられます。
E_EVENT_NEW_STATE	状態遷移したときに最初に呼ばれるイベントです。 ※ システム始動時には呼び出されません（START_UPイベントが発生する） イベント引数は未定義。
E_EVENT_TICK_TIMER	システムタイマー(4ms)毎に呼び出されます。 イベント引数は未定義。 ※ システム内のモジュール処理が終わった後に呼ばれます。呼び出しははらつきが発生するため、タイミングを優先したい処理は cbToCoNet_u8HwInt() 中の割り込みハンドラ中の E_AHI_DEVICE_TICK_TIMER に記述します。
E_EVENT_TICK_SECOND	1秒ごとに呼び出されます。 イベント引数は未定義。

イベント処理のフロー

- 上述の全体の動作の流れが下図概略です。次頁から、より詳細なフローを記述します。

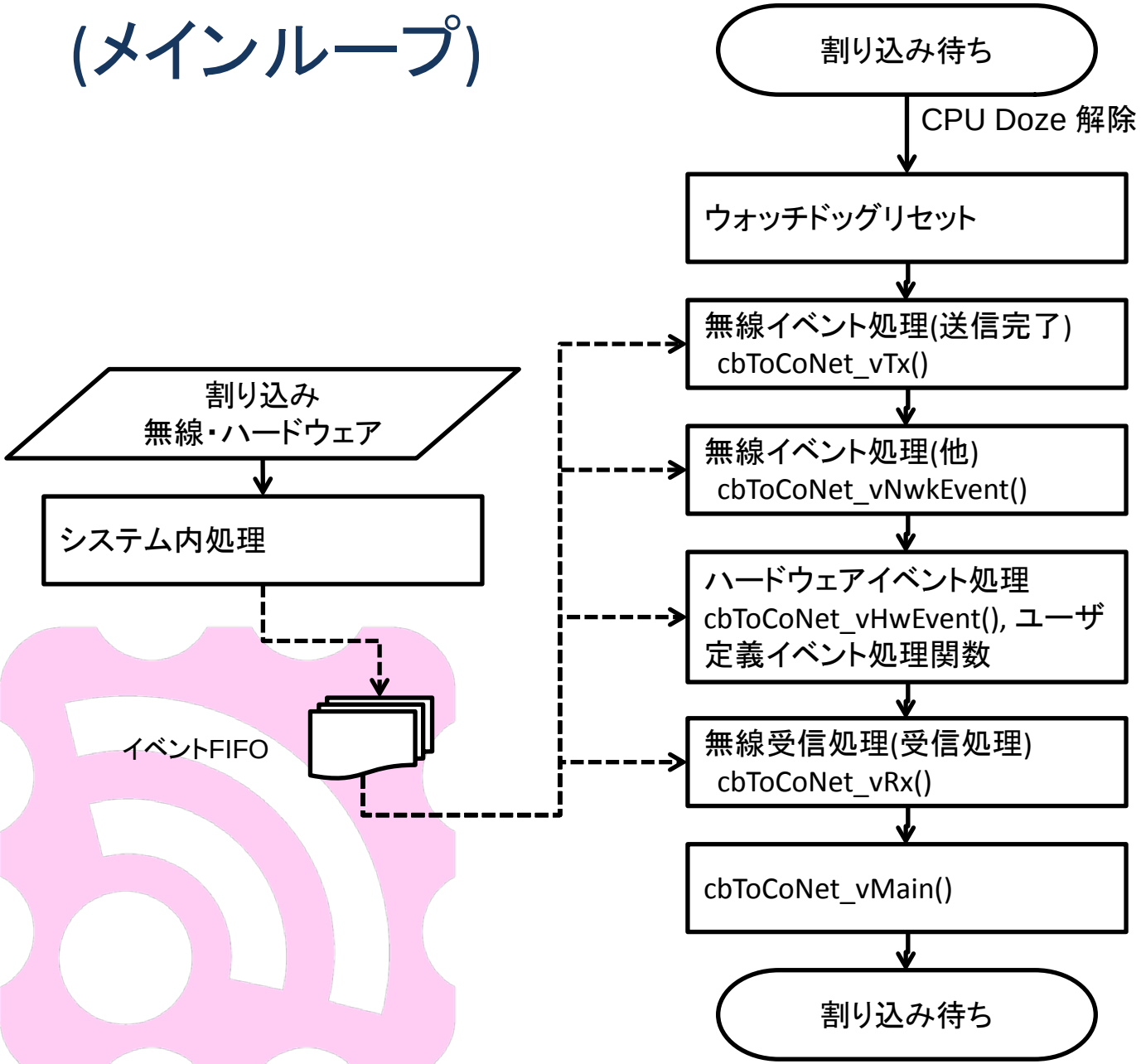


イベント処理フロー (始動)



※ システムがメインループ動作を始める前に、E_EVENT_START_UP イベントがユーザ定義イベント処理関数に発生します。

イベント処理フロー(メインループ)



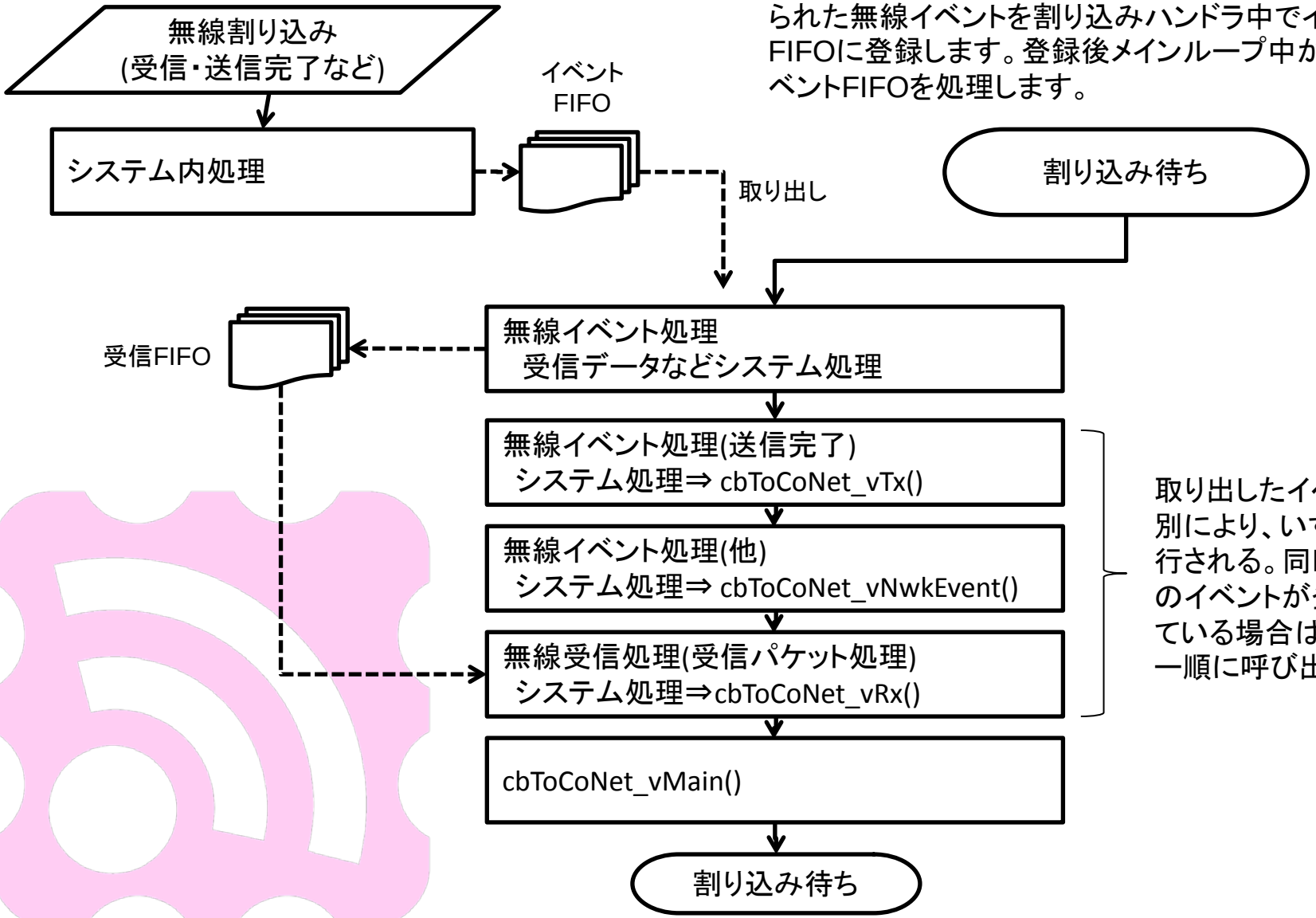
※ 割り込み待ち状況では、割り込みハンドラにより登録されたイベント FIFO キュー順に処理が行われるが、同時にキューに投入された場合、左記の順で処理が行われる。

※ マイコンに無線・ハードウェア関連のイベント(イベントFIFOに登録されないものも含む)が発生すると、最後に cbToCoNet_vMain() が必ず呼び出され、CPU Doze 状態に遷移する。

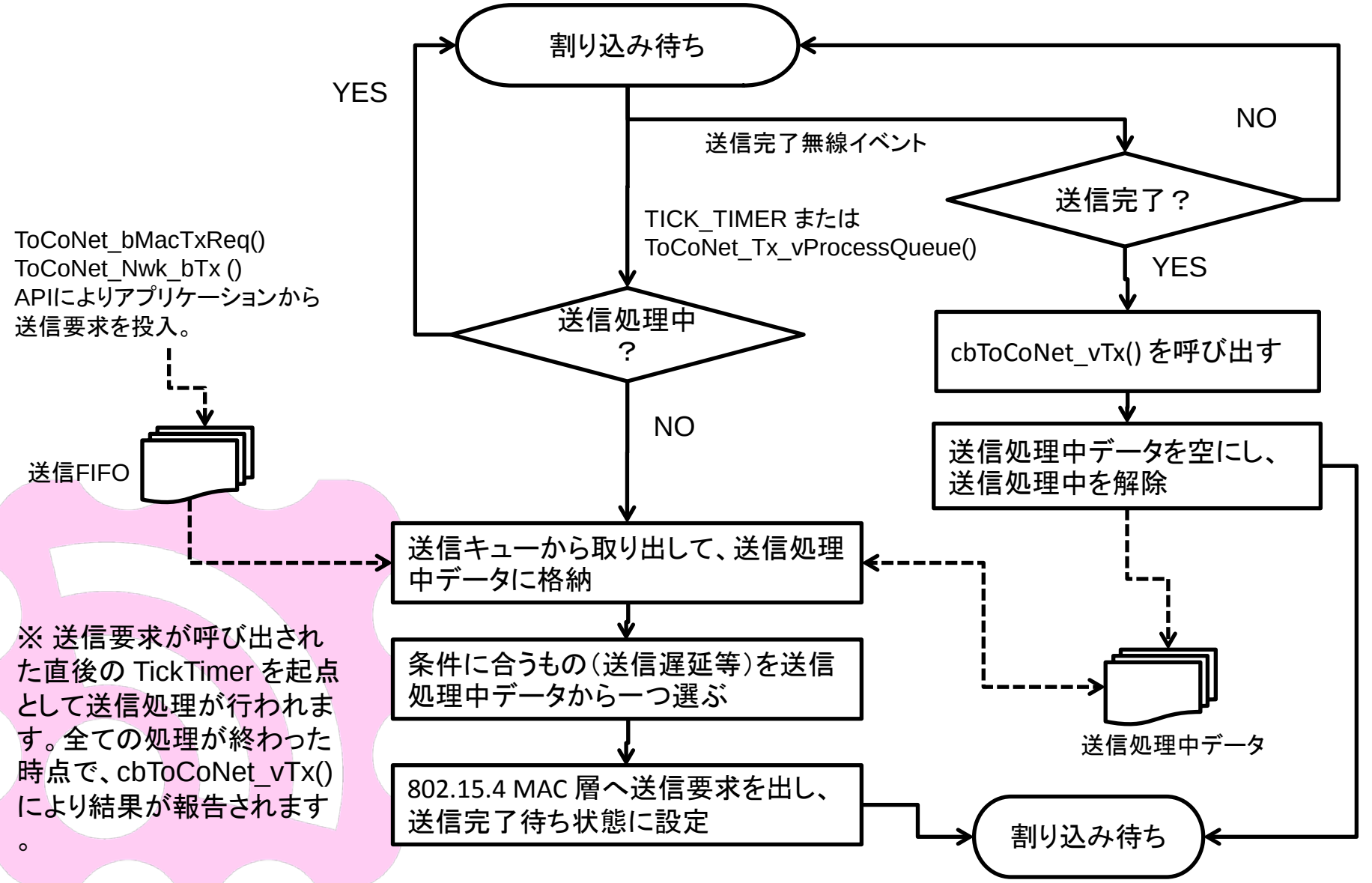
※ 左図は TWE-NETの名ループで、各コールバック関数で相対的に長い時間のかかる処理を行うと、他の処理が滞るため、各処理は簡潔にとどめる。

イベント処理フロー(無線関連)

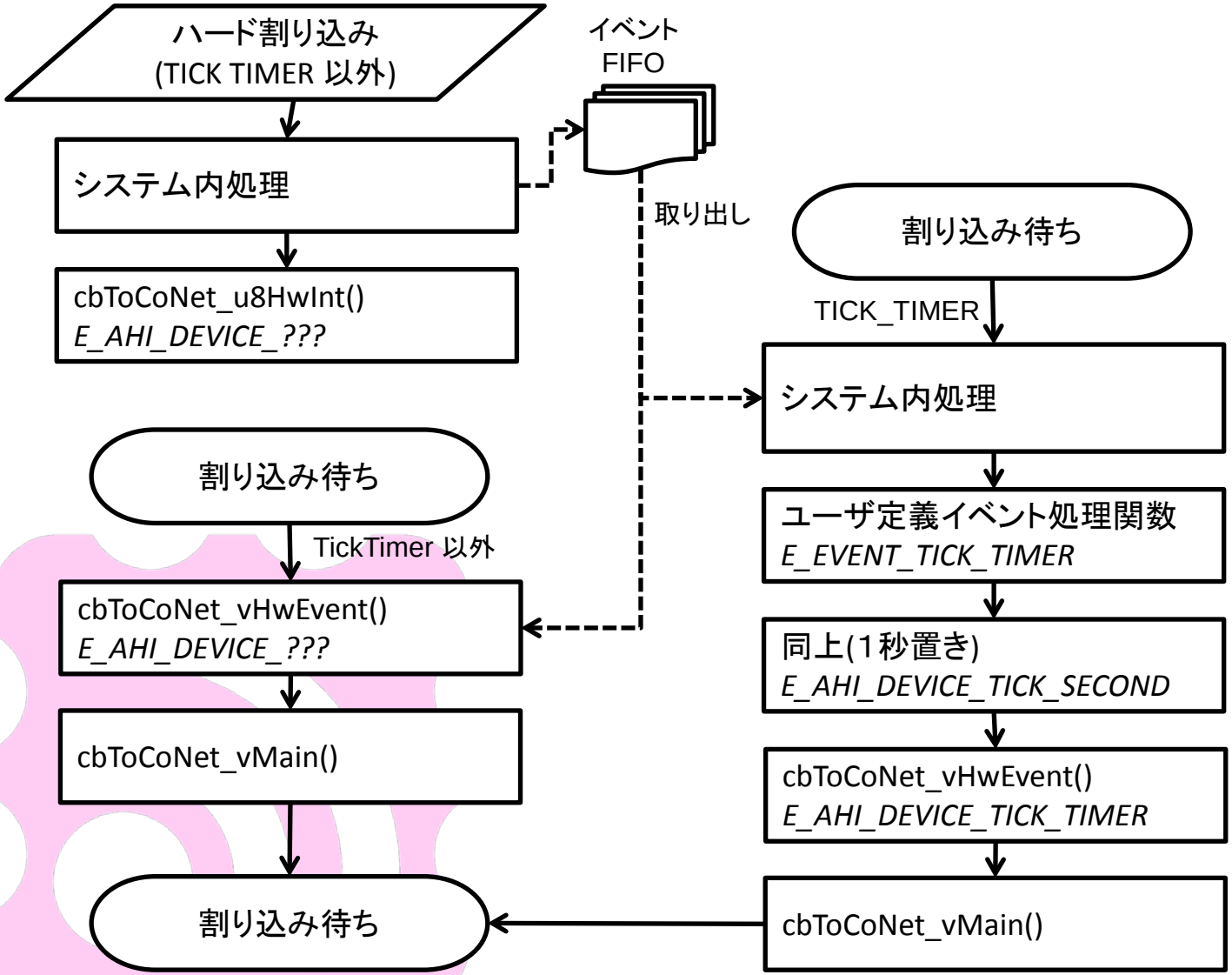
※ 無線関連の処理は、割り込みで(内部処理)得られた無線イベントを割り込みハンドラ中でイベントFIFOに登録します。登録後メインループ中からイベントFIFOを処理します。



