ZigBeeアプリケーション構築ガイド

スカイリー・ネットワークス

スタックアーキテクチャ

ZigBeeスタックの基本構造

1. 各層は「メッセージボックス」経由でイベントを交換します
   ＝レイヤ間は疎結合

2. SK_Base_Main()関数を周期的に呼び出すことでスタック各層の処理が行われます。

3. ZDOとAPS 層がアプリケーションで使う主な関数を提供します。それぞれ各関数の先頭に “SK_ZDO” または“SK_APS“と接頭辞が付きます。

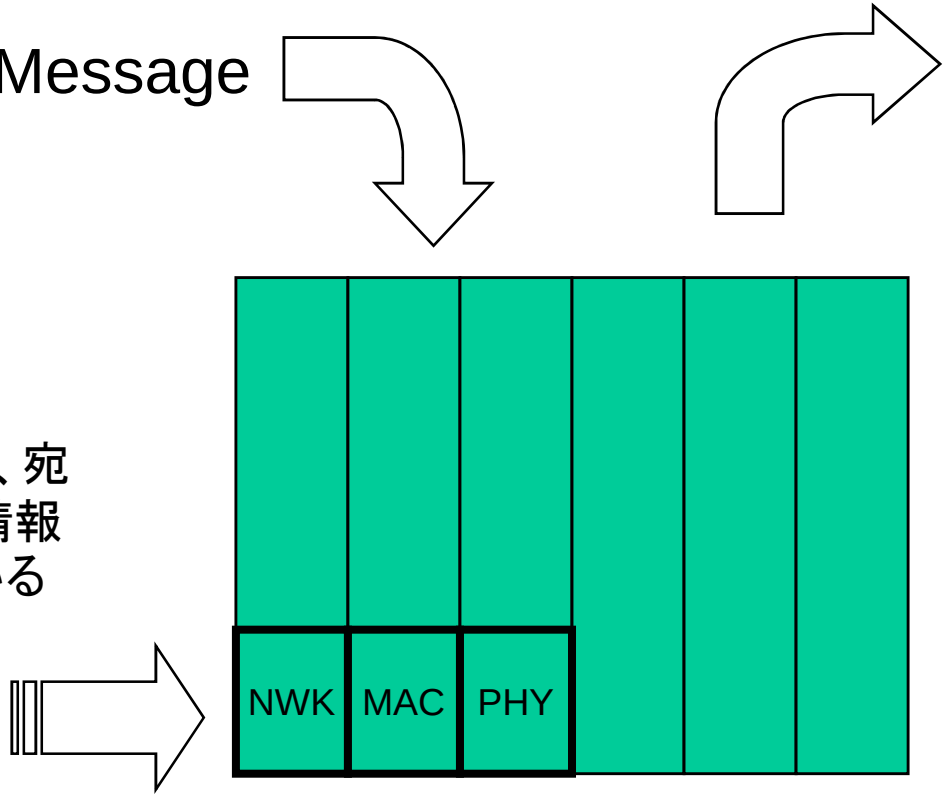
4. アプリケーションに公開されているタイマは 1tick = 10msecです。

5. OSレスですが、擬似的な同期・スリープ処理のためのマクロが提供されています。

メッセージボックス

SK_PostMessage

SK_GetMessage



各エントリは、宛先レイヤの情報を保持している

送受信のためのデータ、ZigBeeスタック内部のイベント、プリミティブなどはすべてMessageBoxに格納される。

Message Box (FIFO)

PostMessage

START_REQUESTコマンドを発行する例：

```
SK_AllocDataMemory((SK_VP *)&tNstrReq);

tNstrReq->m_BeaconOrder = gBeaconOrder;
tNstrReq->m_SuperframeOrder = gSuperframeOrder;
tNstrReq->m_BatteryLifeExtension = 0;

SK_PostMessage(
        SK_LAYER_NWK,
        SK_LAYER_APL,
        SK_NLME_START_ROUTER_REQUEST_CMD,
        (SK_VP)tNstrReq);
```

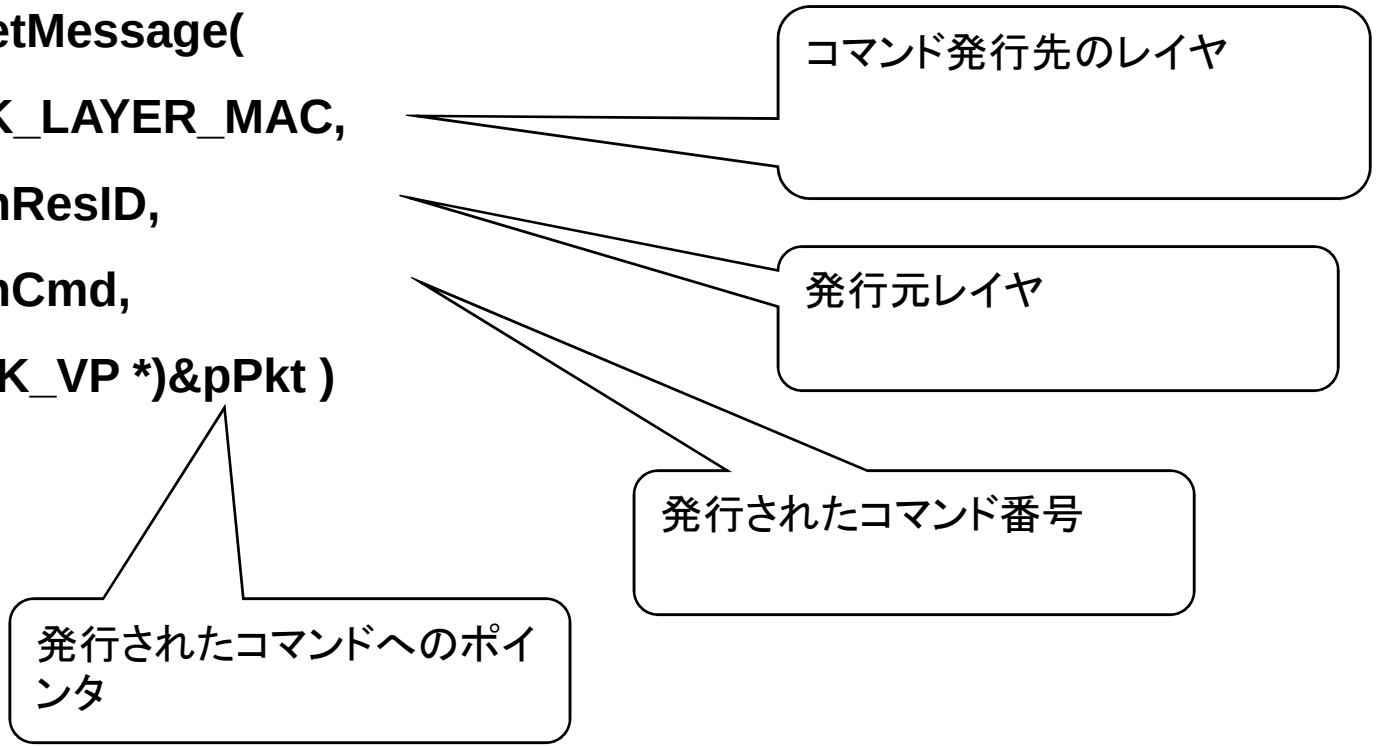
バッファからメモリ領域を確保する