イベント処理フロー(無線関連:送信)



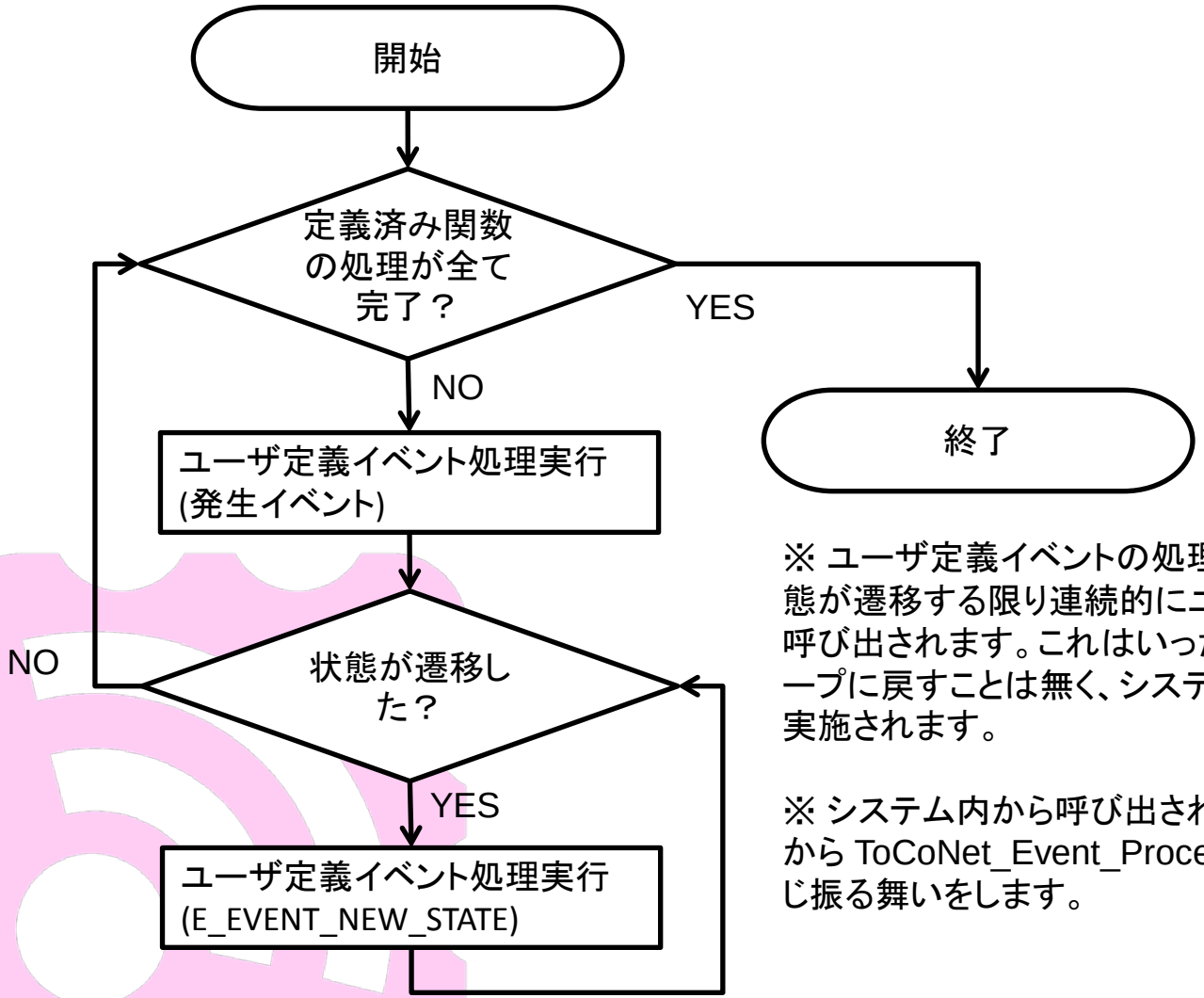
イベント処理フロー(ハードウェア)



※ 左図はハードウェア関連の割り込みのみが発生したときの処理順を示します。メインループ中では、ハードウェア処理は後の方で処理されます。タイミングが重要な割り込み処理は cbToCoNet_u8HwInt() に記述してください。

※ UART 関連の処理を uart.c, serial.c によって実施する場合は、上記コード中で割り込み処理を行うため、割り込みやイベント関数は呼び出されません。
なお UART 割り込み処理後に cbToCoNet_vMain() は呼び出されます。

イベント処理フロー(ユーザ定義イベント)



※ ユーザ定義イベントの処理が呼び出されたときには、状態が遷移する限り連続的にユーザ定義イベント処理関数が呼び出されます。これはいったん制御をシステムのメインループに戻すことは無く、システム内の一関数呼び出し内で実施されます。

※ システム内から呼び出される場合も、アプリケーション側から ToCoNet_Event_Process() により呼び出す場合も同じ振る舞いをします。

ユーザ定義イベントと状態

- システム定義の状態、イベントとの重複を避けるため、ToCoNet_EVENT_APP_BASE, ToCoNet_STATE_APP_BASE 以降に定義します。

定義例

```
typedef enum
{
  E_EVENT_APP_BASE = ToCoNet_EVENT_APP_BASE,//!< ToCoNet 組み込みイベントと重複させないため
  E_EVENT_APP_TICK_A,                       //!< 64FPS のタイマーイベント
  E_EVENT_APP_ADC_COMPLETE,                 //!< ADC完了
  E_EVENT_APP_TX_COMPLETE                   //!< TX完了
} teEventApp;

typedef enum
{
  E_STATE_APP_BASE = ToCoNet_STATE_APP_BASE,//!< ToCoNet 組み込み状態と重複させないため
  E_STATE_APP_WAIT_IO_FIRST_CAPTURE,        //!< 最初のADCやDIの状態確定を待つ
  E_STATE_APP_SLEEPING                      //!< スリープ処理
} teStateApp;
```

メインループ、コールバック関数
cbToCoNet_vMain()

- TWE-NETのアプリケーションメインループからの呼び出されます。
  - UART からの入力処理など頻繁に確認する必要のある処理を記述します。
    - アプリケーションの主処理は、ユーザ定義イベント処理関数で記述することを推奨します。
  - 割り込み解除時に呼び出されます。
    - 主な割り込みは、無線イベント、TickTimer(4ms)

記述例:

```
void cbToCoNet_vMain(void) {
        /* シリアルポートからの入力処理 */
        vHandleSerialInput(SERIAL_FROM_QUEUE);
}
```

送信の概要

- 無線パケットの送信は、送信したいデータ、宛先アドレス、送信元アドレスを指定します。送信時には、その振る舞いを決定するいくつかの設定が可能です。詳しくは tsTxDataApp 構造体の解説を参照してください。
- 送信方法は、大きく分けて2つあり、同報通信（相手先を指定せずに送信）または相手先を指定する送信です。
  - 同報通信では、送信完了を知るための ACK という手続きが取れませんが、取り扱いが大変容易です。App_TweLite でも他対他の通信を容易に実装するため同報通信を使用しています。ACKが取れない点は、複数回送信してカバーします。App_TweLite では常に2回同じパケットを送信しています。
  - 相手先を指定する方法には2つあります。相手先を指定した場合 ACK を確認する事で、速やかに送信完了を認識できます。
    - ショートアドレスを利用するもの。ショートアドレスはアプリケーションから指定する必要が有りますが、パケット長が小さくなる利点が有り、また割り振ったアドレスによって役割を持たせる事も可能です。動的な割り振りを行う場合が有りますが TWE-NETでは、これを行いません。電源断などの状態の変化を同期させるコストが大きいため、シンプルさを追求したい TWE-NETには適切でないと考えます。
    - 拡張アドレス（シリアル番号）を利用するもの。パケットサイズが大きくなることが問題になります。また電波範囲内に存在する無線モジュールのアドレスを知ることは困難です。(本書では記述しない) TWE-NETの中継ネットワーク層の実装では、拡張アドレス(32bit)による管理と、近隣探索 (NeighbourScan) によりこれを解決します。

無線パケット

- パケットの最大長
  - 平文の通常パケット: 92バイト
  - 暗号化利用時: 90バイト
- パケット送受信時の附随データ
  - シーケンス番号(0-255):続き番号。パケットの重複の除去を目的とするため、通常は昇順に値を割り当てます。
  - コマンド番号(0...7): パケット種別を表します。
    - 自前定義は以下の２種類です。
      - TOCONET_PACKET_CMD_APP_DATA = 0（データ）
      - TOCONET_PACKET_CMD_APP_CMD = 1（制御用）
      - 2..7 はアプリケーション定義。

送信API
ToCoNet_bMacTxReq()

- 送信要求をTWE-NETライブラリに伝えます。
  ※ 本関数はネットワーク層非使用時に使用します。
- 引数は tsTxDataApp *psTx
  - 本構造体は、呼び出しスコープ内で破棄されるローカル変数で構いません。データは ToCoNet 管理の送信キューにコピーされます。
- 戻り値は bool_t
  - TRUE なら要求が受け付けられました。(この時点ではまだ送信は開始されていません)
  - FALSE なら要求が受け付けられませんでした。(キューがいっぱいなど)

送信API
ToCoNet_bMacTxReq()

本APIはネットワーク無し送信時に利用します。LayerTree ネットワーク利用時には ToCoNet_Nwk_bTx() を利用します。

- tsTxDataApp 構造体を確保します（必ず0クリアする）
- ペイロードを格納します。以下のマクロが便利です。
  - uint8 *q = sTx.auData;
  - S_OCTET() 1バイト格納
  - S_BE_WORD() 2バイト格納
  - S_BE_DWORD() 4バイト格納
- u8Len （パケット長）を格納。
- u32DstAddr （送信先）を格納。
- u32SrcAddr （送信元）を格納。
- その他オプションを指定します。
- ToCoNet_bMacTxReq() を実行。
- 実行後 cbToCoNet_TxEvent() が呼び出されます。
  - イベントが消失する事は滅多にありませんが、送信完了待ちを行う場合はタイムアウトを設けてください（タイムアウトは送信オプションに依存します。右の例では 100ms が良いでしょう）

```
static int16 i16TransmitIoData() {
  tsTxDataApp sTx;
  memset(&sTx, 0, sizeof(sTx));
  uint8 *q = sTx.auData;

  // ペイロードを構成
  S_OCTET(sAppData.u8AppIdentifier);
  S_BE_WORD(sAppData.sIOData_now.u16Volt);
  ...

  // 送信
  sTx.u8Len = q - sTx.auData; // パケット長
  sTx.u8Cmd = TOCONET_PACKET_CMD_APP_USER_IO_DATA; // パケット種別

  // 送信する
  sTx.u32DstAddr  = TOCONET_MAC_ADDR_BROADCAST; // ブロードキャスト
  sTx.u8Retry     = 0x81; // 1回再送

  // フレームカウントとコールバック識別子の指定
  sAppData.u16TxFrame++;
  sTx.u8Seq = (sAppData.u16TxFrame & 0xFF);
  sTx.u8CbId = sTx.u8Seq;

  /* MAC モードでは細かい指定が可能 */
  sTx.bAckReq = FALSE;
  sTx.u32SrcAddr = sToCoNet_AppContext.u16ShortAddress;
  sTx.u16RetryDur = 4; // 再送間隔[ms]
  sTx.u16DelayMax = 16; // 送信開始タイミングにブレを作る(最大16ms)

  // 送信API
  if (ToCoNet_bMacTxReq(&sTx)) {
    // 成功(処理が終わると cbToCoNet_vTxEvent() が発生する
    return sTx.u8CbId; // 成功
  } else {
    // 失敗
    return -1; // 失敗
  }
}
```

送信完了コールバック関数
cbToCoNet_vTxEvent()

- 送信終了時に呼び出されます。
  - 送信の失敗は、MAC層の再送・アプリケーション再送が全て失敗した場合に報告されます。
- 引数は uint8 u8CbId, uint8 u8Status
  - u8CbId は送信時に指定したID。送信したデータとの紐付けのために使用します。
  - (u8Status & 0x01) == 0 なら失敗。1 なら成功。

リファレンス tsTxDataApp 構造体 (ネットワーク無し)

※ LayerTree ネットワーク使用時の各メンバーの定義は「LayerTree ネットワーク API」(後述) に記載します。

定義名	解説
uint32 u32SrcAddr	送信元アドレス。0xFFFF 未満ならショートアドレス。0x8000000 以上は拡張アドレス。
uint32 u32DstAddr	宛先アドレス。0xFFFF 以下ならショートアドレス。0x8000000 以上は拡張アドレス。 TOCONET_MAC_ADDR_BROADCAST ブロードキャスト送信
uint8 u8Cmd	パケット種別。アプリケーションで自由に設定できる。 値域: 0..7
uint8 u8CbId	コールバックID。送信要求時に設定した値が cbToCoNet_TxEvent() により渡されます。この値によって送信したパケットと送信完了イベントを対応付けます。本値は送信パケットには含まれません。 値域:0-255
uint8 au8Data[]	パケットのペイロード。送信最大バイト数は98バイトです。(宛先が拡張アドレスの場合は92バイトです)
bool_t bAckReq	TRUE ならACK送信要求を設定します。同報通信時には FALSE を設定します。
uint8 u8Retry	ToCoNet で実施する再送回数。MAC層での再送に失敗した場合、ToCoNet で再度パケット送信を試みます。MSB を設定すると、成功失敗に関わらず指定回数の再送を行います。同報通信では殆どの場合送信が成功するため、複数回送信が必要な場合はこのビットを立てます。例えば 0x83 を指定すると、都合4回送信が行われます。値域は 0x0～0xF, 0x80～0x8F。
uint16 u16DelayMin, u16DelayMax, u16RetryDur	送信開始まで u16DelayMin [ms] 遅延し、最大 u16DelayMax [ms] 待ちます。最大値は乱数により決定するため、送信にタイミングに意図的なブレを作るため使用します。送信開始後は u16RetryDur[ms] 間隔で再送が行われます(この間隔は固定)。この処理はシステムのタイマーにより制御されるため、タイマー刻みの精度(標準は4ms)で実行されます。値域には制限はありませんが、一般的には、長くとも 1000ms 程度です。
uint16 bSecurePkt	平文で送信する場合は FALSE, 暗号化する場合は TRUE を指定します。事前に ToCoNet_bRegisterAesKey() により暗号化鍵を登録しておきます。 ※最大利用バイト数が2バイト減少します。

受信コールバック関数
cbToCoNet_vRxEvent()

- 無線パケット受信時に呼び出されます。
- 引数は tsRxDataApp *psRx
  - 本構造体は受信データを格納し、本呼び出しスコープ内のみで有効なデータです。関数終了後にはこのデータ領域は、新しい受信データなどの到着で破壊されます。
  - 構造体の詳細は次ページに記載。
- その他
  - 頻繁に受信される場合は、本関数を終了を遅延すべきではありません。受信キューの解放が遅れ取りこぼしが発生するかもしれません。

リファレンス tsRxDataApp 構造体 (ネットワーク無)

※ LayerTree ネットワーク使用時の各メンバーの定義は「LayerTree ネットワーク API」(後述) に記載します。

定義名	解説
uint8 u8Cmd	パケット種別。送信元で任意に指定できます。データ用、コマンド用といった区分けに使います。 値域：0-7
uint8 u8Len	ペイロード長。後述の au8Data[] に格納されるデータ長。
uint8 u8Seq	シーケンス番号。送信元で任意に設定できます。重複パケットの判定などに使用します。 値域: 0-255
uint32 u32SrcAddr uint32 u32DstAddr	送信元、送信先アドレス。0-0xFFFF の範囲はショートアドレスモード、0x8000000 以上は拡張アドレスによる送受信を示します。0xFFFF はブロードキャスト（同報）通信になります。
uint8 auData[]	ペイロード。送信元で任意に設定できます。
uint8 u8Lqi	受信品質 LQI 値。 値域：0..255（受信時の電界強度に対応し 0 が最弱、255 が最強）
uint8 bSecurePkt	パケットが暗号化されていた場合 TRUE になります。この時点で既に auData[] は平文に復号されています。

ネットワーク イベント コールバック関数
cbToCoNet_vNwkEvent

- 本イベントは、ネットワーク層などの各モジュールによって生成されます。
- 引数は teEvent ev, uint32 u32evarg
  - ev: イベント種別
  - u32evarg: イベント引数
- 詳細は、モジュールの解説を参照してください。

ハードウェア割り込み遅延実行部コールバック関数
cbToCoNet_vHwEvent

- ハードウェア割り込みの遅延処理
  - 遅延実行部で、パケットの送信を含む大きな処理を実行できます。
  - 第一引数、第二引数はデバイスID、ビットマップ
    - AHI ペリフェラル API に定義された値を利用します。
  - 組み込みハードウェア割り込みは TICK_TIMER のみです。
    - *E_AHI_DEVICE_TICK_TIMER* → 4msタイマ

ハードウェア割り込みハンドラ
cbToCoNet_u8HwInt

- ハードウェア割り込みハンドラ
  - 割り込み中に呼ばれるため低遅延です。
  - ごく短い処理のみを記述可能です。
  - FALSE を返すと遅延実行処理を行い、TRUEを返すと遅延実行処理を行いません。
    - TICK_TIMER に対して TRUE を返すと TWE-NETは動作しなくなります。

MONO WIRELESS

LayerTree ネットワーク用API

ToCoNet_NwkLyTr_psConfig()

- tsToCoNet_Nwk_Context*
  ToCoNet_NwkLyTr_psConfig(tsToCoNet_NwkLyTr_Config *_psConf)

- LayerTree ネットワークを設定します。

- 引数
  - tsToCoNet_NwkLyTr_Config *
    - 設定パラメータを格納した構造体

- 戻り値は tsToCoNet_Nwk_Context *
  - LayerTree のネットワークコンテキスト（APIの引数に使う）

リファレンス
tsToCoNet_NwkLyTr_Config 構造体 (1)

LayerTree ネットワークを構成するための基本パラメータです。設定時は0クリアし、設定したいメンバーのみ値を格納します。ToCoNet_NwkLyTr_psConfig() 呼び出し時に渡され、呼び出し後には本構造体領域は破棄してもかまいません。

定義名	解説
uint8 u8Role	LayerTree ネットワーク中の役割を指定します。 TOCONET_NWK_ROLE_PARENT (親機), TOCONET_NWK_ROLE_ROUTER (中継機), TOCONET_NWK_ROLE_ENDDEVICE (子機)
uint8 u8Layer	中継機のレイヤ数を指定します。 値域: 1-63
uint8 u8Second_To_Rescan	上位ノードとの接続が切れたと判断されたときに、再度上位ノードを探索するまでの期間 値域: 3-240 (デフォルト 5) [秒], 0xFF の場合、最探索しない
uint8 u8Second_To_Relocate uint8 u8Ct_To_Relocate	上位ノードとの定期ポーリング（ダミー通信を行う）間隔 値域: 1-240 (デフォルト 3) [秒], 0xFF の場合、ポーリングしない 定期ポーリングの許容連続失敗回数。回数を超えると再探索する。 値域: 1-240 (デフォルト 5) [回], 0xFF の場合、失敗させない
uint8 u8LayerOptions	0x01 なら u8Layer が一つ上のノードにしか接続しない。このオプションを設定すると自身の u8layer = 3 の場合、u8layer= 1 または親機と通信可能であったとしても u8layer = 2 のノードにしか接続しません。
uint8 u8MaxSublayers	同一レイヤに対してサブレイヤを構成するかどうかの設定。0xFF なら構成せず、1 の場合、1ホップのみサブレイヤを許容し、2 の場合2ホップまで許容します。サブレイヤは、u8Layer による上位ノードが見つからない時に、同一レイヤかつ上位レイヤとの接続済みのノードに対し通信先として接続します。上位レイヤが見つかる場合は、上位レイヤを優先接続します。 値域: 0xFF, 1, 2 (デフォルト 2)
uint8 u8minLQI	接続先を決定する場合の LQI (通信品質) の閾値。設定された LQI 以下のノードを除外して接続を試みます。0を設定すると LQI を評価しません。 値域: 0-255 (デフォルト 0)

リファレンス
tsToCoNet_NwkLyTr_Config 構造体 (2)

定義名	解説
uint16 u16TxMinDelayUp_ms uint16 u16TxMaxDelayUp_ms uint16 u16TxMinDelayDn_ms uint16 u16TxMaxDelayDn_ms	上り(子機から親機)パケットの最小および最大遅延。 デフォルトは 10ms および 30ms 下り(親機から子機)パケットの最小および最大遅延。 デフォルトは 10ms および 300ms
uint8 u8TxRetryCtUp uint8 u8TxRetryCtDn	上り、および下りパケットの(アプリケーション)再送回数。 値域: 1-7, 0xFF (デフォルト 3, 0xFF は再送なし)
uint8 u8StartOpt	0x01 の場合、開始時にスキャンせず.u32AddrHigherLayer を接続先とします。(EndDevice のみ)
uint8 u8ResumeOpt	0x01 の場合、レジューム時に接続先のポーリングを行わず、過去のスキャン時に確定した接続先が存在すると仮定します。
uint32 u32AddrHigherLayer	u8StartOpt オプションで接続先を指定する場合に値を格納します。

リファレンス
tsToCoNet_Nwk_Context 構造体

ネットワーク層の基本的なパラメータを定義します。現在のところ LayerTree ネットワークのみ定義されていますが、別の種類のネットワークへの拡張を想定した設計になっています。

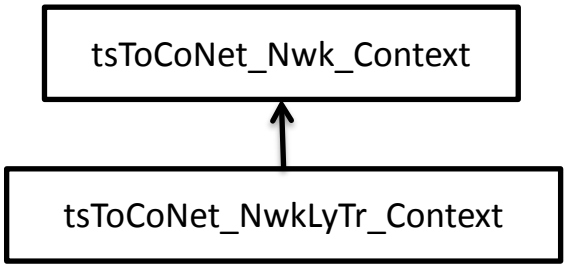
API の戻り値は tsToCoNet_Nwk_Context 構造体ですが、LayerTree での Context 構造体は tsToCoNet_NwkLyTr_Context で定義され、tsToCoNet_Nwk_Context * 構造体ポインタを tsToCoNet_NwkLyTr_Context * にダウンキャストしてLayerTree 特有のデータ構造にアクセスします。

定義名	解説
tsToCoNet_Nwk_Context_Base_Info sInfo	ネットワーク情報の共通構造体。 本構造体は主にデバッグ表示目的として参照できる。 （ただし今後仕様変更などで意味合いなどが変更される可能性が有ります）
sInfo. u8Role	ネットワークの役割(親機・中継機・子機)を保存。
sInfo. u8State	接続状態を示す。 TOCONET_NWK_STATUS_INACTIVE (0x00) : ネットワークが開始されていない TOCONET_NWK_STATUS_START_MASK : ネットワークが開始ビット（未接続の場合も） TOCONET_NWK_STATUS_READY_MASK : ネットワークが接続可能を意味するビット
sInfo. u8Layer	中継機の場合、設定したレイヤ数（接続先のレイヤではなく自身のレイヤ） 子機の場合、接続先のレイヤ数
sInfo. u8LayerSub	中継機の場合、サブレイヤになった時のサブレイヤ数
sInfo.u8NwkTypeId	どの種類のネットワークかを格納する。 TOCONET_NWK_TYPE_LAYER_TREE: LayerTree ネットワーク (tsToCoNet_NwkLyTr_Context * にダウンキャスト可能)
tsToCoNet_Nwk_Context_Base_Method sMethod	ネットワーク操作のメソッド定義 (ユーザの参照は不可)

リファレンス
tsToCoNet_NwkLyTr_Context 構造体

LayerTree 特有の構造体です。tsToCoNet_Nwk_Context * 構造体にアップキャストできるようになっています。
ToCoNet_NwkLyTr_psConfig() 呼び出し後にシステム内部に確保された構造体領域として戻されます。
本構造体もアプリケーションからの書き換えは行わず、デバッグ目的を主として参照のみを行います。

定義名	解説
tsToCoNet_Nwk_Context_Base_Info sInfo	ネットワーク情報の共通構造体。 本構造体は主にデバッグ表示目的として参照できる。
tsToCoNet_Nwk_Context_Base_Method sMethod	ネットワーク操作のメソッド定義 (ユーザの参照は不可)
tsToCoNet_NwkLyTr_Config sConf	設定情報です。ToCoNet_NwkLyTr_psConfig() による設定が行われたときに、規定値の展開などが行われます。
uint32 u32AddrHigherLayer	接続先の上位ノードのアドレス
uint8 u8Ct_Second_To_Rescan	上位を再探索するまでのカウント [秒]
uint8 u8Ct_Second_To_Relocate	上位の存在確認を行うまでのカウント [秒]
uint8 u8Ct_LostParent	上位の存在確認の失敗数カウント（この値が規定回数まで達すると上位喪失として、再探索が行われる）



ToCoNet_Nwk_bInit()

- bool_t ToCoNet_Nwk_bInit(tsToCoNet_Nwk_Context *)

- ネットワークを初期化します。この時点では設定内容の反映など下準備のみで実際の通信は行いません。E_EVENT_START_UP イベント中に記述します。

- 引数
  - tsToCoNet_Nwk_Context *
    - ネットワークコンテキスト
      (LayerTree 設定API ToCoNet_NwkLyTr_psConfig() の戻り値)

- 戻り値は bool_t
  - TRUE なら要求が受け付けられました。
  - FALSE なら要求が受け付けられませんでした。
    - 既に初期化済みの場合。

MONO WIRELESS

ToCoNet_Nwk_bStart()

- bool_t ToCoNet_Nwk_bStart(tsToCoNet_Nwk_Context *)

- ToCoNet LayerTree ネットワークを開始します。通常は E_EVENT_START_UP イベント中に記述し、ToCoNet_Nwk_bInit() 正常終了後に呼び出します。

- 引数
  - tsToCoNet_Nwk_Context *
    - ネットワークコンテキスト
      (LayerTree 設定API ToCoNet_NwkLyTr_psConfig() の戻り値)

- 戻り値は bool_t
  - TRUE なら要求が受け付けられました。
  - FALSE なら要求が受け付けられませんでした。
    - 既に開始状態の場合。

ToCoNet_Nwk_bPause()
ToCoNet_Nwk_bResume()

- bool_t ToCoNet_Nwk_bPause(tsToCoNet_Nwk_Context *)

  ToCoNet LayerTree ネットワークを中断します。RAM保持スリープする直前に実行します。

- bool_t ToCoNet_Nwk_bResume(tsToCoNet_Nwk_Context *)

  ToCoNet LayerTree ネットワークを再開します。E_EVENT_START_UP イベント中で呼び出します。

  ※ 本呼び出しを実行後、Pause 直前にアクセスしていた上位ノードのアドレスに一度ダミー送信を行い、成功すればそのノードとの通信を行います。失敗すれば、再度上位ノードの探索を行います。

- 引数
  - tsToCoNet_Nwk_Context *
    - ネットワークコンテキスト
      (LayerTree 設定API ToCoNet_NwkLyTr_psConfig() の戻り値)

- 戻り値は bool_t
  - TRUE なら要求が受け付けられました。
  - FALSE なら要求が受け付けられませんでした。

送信API
ToCoNet_Nwk_bTx()

- bool_t ToCoNet_Nwk_bTx(tsToCoNet_Nwk_Context *, tsTxDataApp *)

- 送信要求をToCoNetライブラリに伝えます。
- 引数
  - tsToCoNet_Nwk_Context *
    - ネットワークコンテキスト
      (LayerTree 設定API ToCoNet_NwkLyTr_psConfig() の戻り値)
  - tsTxDataApp *
    - 本構造体は、呼び出しスコープ内で破棄されるローカル変数で構いません。データは TWE-NET管理の送信FIFOキューにコピーされます。
- 戻り値は bool_t
  - TRUE なら要求が受け付けられました。(この時点ではまだ送信は開始されていません)
  - FALSE なら要求が受け付けられませんでした。(キューがいっぱいなど)

送信API
ToCoNet_Nwk_bTx()

本APIは LayerTree ネットワーク送信時に利用します。

- tsTxDataApp 構造体を確保します（必ず0クリアする）
- ペイロードを格納します。以下のマクロが便利です。
  - uint8 *q = sTx.auData;
  - S_OCTET() 1バイト格納
  - S_BE_WORD() 2バイト格納
  - S_BE_DWORD() 4バイト格納
- u8Len （パケット長）を格納。
- u32DstAddr （送信先）を格納。
- その他オプションを指定します。
- ToCoNet_bMacTxReq() を実行。
- 実行後 cbToCoNet_TxEvent() が呼び出されます。
  - イベントが消失する事は滅多にありませんが、送信完了待ちを行う場合はタイムアウトを設けてください。右の例では 500ms 程度が良いでしょう。

```
tsToCoNet_Nwk_Context *pContextNwk = NULL;
tsToCoNet_NwkLyTr_Config sNwkLayerTreeConfig; // 0クリア初期化しておくこと

// ユーザ定義イベント処理関数
static void vProcessEvCore(tsEvent *pEv, teEvent eEvent, uint32 u32evarg) {
  switch (pEv->eState) {
  case E_STATE_IDLE:
    If (eEvent == E_EVENT_START_UP) {
      sNwkLayerTreeConfig.u8Role = TOCONET_NWK_ROLE_ENDDEVICE;
      sAppData.pContextNwk =   // 設定
        ToCoNet_NwkLyTr_psConfig(&sAppData.sNwkLayerTreeConfig);
      ToCoNet_Nwk_bInit(sAppData.pContextNwk);
      ToCoNet_Nwk_bStart(sAppData.pContextNwk);
      ... // LayerTree初期化
    }
}

static int16 i16TransmitIoData() {
  tsTxDataApp sTx;
  memset(&sTx, 0, sizeof(sTx));
  uint8 *q = sTx.auData;
  // ペイロードを構成
  S_OCTET(sAppData.u8AppIdentifier);
  S_BE_WORD(sAppData.sIOData_now.u16Volt);
  ...
  sTx.u8Len = q - sTx.auData; // パケット長
  sTx.u8Cmd = TOCONET_PACKET_CMD_APP_USER_IO_DATA; // パケット種別
  sTx.u32DstAddr  = TOCONET_NWK_ADDR_PARENT; // 親機宛
  sTx.u8Seq = ((++sAppData.u16TxFrame) & 0xFF);
  sTx.u8CbId = sTx.u8Seq;
  // 送信API
  if (ToCoNet_Nwk_bTx(pContextNwk, &sTx)) {
    // 成功（処理が終わると cbToCoNet_vTxEvent() が発生する
    return sTx.u8CbId; // 成功
  } else {
    // 失敗
    return -1; // 失敗
  }
}
```

リファレンス tsTxDataApp 構造体 (LayerTree)

定義名	解説
uint32 u32SrcAddr	送信元アドレスを示す。
uint32 u32DstAddr	宛先アドレス。以下の定義済みアドレスは特殊な意味合いを持つ。 TOCONET_NWK_ADDR_PARENT 親機宛に送信する。 TOCONET_NWK_ADDR_NEIGHBOUR_ABOVE 接続している親機へ直接送信する。 TOCONET_NWK_ADDR_BROADCAST 親機からブロードキャスト送信する。
uint8 u8Cmd	パケット種別。アプリケーションで自由に設定できる。値域は 0..7。
uint8 u8CbId	コールバックID。送信要求時に設定した値が cbToCoNet_TxEvent() により渡されます。この値によって送信したパケットと送信完了イベントを対応付けます。本値は送信パケットには含まれません。 値域：0-255
uint8 au8Data[]	パケットのペイロード。送信最大バイト数は84バイトとなります。
bool_t bAckReq	yerTree ネットワーク層では使用されない。ただし LAYERTREE_MININODES で利用される。
uint8 u8Retry	ToCoNet で実施する再送回数。MAC層での再送に失敗した場合、ToCoNet で再度パケット送信を試みます。MSB を設定すると、成功失敗に関わらず指定回数の再送を行います。同報通信では殆どの場合送信が成功するため、複数回送信が必要な場合はこのビットを立てます。例えば 0x83 を指定すると、都合４回送信が行われます。値域は 0x0～0xF, 0x80～0x8F。
uint16 u16DelayMin, u16DelayMax, u16RetryDur	LayerTree ネットワーク層では使用されない。ただし LAYERTREE_MININODES で利用される。
uint16 bSecurePkt	平文で送信する場合は FALSE, 暗号化する場合は TRUE を指定します。事前に ToCoNet_bRegisterAesKey() により暗号化鍵を登録しておきます。 ※最大利用バイト数が2バイト減少します。

リファレンス tsRxDataApp 構造体 (LayerTree)

定義名	解説
uint8 u8Cmd	パケット種別。送信元で任意に指定できます。データ用、コマンド用といった区分けに使います。 値域：0-7
uint8 u8Len	ペイロード長。後述の au8Data[] に格納されるデータ長。
uint8 u8Seq	シーケンス番号。送信元で任意に設定できます。重複パケットの判定などに使用します。 値域: 0-255
uint32 u32SrcAddr uint32 u32DstAddr	送信元、送信先アドレス。拡張アドレスが格納されます。
uint8 auData[]	ペイロード。送信元で任意に設定できます。
uint8 u8Lqi	受信品質 LQI 値。中継されてきた場合は、最後の中継機のパケットの受信強度になります。 値域：0..255（受信時の電界強度に対応し 0 が最弱、255 が最強）
uint8 u8Tick	受信時のミリ秒カウンタ
uint8 u8Hops	中継ホップ数（直接受信した場合は 0）
uint8 u8RouteXOR	中継経路の中継機のアドレスの XOR 値で中継経路の推定に使用します。 直接受信した場合は 0 となり、0x81234567 と 0x82345678 の中継機を経由した場合、0x81 0x23 0x45 0x67 0x82 0x34 0x56 0x78 の XOR 値 0x18 が格納されます。
uint8 u8Lqi1st	最初の中継機が受信したときの LQI 値が格納されます。
uint8 bSecurePkt	パケットが暗号化されていた場合 TRUE になります。この時点で既に auData[] は平文に復号されています。

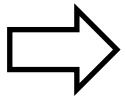
MONO WIRELESS

ユーザ定義イベント処理
PRSEV 補助関数

PRSEV 補助関数の概要

- vProcessEvCore() などのユーザ定義イベント処理関数の処理は、大きな switch case 節からなる場合があります。
- PRSEV 補助関数を用いることで、各 case 節を関数定義に置き換えます。
- 一つのユーザ定義イベント処理関数について1つのCファイルに記述します。(static 定義を多用し名前が衝突するため)

```
void vProcessEvCore(tsEvent *pEv,
    teEvent eEvent, uint32 u32evarg) {
  switch(pEv->eState) {
     case E_STATE_AAA;
        …
        break;

     case E_STATE_BBB;
        …
        break;

     case E_STATE_CCC;
        …
        break;

     ….
  }
```



```
[myEvent.c]

// E_STATE_AAA 関数定義
PRSEV_HANDLER_DEF(E_STATE_AAA, …) {
  …
}

// E_STATE_BBB 関数定義
PRSEV_HANDLER_DEF(E_STATE_BBB, …) {
  …
}

// E_STATE_CCC 関数定義
PRSEV_HANDLER_DEF(E_STATE_CCC, …) {
  …
}
```

関数テーブルの定義

- 定義する関数リスト配列を宣言します。
- 状態定義の数だけ PRSEV_HANDLER_TBL_DEF({状態}) を列挙し、最後に PRSEV_HANDLER_TBL_TRM を記述した配列を用意します。

```
static const tsToCoNet_Event_StateHandler asStateFuncTbl[] = {
  PRSEV_HANDLER_TBL_DEF(E_STATE_IDLE),
  PRSEV_HANDLER_TBL_DEF(E_STATE_RUNNING),
  PRSEV_HANDLER_TBL_DEF(E_STATE_APP_WAIT_TX),
  PRSEV_HANDLER_TBL_DEF(E_STATE_APP_SLEEP),
  PRSEV_HANDLER_TBL_TRM
};
```

実際は以下のように展開されます。vHnd_E_STATE_IDLE が関数名です。

```
static const tsToCoNet_Event_StateHandler asStateFuncTbl[] = {
  { E_STATE_IDLE, vHnd_E_STATE_IDLE },
  ..,
  { E_STATE_VOID, ((void *)0)}
};
```

各関数の定義

- PRSEV_HANDLER_DEV マクロを用いて関数を定義します。

```
PRSEV_HANDLER_DEF(E_STATE_APP_SLEEP, tsEvent *pEv, teEvent eEvent, uint32 u32evarg) {
  if (eEvent == E_EVENT_NEW_STATE) {
    ..
    //  5秒のスリープを実行
    vSleep(5000, TRUE, FALSE);
  }
}
```

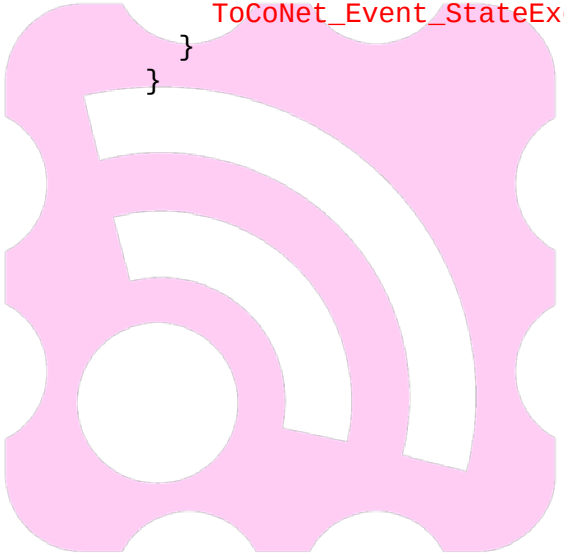
実際は以下のような関数定義に展開します。

```
static void vHnd_E_STATE_APP_SLEEP(tsEvent *pEv, teEvent eEvent, uint32 u32evarg) {
  ..
}
```

ユーザ定義イベント処理関数

- ユーザ定義イベント処理関数からは、ToCoNet_Event_StateExec() を呼び出します。引数に先ほど定義した関数テーブルを渡します。状態に応じた関数をテーブル中から線形検索し、実行します。
- 関数テーブルで処理したくないような場合は、以下の例のように処理を記述できます。

```
void vProcessEvCore(tsEvent *pEv, teEvent eEvent, uint32 u32evarg) {
  if (eEvent == E_EVENT_RX) {
    // この例では E_EVENT_RX が来たときは、ここで処理し、asStateFuncTbl[] の関数は呼び出さない。
  } else {
    ToCoNet_Event_StateExec(asStateFuncTbl, pEv, eEvent, u32evarg);
  }
}
```



MONO WIRELESS

機種依存コードについて

組み込みマクロ

- 半導体のモデルを識別します。
  - JN516x　　Makefile にて TWE_CHIP_MODEL=JN5164 が指定されたときに、自動的に付加されます。(Lite)

```
#if defined(JN514x)
  // 旧モデル用のコード
#elif defined(JN516x)
  // TWE-Lite 用のコード
#endif
```

※ 文中で JN516x が定義されているコード部が TWE-Lite 用です。

MONO WIRELESS

その他のAPI

ToCoNet_u32GetSerial()

- uint32 **ToCoNet_u32GetSerial()**

- 32bit モジュールシリアル番号を得る。このシリアル番号は一意に決まる通信相手を指定する目的でも使用されます。TWEモジュールに表示されている7桁のシリアル番号に 0x80000000 を加えた値となります。

- 引数
  - なし

- 戻り値
  - 32bit のモジュールシリアル番号

ToCoNet_u32GetRand()

- uint32 **ToCoNet_bRegisterAesKey();**

- 32bitの乱数を得ます。乱数の生成方法は sToCoNet_AppContext.u8RandMode で指定します。
- デフォルトはハードウェアによる乱数生成ですが、32Khz タイマーに高精度外部タイマーを用いた場合、期待した乱数は得られません。またハードウェアによる方法では 16bit 値の取得となるため上位は下位をビット変換したものとなります。

- 戻り値：32bitの乱数

ToCoNet_vSleep()

- **void ToCoNet_vSleep(uint8 u8Device, uint32 u32Periodms, bool_t bPeriodic, bool_t bRamOff);**

- ウェイクアップタイマーによるスリープを行います。
- 引数
  - u8Device 使用するウェイクアップタイマー。
  - u32Periodms スリープ期間。
  - bPeriodic 周期スリープを行うかどうか。
    周期スリープを指定した場合、前回の起床時間からの経過時間を差し引いて次の起床を行います。すでに次の周期をすぎていた場合は本関数呼び出し時点から指定期間のスリープとし、周期スリープを指定しない場合、本関数呼び出し時点から指定期間のスリープを行います。
  - bRamOff FALSE なら RAM を保持した状態でスリープを行います。TRUE なら、RAMを保持しないスリープを行います。
- 注意
  - ユーザ定義イベント処理関数から呼び出す場合、ToCoNet_vSleep() の処理内で *E_EVENT_TOCONET_ON_SLEEP* イベントでのユーザ定義イベント処理関数が呼び出されます。例えば以下のように E_USER_STATE_SLEEPING といった状態を定義し E_EVENT_NEW_STATE イベント中に処理するといった処理にします。

```
case E_STATE_SLEEPING:
  if (eEvent == E_EVENT_NEW_STATE) { ToCoNet_vSleep(…); }
  break;
```

ToCoNet_vDebugInit

- **void ToCoNet_vDebugInit(tsFILE*pSer);**

- ToCoNet デバッグ出力の初期化。初期化済みの tsFILE* (UART)を渡す。
- 引数
  - pSer 出力先構造体（UART）

ToCoNet_vDebugLevel

- **void ToCoNet_vDebugLevel(uint8 u8lvl);**

- ToCoNet スタック内のデバッグ出力レベルを指定する。
  - デバッグ目的の出力ですので、書式の解説は行いません。スタックソースコードを入手したユーザは、スタックコード中の DGBOUT(), DGBFTL() を参照してください。
- 引数
  - u8lvl デバッグレベルの指定。u8lvl以下のメッセージを出力する。

ToCoNet_u32GetVersion()

- uint32 **ToCoNet_u32GetVersion();** ToCoNet ライブラリ
  uint32 **ToCoNet_u32GetVersion_LibEx();** ToCoNetExt ライブラリ
  uint32 **ToCoNet_u32GetVersion_LibUtils();** ToCoNetUtils ライブラリ

- ライブラリのバージョンを取得します。
- 戻り値 バージョン番号。0x00bbccdd bb:メジャー cc:マイナー dd:ビルド

ToCoNet_u32GetRand()

- uint32 **ToCoNet_u32GetRand();**
  uint16 **ToCoNet_u16GetRand();**

- 32bit/16bit の乱数を得ます。乱数の生成方法は sToCoNet_AppContext.u8RandMode で指定します。
- デフォルトはハードウェアによる乱数生成ですが、32Khz タイマーに高精度外部タイマーを用いた場合、期待した乱数は得られません。またハードウェアによる方法では 16bit 値の取得となるため上位は下位をビット変換したものとなります。
- ハードウェアタイマに不都合が有る場合は、ソフトウェア乱数を使用ください。

- 戻り値：32bitの乱数

ToCoNet_u16RcCalib()

- uint16 **ToCoNet_u16RcCalib(uint16 u16val);**
  uint16 **ToCoNet_u16GetRcCalib();**

- RCタイマーのキャリブレーションを行います。
- 引数
  - u16val : 0 の場合、キャリブレーションを実行します。
    5000-15000 の場合、過去に実行したキャリブレーションを設定します
    0xffff の場合、キャリブレーション値取得します。

- 戻り値 : RCキャリブレーション値

時間 T[sec] としたときに、RCタイマーのカウント数 N は 5000～15000の値を持ち、
N = (10000/キャリブレーション値) x 32000 x T
となります。

void ToCoNet_vRfConfig()

- void **ToCoNet_vRfConfig();**

- MAC 層のパラメータ変更を行います。sToCoNet_AppContext の以下に挙げるメンバーが変更可能です。
  - u32ChMask
  - u16ShortAddress
  - u8Channel
  - u8CPUClk
  - u8TxPower
  - bRxOnIdle (FALSE にするとパケットが受信できなくなる)
- 注意
  - LayerTreeネットワーク利用時は、u8TxPower, u8CPUClk, bRxOnIdle のみ変更可能。
  - 送信中、受信中などはチャネル変更がブロックされます。
  - チャネルマネージャ動作中は、チャネル変更は行われません。

void ToCoNet_vChConfig()

- void **ToCoNet_vChConfig()**

- チャネルのみを変更します。sToCoNet_AppContext.u8Channel を書き換えて実行します。
- 注意
  - LayerTreeネットワーク利用時は、u8TxPower, u8CPUClk, bRxOnIdle のみ変更可能。
  - 送信中、受信中などはチャネル変更が遅延実行されます。

ToCoNet_bRegisterAesKey()

- bool_t **ToCoNet_bRegisterAesKey(uint8 *pu8Key, tsCryptDefs *sCryptDefs);**

- 通信時に用いる暗号化鍵を設定します。この処理はシステム起動時の E_EVENT_START_UP イベントで記述します。

- 引数
  - pu8Key　　16バイトの鍵データへのポインタ
  - cCrypyDefs　　予約(NULLにします)

- 戻り値：TRUE: 鍵が登録できた FALSE: 鍵の登録に失敗した

- 注意
  - 鍵自体を暗号化する事は出来ません。EEPROM への保管・ファームウェア読み出しにより解析されてしまうことを防ぎたい場合は、鍵の秘匿化のためのアルゴリズムを含めるようにしてください。

ToCoNet_bIsStrong()

- bool_t **ToCoNet_bIsStrong();**
- 外部にパワーアンプを用いたモジュールの場合 TRUE を戻します。
- 戻り値 : TRUE: パワーアンプつきモジュール FALSE: 通常モジュール

ToCoNet_Tx_vProcessEventQueue()

- void **ToCoNet_vProcessEventQueue();**

- パケット送信要求を即時に依頼する。

- 使用方法

  送信API実行直後に呼び出す。

  ```
  ToCoNet_bMacTxReq(); //  または ToCoNet_Nwk_bTx();
  ToCoNet_Tx_vProcessQueue(); // 送信処理をタイマーを待たずに実行します。
  ```

- 注意：
  - 処理前のキューがある場合や、ライブラリ内での送信があるような場合、直前の送信要求が処理されることは保証されません。
  - 送信遅延が設定されている場合は、速やかに送信処理が行われません。
  - ネットワーク層の送信の場合、LayerTree_Mininode のみ利用可能です。

uint32 u32TickCount_ms

- extern uint32 u32TickCount_ms

- システムの稼働カウンタで、ミリ秒単位としてカウントアップされます。カウントアップの刻みは sToCoNet_AppContext.u16TickHz で決定されます。

- 注意
  - 外部から値の変更不可
  - スリープ復帰後のカウント値は 0 である事を保証しない

PANIC について

PANIC は、TWE-NETがシステム動作の継続を不可能と考えた場合に発生し、ネットワークイベントして通知されます。

- PANIC構造体定義

```
typedef struct {
    bool_t bCancelReset; //!< TRUE を格納するとリセットをキャンセルする
    uint8 u8ReasonCode; //!< 要因コード
    uint32 u32ReasonInfo; //!< 要因の補助情報
    string strReason; //!< 要因の文字列情報
} tsPanicEventInfo;
```

- PANIC 処理の参考コード

```
void cbToCoNet_vNwkEvent(teEvent eEvent, uint32 u32arg) {
    switch(eEvent) {
    case E_EVENT_TOCONET_PANIC:
        if (u32arg) {
            tsPanicEventInfo *pInfo = (void*)u32arg;
            V_PRINTF( “PANIC! %d/%s“, pInfo-> u8ReasonCode, pInfo->strReason);
            pInfo->bCancelReset = TRUE; // TRUE を設定すると、直後にリセットしない。
                                        // ただしその後の動作は未定義である。
        } break;
…
```

- PANIC 要因のID

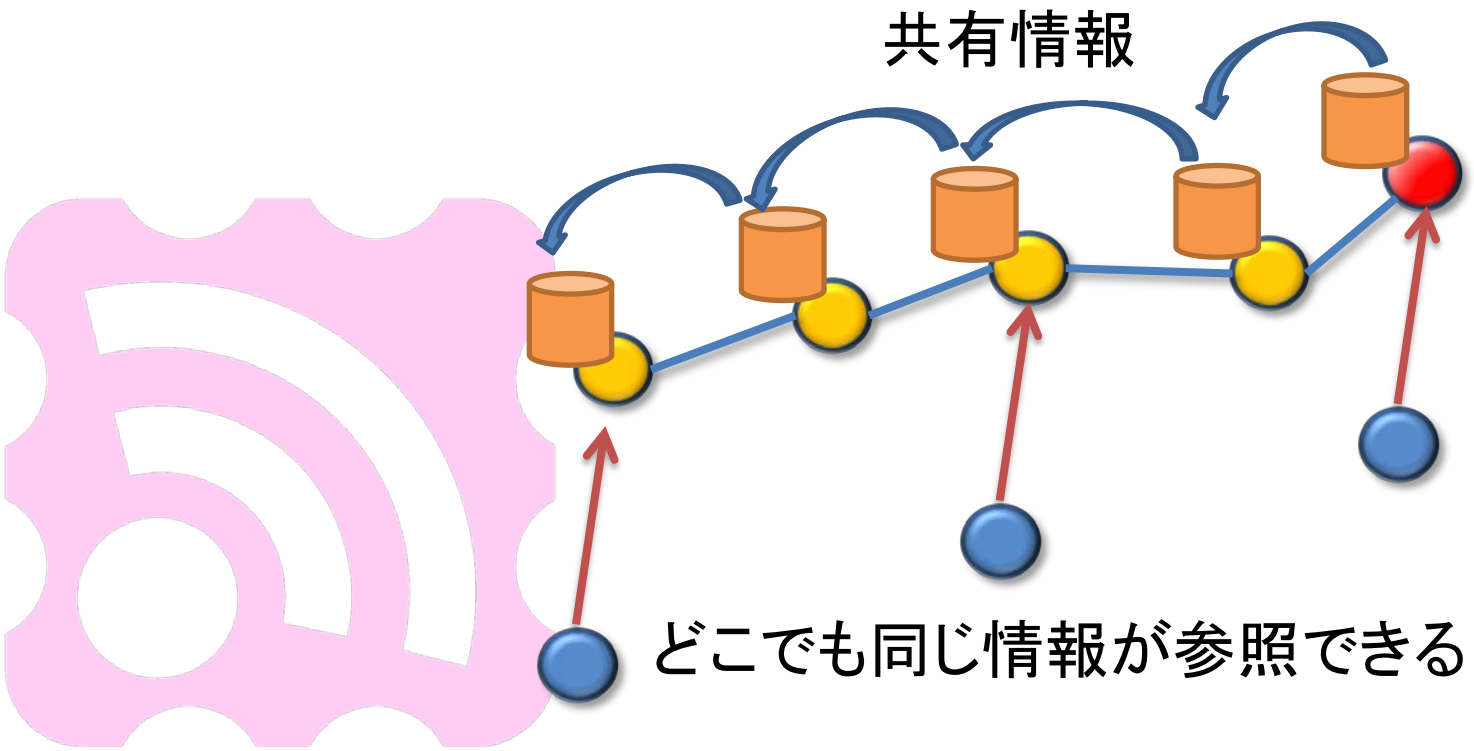
TOCONET_PANIC_TX_FAIL_COUNT (MAC層内で送信処理の失敗が継続した), TOCONET_PANIC_LAYERTREE_SCAN_FAIL_COUNT, TOCONET_PANIC_LAYERTREE_SCAN_TIMEOUT, TOCONET_PANIC_LAYERTREE_LOCATE_TIMEOUT (LayerTree 内での接続先探索処理の失敗回数、探索処理のタイムアウト)

MONO WIRELESS

モジュール

MessagePool (メッセージプール) 概要 (1)

- メッセージプールは、親機から各子機への情報展開を行い、各中継機から共有情報を得ることを目的とします。
  - 共有情報は 64 バイトです。
  - 親機からの情報展開は、中継を含む同報通信により実施されます。中継に失敗した時は以降のノードでは更新されません。



MessagePool (メッセージプール)
概要 (2)

- データの展開タイミング
  - 親機ノードから **ToCoNet_MsgPl_bSetMesage** を発行した時に、全中継機に展開されます。定期的な送信は行う場合はアプリケーションから行います。
- データ展開の失敗
  - 中継機が中継に失敗した場合は、以降のノードに対するデータ展開は行われません。また失敗を検出する事も出来ません。
- 複数のスロット
  - １回のデータ展開時に1スロットのみ展開されます。
  - １回のデータ要求時には1スロットのみ要求可能です。

MessagePool (メッセージプール)
使用方法：共通

- ToCoNet_USE_MOD_NWK_MESSAGE_POOL モジュールを定義します。

```
#define ToCoNet_USE_MOD_NWK_MESSAGE_POOL // モジュール宣言部にて
```

- 始動時の E_EVENT_START_UP イベントで初期化します。

```
ToCoNet_MsgPl_vConfig(NULL);
```

- 親機側ではメッセージの展開を行います。

```
uint8 au8dat = “abcde”;
uint8 u8len = strlen(au8dat);
ToCoNet_MsgPl_bSetMesage(0, 0, u8len, au8dat); // メッセージの設定
```

- 子機側では、メッセージの要求を行います。接続中の中継機または親機に要求を行い、結果が得られた時点で cbToCoNet_vNwkEvent() が呼び出される。

```
ToCoNet_MsgPl_Request(0); // メッセージの要求

… 中継機からの応答が来るとイベントが発生する。
void cbToCoNet_vNwkEvent(teEvent eEvent, uint32 u32arg) {
switch(eEvent) {
case E_EVENT_TOCONET_NWK_MESSAGE_POOL:
  if (u32arg) {
    uint8 au8dat[TOCONET_MOD_MESSAGE_POOL_MAX_MESSAGE];
    tsToCoNet_MsgPl_Entity *pInfo = (void*)u32arg;
    memcpy(au8dat, pInfo->au8Message, pInfo->u8MessageLen);
  }}}
```

MessagePool (メッセージプール)
関数： `ToCoNet_MsgPl_bSetMesage`

- void ToCoNet_MsgPl_vConfig(tsToCoNet_MsgPl_Config *_psConf) ;
- 概要
  メッセージプールの初期化を行う。
  ※ E_EVENT_START_UP イベント中に呼び出す。
- パラメータ
  - _psConf 設定用の構造体
    ※ 現時点では設定内容は存在しないため NULL を指定する。

MessagePool (メッセージプール)
関数： ToCoNet_MsgPl_bSetMesage

- bool_t ToCoNet_MsgPl_bSetMessage(uint8 u8Slot, uint16 u16Lifs_s, uint8 u8len, uint8 *puDat);
- 概要
  メッセージプールにメッセージを設定します。これは親機(Parent)のみ実行可能です。この関数が呼び出されると速やかに各中継機へメッセージを伝達します。このメッセージは下り方向の中継込みのブロードキャストで実行されているため、頻繁な呼び出しではネットワークの負荷は大きくなり、また伝達に成功しない可能性も考慮する必要が有ります。
- パラメータ
  - u8Slot スロット番号 (0-7)
  - u16Lifs_s 寿命[秒] (未使用)
  - u8len メッセージ長 (0: 未設定にする 1-64:データを設定する)
  - puDat データへのポインタ。最大64バイトまで設定できる。(u8len == 0 の時は NULL を指定する)
- 戻り値
  - TRUEなら成功で要求が受けつけられた。
  - FALSEの要因はパラメータエラー

MONO WIRELESS

MessagePool (メッセージプール)
関数： `ToCoNet_MsgPl_bRequest`

- bool_t ToCoNet_MsgPl_bRequest(uint8 u8Slot);
- 概要
  メッセージプールのデータ要求を行います。
  本要求は中継機や親機では、*E_EVENT_TOCONET_NWK_MESSAGE_POOL* イベントが発生します。
- パラメータ
  - u8Slot スロット番号 (0-7) または複数スロットビットマスク
    ※ MSB (0x80) のビットを設定するとスロット #0 ～ #6 までを複数指定して要求を発行できる。スロット 0,2,3 bit0, 2, 3 を設定して 0x8D となる。最大スロット数は3までで、到達順は保証しない。
- 戻り値
  - TRUE なら成功で要求が受けつけられた。
  - FALSE の場合は失敗で、パラメータエラー、または要求送信に失敗した。
- 注意
  - 通信失敗による要求失敗が発生しうるため。失敗の検出はできないため、タイムアウトにより失敗とする。

MONO WIRELESS

MessagePool (メッセージプール)
関数： ToCoNet_MsgPl_bRequest_w_Payload_to_Parenet

- bool_t ToCoNet_MsgPl_bRequest_w_Payload_to_Parenet(uint8 u8Slot, tsTxDataApp *psTx);
- 概要
  メッセージプールのデータ要求を行います。
  本要求は中継機や親機では、*E_EVENT_TOCONET_NWK_MESSAGE_POOL* イベントが発生します。
- パラメータ
  - u8Slot : ToCoNet_MsgPl_bRequest() の解説を参照
  - psTx : 送信データ (ペイロード部の上限は4バイト減少します)
- 戻り値
  - TRUE なら成功で要求が受けつけられた。
  - FALSE の場合は失敗で、パラメータエラー、または要求送信に失敗した。
- 注意
  - ToCoNet_MsgPl_bRequest() の解説を参照

MessagePool (メッセージプール)
メッセージ： *E_EVENT_TOCONET_NWK_MESSAGE_POOL*

- 概要
  メッセージプールが展開されてきたときに発生する。子機の場合は ToCoNet_MsgPl_bRequest() による応答、また中継機の場合は親機からのメッセージの展開時に発生する。
- イベントパラメータ
  - tsToCoNet_MsgPl_Entity 構造体へのポインタ

```
void cbToCoNet_vNwkEvent(teEvent eEvent, uint32 u32arg) {
switch(eEvent) {
case E_EVENT_TOCONET_NWK_MESSAGE_POOL:
  if (u32arg) {
    uint8 au8dat[TOCONET_MOD_MESSAGE_POOL_MAX_MESSAGE];
    tsToCoNet_MsgPl_Entity *pInfo = (void*)u32arg;
    memcpy(au8dat, pInfo->au8Message, pInfo->u8MessageLen);
    if (pInfo.bGotData) { … } // データが格納されている時の処理
  }}}
```

  - 本構造体は固定永続的なメモリ確保が行われているので、メッセージ発生後に参照してもかまわない。

MessagePool (メッセージプール)
メッセージ： *E_EVENT_TOCONET_NWK_MESSAGE_POOL_REQUEST*

- 概要
  中継機または親機が子機からのメッセージプールの要求を受け取った時に発生する。主にデバッグ目的で使用する。
- イベントパラメータ
  - tsToCoNet_MsgPl_Entity 構造体へのポインタ

```
void cbToCoNet_vNwkEvent(teEvent eEvent, uint32 u32arg) {
switch(eEvent) {
case E_EVENT_TOCONET_NWK_MESSAGE_POOL_REQUEST:
  if (u32arg) {
    uint8 au8dat[TOCONET_MOD_MESSAGE_POOL_MAX_MESSAGE];
    tsToCoNet_MsgPl_Request *pReq = (void*)u32arg;
    uint32 u32Addr = pReq->u32Addr; // 要求元のアドレス
    uint8 u8Slot = pReq->u8Slot;    // 要求されたメッセージスロット番号

  }}}
```

MessagePool (メッセージプール)
構造体： tsToCoNet_MsgPl_Entity

- 概要
  メッセージデータを格納する構造体。
- メンバー
  - uint8 au8Message[TOCONET_MOD_MESSAGE_POOL_MAX_MESSAGE];
    メッセージ内容
  - uint32 u32TickGot
    メッセージ取得時刻 (u32TickCount_ms値)
  - uint16 u16Life_s
    メッセージに寿命(未使用)
  - uint8 u8MessageLen
    メッセージ長(0はデータ無し、1～TOCONET_MOD_MESSAGE_POOL_MAX_MESSAGE)
  - uint8 u8Slot;
    メッセージのスロット(0～TOCONET_MOD_MESSAGE_POOL_MAX_ENTITY-1)
  - uint8 u8LqiRecv;
    メッセージ取得時に LQI
  - bool_t bGotData
    構造体に値が入っていれば TRUE, 未設定・無効時は FALSE

MessagePool (メッセージプール)
構造体： tsToCoNet_MsgPl_Request

- 概要
  メッセージ要求を受け取った時にイベントで報告される構造体。主にデバッグ目的で使用する。
- メンバー
  - uint32 u32Addr
    要求元のアドレス
  - uint8 u8Slot
    要求スロット番号

MessagePool (メッセージプール)
構造体： tsToCoNet_MsgPl_Request

- 概要
  メッセージ要求を受け取った時にイベントで報告される構造体。主にデバッグ目的で使用する。
- メンバー
  - uint32 u32Addr
    要求元のアドレス
  - uint8 u8Slot
    要求スロット番号

NBSCAN(近隣探索)
概要

- Neighbour Scan は、近隣にどのようなノードが存在するか探索すします。
  - チャネルマスクに指定された各チャネルに対して、ブロードキャストパケットにより応答要求を出力します
  - オプションで指定された時間中、スレーブ側のモジュール(mod_NbScan_Slave) を登録したノードからの応答を待ちます。
  - 近隣ノードは探索要求を受信後、探索条件に合致していた場合、ランダムなタイミングで送信元へ応答を返します。
  - 探索条件に合致するノードの情報を内部に記録し、LQIが大きいものを優先して登録します。
  - 探索が完了すればアプリケーションまたは呼び出し元のモジュールへのコールバックによりイベント通知を行います。
  - LayerTree ネットワーク利用時は利用不可。

NBSCAN(近隣探索)
開始関数

- uint8 ToCoNet_NbScan_bStart (uint32 *u32ChMask*, uint16 *u16Dur_ms*)
  - u32ChMask は探索チャネル。u16Dur_ms は各チャネルの探索時間。おおむね 50ms 以上を推奨値とします。
  - 近隣探索を開始します。本探索では、各チャネルにスキャン要求パケットを送信し、NeighbourScan Slave モジュールをロードされたノードからパケット返答を受信 することで近隣の探索を行います。
  - 最大探索数は１６で、LQI の大きなものからリストされます。
- uint8 ToCoNet_NbScan_bStartToFindAddr (uint32 *u32ChMask*, uint32 *u32Addr*)
  - 既にアドレスが判っているノードを探索します。各チャネルにパケット送信を試み、パケット送信に成功したチャネルを返します。
  - u32ChMask 探索対象チャネル, u32Addr 探索するアドレス
- 結果は cbToCoNet_vNwkEvent() より *E_EVENT_TOCONET_NWK_SCAN_COMPLETE* イベントにて報告されます。
  - イベント引数が tsToCoNet_NbScan_Result 構造体へのアドレス

NBSCAN(近隣探索)
応答イベント(cbToCoNet_vNwkEvent)

```
case E_EVENT_TOCONET_NWK_SCAN_COMPLETE:
        _C {
            tsToCoNet_NbScan_Result *pNbsc = (tsToCoNet_NbScan_Result *)u32arg;

            if (pNbsc->u8scanMode & TOCONET_NBSCAN_NORMAL_MASK) { // 通常探索による結果
                // 全チャネルスキャン結果
                for(i = 0; i < pNbsc->u8found; i++) { // 見つかった数
                    tsToCoNet_NbScan_Entitiy *pEnt = &pNbsc->sScanResult[pNbsc->u8IdxLqiSort[i]];
                    if (pEnt->bFound) {
                        // エントリが見つかった
ここで pEnt->u8ch(チャネル), pEnt->u32addr(ロングアドレス), pEnt->u16addr(ショートアドレス), pEnt->u8lqi(LQI値) が参照可能。
                    }
                }
            } else if (pNbsc->u8scanMode & TOCONET_NBSCAN_QUICK_EXTADDR_MASK) { // アドレス指定探索の結果
                if(pNbSc->u8found) {
                    tsToCoNet_NbScan_Entitiy *pEnt = &pNbsc->sScanResult[0];
                        // 見つかった場合は一つだけなので最初の要素を取り出す。
                }
            }
        }
```

ENERGYSCAN(チャネル受信レベル)
概要

- ToCoNet_EnergyScan_bStart(uint32 u32ChMask, uint8 u8Scale)
    - u32ChMask 測定するチャネルマスク(ch16のみなら 1UL<<16 を指定、ch16,17なら 1UL<<16|1UL<<17 を指定)
    - u8PreScale 測定時間。およそ(2^u8PreScale)*16[ms]
    - 戻り値 TRUEなら要求がMAC層に伝達された
  - 本関数はエナジースキャンを開始します。関数呼び出し後は速やかに無線送受信が不可能になるため、無線送受信が無い事を確認の上使用する。
  - 完了後 E_EVENT_TOCONET_ENERGY_SCAN_COMPLETE イベントを発生します。
  - 約1200msでタイムアウトします。多チャネルを一度にスキャンする場合は、タイムアウト時間を超えないように設定します。
  - LayerTree ネットワーク利用時は使用不可。

ENERGYSCAN(チャネル受信レベル)応答イベント (cbToCoNet_vNwkEvent)

```
case E_EVENT_TOCONET_ENERGY_SCAN_COMPLETE:
    _C {
        uint8 *pu8Result = (uint8*)u32arg;
    }
```

ここで、pu8Result[0] が計測されたチャネル数。
pu8Result[1] … pu8Result[pu8Result[0]] に値が入ります。

ch11,13,15 を探索した場合は、
pu8Result[1] は ch11 のレベル。
pu8Result[2] は ch13 のレベル。
pu8Result[3] は ch15 のレベル。

レベルは 0..255 の値を取り、0 が最弱、255が最強となる。**値が高いほどノイズの多いチャネル**となります。

MONO WIRELESS

ユーティリティ

EEPROM

- TWE-Lite で EEPROM を使用する API です。(Regular/Strong) では使用できません。
- インクルード
  #include “eeprom_6x.h”
- マクロ
  - EEPROM_6X_USER_SIZE　利用可能な最大バイト数 (64x60 バイト, 残領域はシステム予約です)
- API
  - bool_t **EEP_6x_bRead(uint16 u16StartAddr, uint16 u16Bytes, uint8 *pu8Buffer);**
    EEPROM の読み出しを行います。
    - u16StartAddr　読み出し開始アドレス。(0… EEPROM_6X_USER_SIZE-1)
    - u16Bytes　読み出しバイト数
    - pu8Buffer　読み出し先のバッファアドレス
  - bool_t **EEP_6x_bWrite(uint16 u16StartAddr, uint16 u16Bytes, uint8 *pu8Buffer);**
    EEPROM の書き込みを行います。書き込みが終了するまでブロックするブロッキング関数ですのでウェイトなどを挟む必要が有りません。
    - u16StartAddr　書き込みの開始アドレス。(0… EEPROM_6X_USER_SIZE-1)
    - u16Bytes　書き込みバイト数
    - pu8Buffer　書き込みのバッファアドレス

btnMgr (ボタン処理)

- IOポートをポーリング監視し、変化を検出します。
  - デフォルトの初期化では、検出は 200Hz, TIMER2を使用します。
  - 五回以上連続で同値なら、その値を採用します。
- API
  - void vBTM_Init(uint32 bmPortMask, uint8 u8TimerDevice)

    簡易版初期化関数。
    - bmPortMask 指定されたポートを監視対象とします。(DIO4 と 10 なら (1UL << 4) | (1UL << 16)) を指定)
    - u8TimerDevice に示すタイマデバイスを初期化し監視します。(E_AHI_TIMER_0 … 4)
  - PR_BTM_HANDLER prBTM_InitExternal(tsBTM_Config *psConf)

    詳細版初期化関数。
    - psConf の値を基に初期化を行います。btnMgr 自身がタイマーを初期化せず外部の周期タイマー割り込み時に呼び出すことを前提とします。戻り値が外部ハンドラーへの関数ポインタ。(psConf->u8DeviceTimer == 0xFF)時。
    - tsBTM_Config
      - bmPortMask　　指定されたポートを監視対象とします。
      - u16Tick_ms　　監視ポーリング周期
      - u8MaxHistory　　連続参照数
      - u8DeviceTimer　　0xFF を与えます
  - typedef void (*PR_BTM_HANDLER)(uint16 u16ms);

    本関数は、別途動作している周期タイマの割り込みハンドラまたは遅延実行部から呼び出す。
    - 定期処理呼び出しのため関数プロトタイプ。(psConf->u8DeviceTimer == 0xFF) 時のみ利用する。
      - u16ms は、前回の呼び出しからの経過ms

    [例]

    ```
    PR_BTM_HANDLER  pBtmHandler = prBTM_InitExternal(psConfig);
    case E_AHI_DEVICE_TICK_TIMER: // 4ms 毎に呼び出される場合
                    (*pBtmHandler)(4);
    ```

btnMgr (ボタン処理)

- API
  - void vBTM_Enable(), vBTM_Disable()
    - 処理を開始、停止する。
  - bool_t bBTM_GetState(uint32 *pbmPorts, uint32 *pbmChanged)
    - ポート状態を読み出します。
    - pbmPorts: ポートの状態 ビットマップ
    - pbmChanged: 変化したポートのビットマップ。本読み出し後ビットマップはクリアされます。読み出し頻度が頻繁でない場合は、変化ありのポートが前回読んだポートの値と同値である場合も発生します（ポートが Lo > Hi > Lo と変化し、その間読み出しが行われなかった場合）。
    - 戻り値：読み取りが有効であれば TRUE。無効で有ればFALSE。

serialInputMgr

- バイナリ形式のシリアル電文の処理を行う。また+ + + 入力によるデバッグ用のモードのトグル処理を行う。
- 電文形式
  - A5 5A [LEN] [PAYLOAD] [XOR] となります。
  - [LEN] が 0x01 ～ 0x7F のとき: 1 ～ 127 バイトのペイロード（一バイト）
  - [LEN] の MSB が1 の場合、二バイト表現となり、一バイト目を 0xXX 二バイト目を 0xYY とすると (0xXXYY & 0x7FFF) がペイロード長となります。
  - [XOR] は [PAYLOAD] の全 XOR を計算します。
- tsSerCmd
  - 本構造体は、予めOで初期化をしておき、au8data, u16maxlen を設定してから使用します。
  - u16len → 読み込んだペイロードの長さ
  - u8xor → 計算されたXOR値
  - au8data → 入力バッファ(別途確保しておいたバッファのアドレス)
  - u16maxlen → au8data の最大長
- uint8 u8ParseSerCmd(tsSerCmd *pCmd, uint8 u8byte)
  - pCmd → 管理構造体
  - u8byte → 入力文字（一文字ずつ読み込ませる）
  - 戻り値 → 処理後の状態 (0x80 以上が特別な意味を持ち E_SERCMD_COMPLETE:完了, E_SERCMD_ERROR:エラー, E_SERCMD_CRCERROR:XORエラー, E_SERCMD_VERBOSE: + + + 系列が入力、となる)
- uint8 bSerCmd_VerboseMode
  - 約200ms以上、1000ms 未満の間隔を空けて + を3回入力すると、この変数がトグルします。
  - デバッグ時や、人間の手で操作するような場合に利用します。

utils

- マクロ：ENCODE_VOLT, DECODE_VOLT
  - 2000～3600 の値を 8bit 値に変換します。2000～2800 の値は 5 刻み、2800～は10 刻みで 8bit 値に割り当てます。

```
uint16 u16Volt = 2860;
uint8 u8Volt_enc = ENCODE_VOLT(u16Volt);
uint16 u16Volt_dec = DECODE_VOLT(u8Volt_Enc);
```

- マクロ：G_OCTET, G_LE_WORD, G_LE_DWORD, G_BE_WORD, G_BE_DWORD
  - ローカル変数 uint8 *p に指し示すアドレスから、1バイト(OCTET), 2バイト (LE_WORD, BE_WORD), 4バイト (LE_DWORD, BE_DWORD) 読み出し p を読み出したバイト分インクリメントします。LE, BE は Little Endian, Big Endian。

```
uint8 *p = &au8data[3]; // 4バイト目から
uint8 u8cmd = G_OCTET(); // u8cmd に audata[3]
uint16 u16len = G_BE_WORD(); // u16len に (audata[4] << 8) | audata[5] を格納
```

- マクロ：S_OCTET, S_LE_WORD, S_LE_DWORD, S_BE_WORD, S_BE_DWORD
  - ローカル変数 uint8 *q に指し示すアドレスに、1～4バイトの値を書き出し、その分だけ q をインクリメンします。

```
uint8 *q = &au8data[0]; // 1バイト目から書き出し
uint8 u8cmd;
uint16 u16len;
S_OCTEC(u8cmd); // au8data[0] に u8cmd を格納
BE_WORD(u16len); // au16data[1] に (u16len >> 8), audata[2] に(u16len&0xff)を格納
```

utils

- DIO関連マクロ
  - `#define vPortAsInput(c) vAHI_DioSetDirection(1UL << (c), 0)` → ポートcを入力に設定する
  - `#define vPortAsOutput(c) vAHI_DioSetDirection(0, 1UL << (c))` → ポートcを出力に設定する
  - `#define vPortSetHi(c) vAHI_DioSetOutput(1UL << (c), 0)` → ポートcをHi状態にする
  - `#define vPortSetLo(c) vAHI_DioSetOutput(0, 1UL << (c))` → ポートcをLo状態にする
  - `#define vPortSet_TrueAsLo(c, s) vAHI_DioSetOutput((s) ? 0 : 1UL << (c), s ? 1UL << (c) : 0)`
    → ポート c を s が TRUE なら Lo, FALSE なら Hi に設定する
  - `#define bPortRead(c) ((u32AHI_DioReadInput() & (1UL<<(c))) ? FALSE : TRUE)`
    → ポート c を読み出す。Loレベルなら TRUE が返る
  - `#define u32PortReadBitmap() (u32AHI_DioReadInput())`
    → ビットマップで読みだす(※ビットマップの1がHi,0がLoとなります)
  - `#define bPortCheckBitmap(bitmap, c) (bitmap & (1UL<<(c))) ? FALSE : TRUE)`
    → 読みだしたビットマップのポート c に対応するビットがLoレベルならTRUEを返す
  - `#define vPortDisablePullup(c) vAHI_DioSetPullup(0x0, 1UL << (c))`
    → ポート c のプルアップを停止する
  - `PORT_KIT_SW*` → 評価キット上のボタンの割り当て
  - `PORT_KIT_LED*` → 評価キット上のLEDの割り当て
- その他
  - `#define _C if(1)` → switch でスコープを定義したい場合、ソースフォーマッタとの相性が悪いので、_C { … } と記述している個所がある。
  - `void vWait(uint32 c)` → カウント c 分だけ空ループを回し時間待ちを行う。
- **vAnalogueConfig(), vAnalogueDisable()**
  → ADCを利用可能設定を行い利用可能になるまでポーリングする、停止する。
  ※ 本APIは設定等に柔軟性が無いため、利用は推奨しない。

serial

- UART 処理関数で入出力FIFOバッファーを管理する。
- void **SERIAL_vInit**(tsSerialPortSetup *psSetup)
  - 初期化関数。サンプルの vSerialInit() を参照のこと。
  - Tx/Rx のバッファのサイズを適切に設定する。
  - ハードフロー、パリティの設定を変えたい場合は、後述の **tsUartOpt** を使用する
  - ボーレート(u32BaudRate) は 9600～230400 を指定する。
    - または MSB を設定し 0x80XX00YY なら、XXがCBP、YYがディバイザの設定となる。
- void **SERIAL_vInitEx**(tsSerialPortSetup *psSetup, tsUartOpt *psUartOpt)
  - SERIAL_vInit() の拡張版。tsUartOpt *psUartOpt が追加され初期化関数 (パリティなどの変更)、NULL なら パリティ none, StopBit 1、ハードフロー無しとなる。
    - bHwFlowEnabled: TRUE:ハードウェアフロー有効 FALSE:無効
    - bParityEnabled: TRUE: パリティ有効 FALSE:無効
    - u8ParityType:
      E_AHI_UART_EVEN_PARITY:偶数
      E_AHI_UART_ODD_PARITY:奇数
      (パリティ無効時は値の設定不要)
    - u8StopBit: E_AHI_UART_1_STOP_BIT: 1bit, E_AHI_UART_2_STOP_BIT: 1.5/2bit
    - u8WordLen: 7: 7bit, 8: 8bit
- bool_t **SERIAL_bRxQueueEmpty**(uint8 u8SerialPort)
  - FIFOキューに文字列が入力されているか調べる。
- int16 **SERIAL_i16RxChar**(uint8 u8SerialPort)
  - FIFOキューから1文字取り出す。
- void **SERIAL_vFlush**(uint8 u8Uart);
  - u8Uart ポートに対応する出力バッファをフラッシュする。終了するまでブロッキングされる。

fprintf

- void vfPrintf(tsFILE *psStream, const char *pcFormat, ...)
- void vPutChar(tsFILE *psStream, const char c) → 1バイト出力

- **対応フォーマット**
  - マイナス符号, 0のパディング, 桁数
  - s, d, x, X, u, c,
  - b (二進数出力)
- tsFILE は UART または、SPRINTF にて定義。
  - tsFILE
    - bool_t (*bPutChar)(uint8 u8Device, uint8 u8Char);
      - 1バイト出力関数
    - uint8 u8Device;
      - 出力先 (UART 出力時に有効)
  - UART の場合　→ SERIAL_bTxChar を bPutChar に指定
  - SPRINTF の場合 → SPRINTF_Stream 定義済み

sprintf

- vfPrintf の tsFILE* を利用した sprintf()
  - 一般の sprintf とのインタフェース互換を持つものではなく、あらかじめ確保されたバッファに対して文字列をフォーマット出力する。
  - 出力バッファは NUL 終端されている。バッファサイズの末尾は NUL となり、それ以降は出力されない。
- extern tsFILE *SPRINTF_Stream;
  - vfPrintf へ渡す tsFILE * 変数。vfPrintf(SPRINTF_Stream, “…”, …) と記述する。
- void SPRINTF_vInitXXX()
  - 初期化と内部バッファの確保を行う。XXXはバッファのサイズで 32 64 128 256 512 1024 バイトから選択する。ただし複数のバッファは管理できないので、いずれか一つの初期化関数を1回だけ呼び出す。
- void SPRINTF_vRewind();
  - 出力先をバッファの先頭に巻き戻す。
- uint8* SPRINTF_pu8GetBuff();
  - バッファのアドレスを得る。
- uint16 SPRINTF_u16Length();
  - バッファ長を得る。
- uint32 u32string2hex(uint8 *p, uint8 u8len)
  - 先頭アドレス p、長さ u8len の文字列を16進数数値32bitに変換する
- uint32 u32string2dec(uint8 *p, uint8 u8len)
  - 先頭アドレス p、長さ u8len の文字列を10進数数値32bitに変換する

ByteQueue

- uint8 型のFIFOキュー
  - UART 処理で内部的に使用されています。
  - 利用方法は ByteQueue.h, ByteQueue.c を参照。

u8CCITT8

- uint8 u8CCITT8(uint8 *pu8Data,uint8 size);
  CRC8 を計算します。

  - pu8Data: 計算するデータ列のアドレス
  - size: バイト数

randMT

- メルセンヌ・ツイスター法に基づく乱数生成
  - ToCoNet での設定
    sToCoNet_AppContext.u8RandMode = 2 を指定した場合、ColdStart 始動時にハードウェア乱数を生成し、この値を元に乱数の種を生成し、MT(Mersenne Twister) による乱数を生成します。 (高精度 32Khz 水晶発振器を利用する場合は内蔵の乱数生成が動作しません。このとき良い乱数系列が必要な場合に使用します)
    処理時間は初期化に 1ms 弱(22000クロック@32Mhzで計算)し、新たな乱数計算に約 44クロックの時間と、2.5KB のメモリが必要になりますが、ソフトウェア乱数としては非常によい系列を生成します。
    本アルゴリズムを利用するには ToCoNet_USE_MOD_MTRAND を定義してアプリケーションをビルドする必要があります。
- API
  - ToCoNet でsToCoNet_AppContext.u8RandMode = 2を指定した場合は**ToCoNet_u32GetRand()**を使用します。システム中でもこの乱数が使用されます。
  - ToCoNet でsToCoNet_AppContext.u8RandMode = 2を指定しない場合は、void **MTRND_vInitSeed(uint32)** にて乱数の種の生成、uint32 **MTRND_u32GenRnd(void)**にて32bit乱数値を得ます。
- ライセンス
  - 本アルゴリズムの利用した場合、BSD License に基づいたライセンス表記が必要になります。(右記参照)

ライセンス記述:

A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

3. The names of its contributors may not be used to endorse or promote
   products derived from this software without specific prior written
   permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)

XorShift 乱数

- XorShift による乱数
  - ToCoNet での設定
    sToCoNet_AppContext.u8RandMode = 3 を指定した場合、ColdStart 始動時にハードウェア乱数を生成し、この値を元に乱数の種を生成します。
    本アルゴリズムを利用するには ToCoNet_USE_MOD_RAND_MT を定義してアプリケーションをビルドする必要があります。
- API
  - ToCoNet で指定した場合は**ToCoNet_u32GetRand()**を使用します。システム中でもこの乱数が使用されます。**XORSRND_** に始まる API を利用してはいけません。
  - ToCoNet で指定しない場合は、void **XORSRND_vInitSeed (uint32)** にて乱数の種の生成、uint32 **XORSRND_u32GetRand (void)**にて32bit乱数値を得ます。

fixmath ライブラリ

- 固定小数点演算ライブラリ libfixmath を TWE 用にビルドツリーに含めました。詳細は http://code.google.com/p/libfixmath/ を参照ください。
  - fix16_t は符号付き 32bit 整数を 上位15bit を整数部、下位16bitを小数部としています。つまり fix16_t の 0x00000001 は 1/65536 ≒ 0.000015 を表現しています。

  ※ 通常の float も利用可能です。(-lm をリンク時に付加します)
- 本SDKからの利用には #include “fixmath/fixmath.h” を追加してください。
- FIXMATH_NO_CACHE を定義してビルドしています。

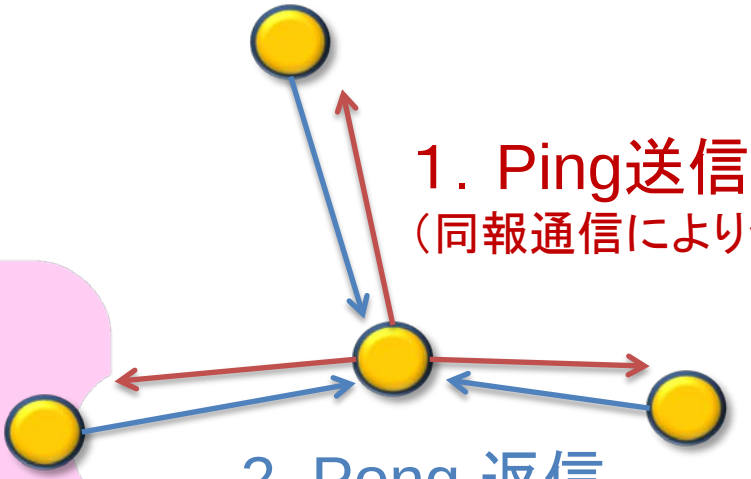
```
#include “fixmath/fixmath.h”

fix16_t complex_karctan2( fix16_t y , fix16_t x)
{
    fix16_t cf1 = 51471; // = 0.78539  = pi / 4
    fix16_t cf2 = 154415; // = 2.35619 = 3pi / 4

    fix16_t absy = y;
    fix16_t angle;
    if ( absy < 0) absy = -absy;
    if ( x>=0) {
        /* r = (x-absy) / ( x + absy) */
        fix16_t r = fix16_sdiv(fix16_sadd(x,-absy),
                fix16_sadd(x,absy));
        /* angle = .1963*r^3 - .9817*r + cf1 */
        angle = fix16_sadd( fix16_sadd(
                    fix16_mul(12864,fix16_mul(r,fix16_mul(r,r))),
                    fix16_mul(-64336,r)),
                cf1);
    }
    else {
        /* r = (x + absy) / (absy - x) */
        fix16_t r = fix16_sdiv(fix16_sadd(x,absy),
                fix16_sadd(absy,-x));
        /* angle = .1964*r^3 - .9817*r + cf2 */
        angle = fix16_sadd( fix16_sadd(
                    fix16_mul(12864,fix16_mul(r,fix16_mul(r,r))),
                    fix16_mul(-64336,r)),
                cf2);
    }
    if( y < 0) return (-angle);
    return angle;
}

…
  vfPrintf(&sSerStream, "START complex_karctan2¥n");
  fix16_t P = fix16_mul(fix16_from_int(10000),
                complex_karctan2(140351, 65536));
  int k = fix16_to_int(P);
  vfPrintf(&sSerStream, "¥nGot %d¥n", k);
```

MONO WIRELESS

サンプルの解説
Wks_ToCoNet/Samp_PingPong

Samp_PingPong

- Samp_PingPong (ピンポン・サンプル) では、親機子機の関係はありません。ある無線機がピン(Ping)の無線パケットを発信すると、これを受信した他の無線機がポン(Pong)の無線パケットを送り返します。



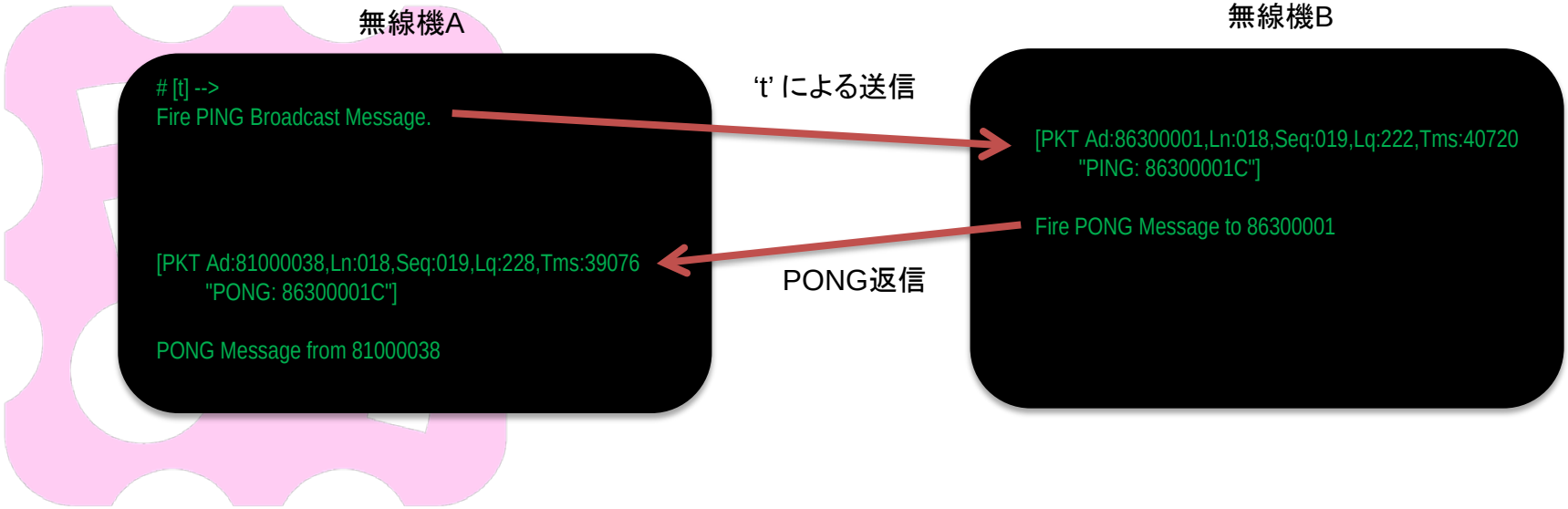
1. Ping送信
（同報通信により全無線機へ送信）

2. Pong 返信
（あて先を指定し、Ping 送信元へ返信）

まず使ってみる

- まず、サンプルを動かしてみます。

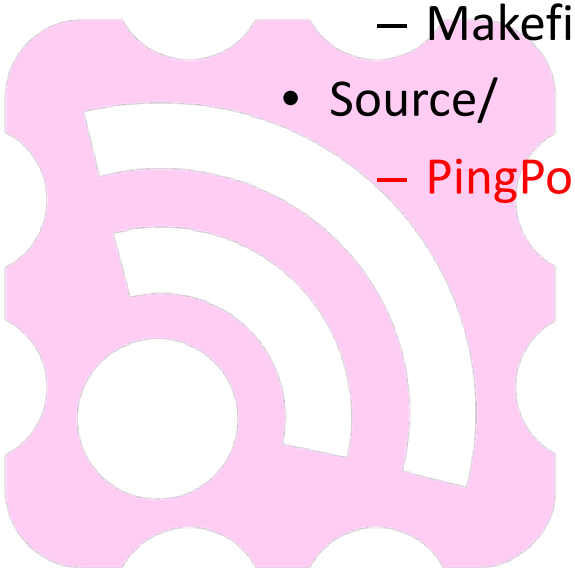
1. ファームウェアを書き込みます。
   C:¥TWESDK¥Wks_ToCoNet¥Samp_PingPong¥PingPong¥Build¥にある
   Samp_PingPong_PingPong_JN5164_X_Y_Z.bin を2台の無線機に書き込みます。
2. 無線機の UART0 ポートを PC などのターミナルソフトで開いておきます。通信条件は
   115200 8N1 (115200bps 8bit パリティ:None ストップビット:1)です。
3. 電源を投入します。メッセージが出ない場合は、ケーブルなどの接続をお確かめの上、1.
   の手順まで遡って確認してください。
4. 片側の無線機から ‘t’ を入力します。



ソースコードの構成について

Samp_PingPong のディレクトリ構成を解説します。このサンプルはごく小規模で、PingPong.c に殆どの処理が記述されています。

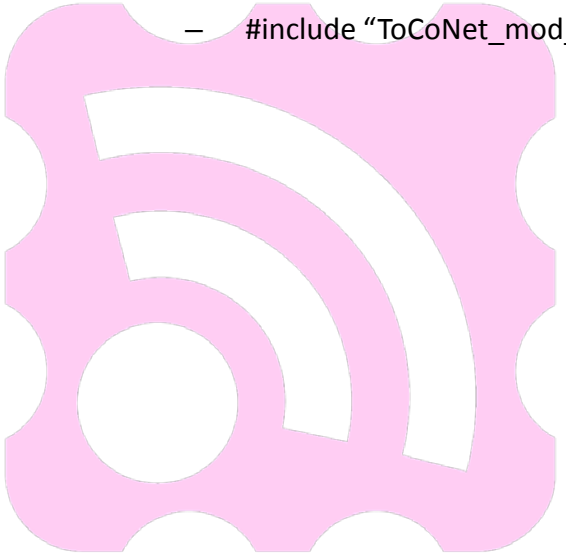
- Wks_ToCoNet/Samp_PingPong
  - Common/　　共通性の高いファイル
    - app_event.h　　イベント定義
    - config.h　　設定
  - PingPong/
    - Build/　　生成バイナリなど
      - Makefile　　ビルド方法の定義
    - Source/
      - PingPong.c　　主要処理を記述



#include

主要な #include を解説します。

- C言語標準ライブラリ
  - #include <string.h>　C言語標準ライブラリ (memcpy など)
- マイコン用ヘッダ
  - #include <jendefs.h>　基本的な定義 (uint8 などの型定義など)
  - #include <AppHardwareApi.h>　ペリフェラルAPI (AHI 関数群)
- ToCoNet Utils ヘッダ
  - #include “utils.h”　ペリフェラル向けライブラリ
  - #include “serial.h“　シリアルポート
  - #include “fprintf.h“　vfPrintf() など
  - #include "sprintf.h“　SPRINTF_*
- ToCoNet ヘッダ
  - #include “ToCoNet.h“　ToCoNet 定義
  - #include “ToCoNet_mod_prototype.h“　ToCoNet モジュール定義



cbAppColdStart()

始動時に最初に呼び出されるいわば main() 関数に相当する関数です。各種初期化を行います。

- ToCoNet 関連の初期設定を行っています
  - sToCoNet_AppContext.u32AppId = APP_ID; // アプリケーションID
  - sToCoNet_AppContext.u8Channel = CHANNEL; // チャネル (11-25,26)
  - sToCoNet_AppContext.bRxOnIdle = TRUE; // 受信回路を有効に！
- SPRINTF_vInit128() // SPRINTF を初期化します
- ToCoNet_Event_Register_State_Machine(vProcessEvCore); // vProcessEvCore() 関数を登録します。
  本関数がアプリケーションでは初期メッセージを出力するのみです。
- vInitHardware(FALSE); // ハードウェアの初期化を行います。
  本アプリケーションで用いるハードウェアは UART (Serial) と、汎用IO のみです。
  ※ UART の出力はここでは行わず、vProcessEvCore() で行ってください。
- ToCoNet_vMacStart(); // 無線部を明示的に開始します。

vInitHardware()

ハードウェアの初期化をまとめています。

- vSerialInit()　　// UART0 の初期化を行います。
  - Tx/Rx の FIFO バッファーの定義
  - SERIAL_vInitEx()　　// UART の初期化
  - sSerStream の設定　　// vfPrintf() で出力先とするための構造体
- ToCoNet_vDebug*()　　// TWE-NETのスタックデバッグ用のコードです。通常使用しません。
- vPort*()　　// IO の入出力と、出力状態の設定。(DIO11 なら 11 を指定)
  ※ 出力設定時は、先に IO の HI/Lo を指定してから、vPortAsOutput() を呼びます。

cbToCoNet_vMain()

全ての割り込みを起点として、もっとも頻繁に呼びされる関数です。TWE-NETでは、CPU DOZE (休止) 状態を有効にするため、処理が終了すると DOZE 状態に入り割り込み待ちとなります。これが解除されたときに呼び出されるのが本関数です。

本関数内で、’t’ の入力処理とパケット送信処理を行います。

- vHandleSerialInput()　　// シリアルポートの入力処理を行う
  - SERIAL_bRxQueueEmpty()　　// シリアルポートに何か入力されると TRUE になる
  - SERIAL_i16RxChar()　　// シリアルポートのFIFOキューから1バイト値を取り出す
    0x00 – 0xFF 以外の値が戻る場合はエラーで無視します。
  - 入力したバイトに対応して処理を実行します。
    - ‘<‘, ‘>’　→ チャネルの変更
    - ‘p’　→ 出力の変更 (3 が標準、2,1,0 と順に弱くなる)
    - ‘t’　→ Pingパケットを送信する

Pingパケット送信処理

Ping パケットはブロードキャスト（同報通信）で、無線チャネルとアプリケーションIDが同じであれば、全ての無線機が受信できます。

- tsTxDataApp tsTx; // パケット送信のための構造体
- memset(&tsTx, 0, sizeof(tsTxDataApp)); // 必ず0クリアします！

- tsTx.u32SrcAddr = ToCoNet_u32GetSerial(); // 自身のアドレスを32bit形式で設定
- tsTx.u32DstAddr = 0xFFFF; // ブロードキャスト

- tsTx.u8Retry = 0x82; // 強制再送は2回
- tsTx.u8CbId // cbToCoNet_vTxEvent() で照合するためのID
- tsTx.u8Seq // パケットのシーケンス番号 (そのまま伝達される)
- tsTx.u8Cmd // パケットの種別番号 (そのまま伝達される)

- SPRINTF_vRewind(); // SPRINTF を用いてペイロードを準備
- vfPrintf(SPRINTF_Stream, …) // SPRINTF には vfPrintf() を用いる
- memcpy(tsTx.auData, SPRINTF_pu8GetBuff(), SPRINTF_u16Length()); // SPRINTFバッファのコピー

- tsTx.u8Len = SPRINTF_u16Length(); // ペイロードサイズの決定

- ToCoNet_bMacTxReq(&tsTx); // 送信する

cbToCoNet_vRxEvent()

パケット受信後、cbToCoNet_vRxEvent() が呼び出されます。パラメータに tsRxDataApp 構造体へのポインタが渡されます。このポインタは、関数内でのみ有効です。

- pRx->u8Seq // 送信時に設定したシーケンス番号。この番号は重複パケットの判定に利用される
  同報通信を利用する場合、通信成功率を上げるため送受信成功失敗に関わらず複数回送信するのが一般的です。複数回受信が成功する事になり、重複パケットの処理が必要になります。
- pRx->auData // 送信されてきたデータ。ここでは、PINGパケットを受信した場合は PING を PONG に書き換えてそのまま返信します。PONG パケットだった場合は、UART0 にPONGを受信したメッセージを出力します。
- pRx->u8Lqi // 受信感度。0～255の値をとり、255が最強となります。値と物理量には厳格な関連はありませんが、おおよそ電力に相当します。(本サンプルでは利用されていません)
- 本関数からパケットを送り主アドレスを特定して返信します。PING の送信処理との相違点を解説します。
  - tsTx.u32DstAddr → pRx->u32SrcAddr を明示します。これで、送信元を特定した送信になります。
  - tsTx.bAckReq → Ack付き送信を行います。Ack付きの利点は相手に届いたかどうか送信元が把握でき、また多くの場合効率的に振る舞います。
- sAppData.u32LedCt // PING 受信したときに LED を点灯するためのカウンタです。cbToCoNet_u32HwEvent() で処理されます。

cbToCoNet_vHwEvent()

ハードウェアの割り込みなどのイベントです。割り込みハンドラが終了した後に呼び出されます。

- sAppData.u32LedCt　　// PING 受信したときに LED を点灯するためのカウンタです。cbToCoNet_vHwEvent() で処理されます。
  - cbToCoNet_vHwEvent() では、u32DeviceID == E_AHI_DEVICE_TICK_TIMER (4ms 毎に呼び出される TWE-NET が内部的に利用している TickTimer) が呼び出されるたびに、カウントを更新し LED の点灯・消灯制御を行います。
    ※ なお、点滅等時間制度が必要な動作を行う場合は割り込みハンドラ cbToCoNet_u8HwInt() を用いてください。cbToCoNet_vHwEvent() は、アプリケーション処理のために遅延実行されるため遅延が大きく不定です。

MONO WIRELESS

変更点

2014/4月 ⇒ 2014/6月 (1)

- SDK
  - 変更点
    - Eclispe を Kepler SR2 ベースの pleiades-e4.3-cpp-32bit-jre_20140321 に変更しました。
- ToCoNet (1.0.4)
  - 変更点
    - チャネルアジリティ利用時の PANIC が発生する問題が修正されました。
    - TWE-Lite で ch26 が使用できない場合があった問題が修正されました。
    - TWE-Regular/Strong のベースSDKのバージョンが 1.6 になりました。
    - sprintf () などのライブラリ関数をコンパイルできるようになりました (非公式対応)
    - チャネルアジリティ時にアプリケーション再送なしかつAckなし送信を指定すると、1チャネルしか送付されない問題を修正しました。
    - Ack なし送信指定で、0x80 より小さい再送回数を設定したときは強制的に 0x80 のビットを立てるようにしました。
    - TWE-NET内の重複チェッカ(TOCONET_DUPCHK)のチェック方式を 32bit ビットマップから 64bit に拡張し、より確実性を高めました。
    - LayerTree で通常通信の失敗をカウントし、失敗蓄積により上位ノードを再探索するようにしました。

2014/4月 ⇒ 2014/6月 (2)

- App_Twelite (1.6.4)
  - PWM出力を L/H 反転できるようになりました。(opt:00010000)
  - PWM出力を起床時にLoに設定できるようになりました。(opt:00020000)
  - 定期通信をキャンセルできるようになりました。(opt:00000002)
  - 定期送信を受信してもUARTに出力させない設定を追加しました。(opt:00000004)
- App_Uart (1.2.8)
  - 各種機能を追加しました (http://tocos-wireless.com/jp/products/TWE-ZERO/App_Uart/)
- App_IO (1.1.0)
  - リモコン長押しモードにて起床・スリープ前後の無反応期間を短縮しました。
  - シリアルポートによる子機から親機への送信コマンドを追加しました。
- Samp_I2C (0.2.0)
  - BH1715/SHT21/EEPROM の読み書きテストが可能です。
- Samp_Monitor (1.3.0)
  - TWE-EH-S 用にメンテナンスを行いました (http://tocos-wireless.com/jp/products/TWE-EH-S/sw_html/index.html)

2013/9月 ⇒ 2014/4月 (1)

- ToCoNet (1.0.0)
  - 変更点
    - ベースのライブラリを v.976 に変更しました
    - Ack 付き送信時の再送回数を指定できるようになりました
    - 回復不能なエラーを伝達する PANIC が実装されました
    - メッセージプール取得時の LQI を報告するようにしました
    - チャネルアジリティのチャネル切り替え維持の排他制御を強化しました
    - メッセージプールの送信タイミングを最適化しました
    - LayerTree_mininodes の起動時、スリープ復帰時の消費電流を最適化しました (Samp_Monitor のサンプルをこれにあわせて更新しました)
    - ToCoNet_Tx_vProcessQueue() API を追加しました。LayerTree_mininode などで、送信処理要求を実施した後、実際の送信処理が TICK_TIMER 処理までの遅延を待たずに MAC 層へ要求を伝達します。
    - ToCoNet_MsgPl_bRequest_w_Payload_to_Parenet() API を追加しました。子機からのデータ転送とメッセージプールの要求を同時に行います。
    - メッセージプールのスロットを複数同時要求できるようにしました。
    - E_EVENT_TOCONET_NWK_ROUTE_PKT イベント処理時に .bAbort = TRUE とすると中継を中断します。
    - LayerTree ネットワークの制御パラメータをAckつき送信の場合、再送１回のみとしました。
    - LayerTree ネットワークの初期化パラメータを追加しました。スリープ復帰時などに、送信先の確認をせずに通信を行う処理が可能になりました。
    - ToCoNet_u16RcCalib() API を追加しました。RCタイマーのキャリブレーションを明示的に行います。
    - 今後、海外対応のためにモジュール仕様が変更されます。この仕様変更に対応します。本SDKで作成したファームウェアは、仕様変更前後に対応します。
      ※ 2013/9月版で作成したファームウェアは、新仕様モジュールでは動作しません。
  - 判明している問題
    - チャネルアジリティ利用時に通信負荷がほぼ100%が継続すると、PANIC による送信不能状態が発生することがあります。

2013/9月 ⇒ 2014/4月 (2)

- ToCoNetUtils (1.0.0)
  - vTimerChangeDutySync() API を追加しました。PWM の DUTY を変更する再に、PWM 周期の末尾までポーリング待ちをしてからレジスタに値を書き込みます。App_Audio での PWM 値の設定に利用しています。
- Makefile
  - Version.h を生成しなくなりました。変わりにコンパイルオプションとして –DVERSION_MAIN=1 といったパラメータを渡すようになりました。
- App_Audio (1.0.1)
  - 新規追加されました。
- App_Twelite (1.6.1)
  - 実験用のコードが追加されました。詳しくは Experimental.h を参照してください。
- App_Uart (1.0.8)
  - 透過モードのパフォーマンスが向上しました。その他、コードメンテナンス。
- Samp_ContTx (0.1.0)
  - データを可能な限り連続して送信するサンプルです。
- Samp_Monitor (1.0.1)
  - ソースコード構成を一新、EndDevice_Input は低消費電力化を実施しました。
- Samp_Wayback (1.0.1)
  - 新規追加しました。LayerTree ネットワーク利用のメッセージプールを利用する省電力子機のサンプルです。

2013/9月 ⇒ 11月 (1)

- ToCoNet (0.9.14)
  - メッセージプールの複数スロット利用が可能になりました。
  - メッセージプールでスロットに対して「データ無し」として、親機からデータ展開できるようになりました。
  - 0x8000FFFF のネットワーク経由ブロードキャストの動作不具合を修正しました (2回層目以降へ中継しなかった)
  - 下り方向の宛先指定メッセージの受信時動作不具合を修正しました (中継機からのパケットを受信表示する・送信元アドレスが 0x8000000 でない)
  - LayerTree ネットワークで上位ノードが喪失した時の振る舞いを修正しました (モジュールリセットが発生していた)
  - LayerTree ネットワーク設定構造体に以下のオプションを追加しました。
    - u8MaxSublayers, u8LayerOptions, u8minLQI
  - LayerTree ネットワークでの下り方向のブロードキャストを子機でも受信できるようにしました
  - LayerTree ネットワークでの送信時構造体の以下の送信遅延・再送間隔設定を反映されるようにした。(ただし中継パケットは固定値タイミングで送信される)
    - u16delay_min, u16delay_max, u16retry_dur
  - スリープの周期実行時に正しく次の周期が計算されないことが有った点を修正した
- ToCoNetUtils
  - uart.c, serial.c: 7bit通信を指定できるようにした

2013/9月 ⇒ 11月 (2)

- App_Twelite (1.3.10)
  - 無線出力の設定 (インタラクティブモードで x)
  - ACM1620/AQM0802A I2C LCD モジュール対応の実験的実装
  - TWE-Regular/Strong で動作するように修正
  - TWE-Lite 評価開発キット向けピン配置を行った
  - SET_DO_ON_SLEEP の実験的実装（ソースコード参照）
  - USE_BROWN_OUT_DETECT の実験的実装（ソースコード参照）
- App_Uart (1.0.6)
  - 各モード(書式、透過など)間の混信が有ったのを修正
  - チャットモード実装
  - UART の設定を 8N1 や 7E2 といった形式で設定できるようにした
  - Regular/Strong 500/667kbps 対応
  - 暗号モード時に平文パケットを表示しないようにした
  - LayerTree ネットワーク利用の実験的な実装 (再コンパイルが必要)
- App_Io (0.2.3)
  - 新規追加

2013/9月 ⇒ 11月 (2)

- Samp_Monitor (0.7.4)
  - スリープ前に UART Disable のコードを含めた（動作確認、省略しても良い）
  - TWX-0003/0009 の始動時のコードを追加
  - EndDevice の再送回数が1回のみになっていたのを訂正
  - Remote に LCD 対処用のコードを追加
  - Remote に メッセージプールテスト用のコードを追加 (MSGPL_SLOT_TEST)
  - Parent に LED 点灯用のコードを追加
  - Parent に メッセージプールの各スロットの設定テストコードを追加
  - Router のパケット送出タイミングをランダム的にずらすコードを追加
  - sTx.u16DelayMax = 300; // 送信開始の遅延を大きめに設定する
- Samp_PingPong (0.1.1)
  - 新規追加

2013/9月 ⇒ 11月 (3)

- Tools/tweprog
  - 新規追加 (Windows専用プログラマ)
- MkFiles
  - ビルド自体は完了するものの後処理のファイル移動スクリプトがエラーになる場合があった点を修正
- Wks_ToCoNet
  - eclipse によるコード解析 (C:¥TWESDK にインストール前提) を含めたワークスペースとした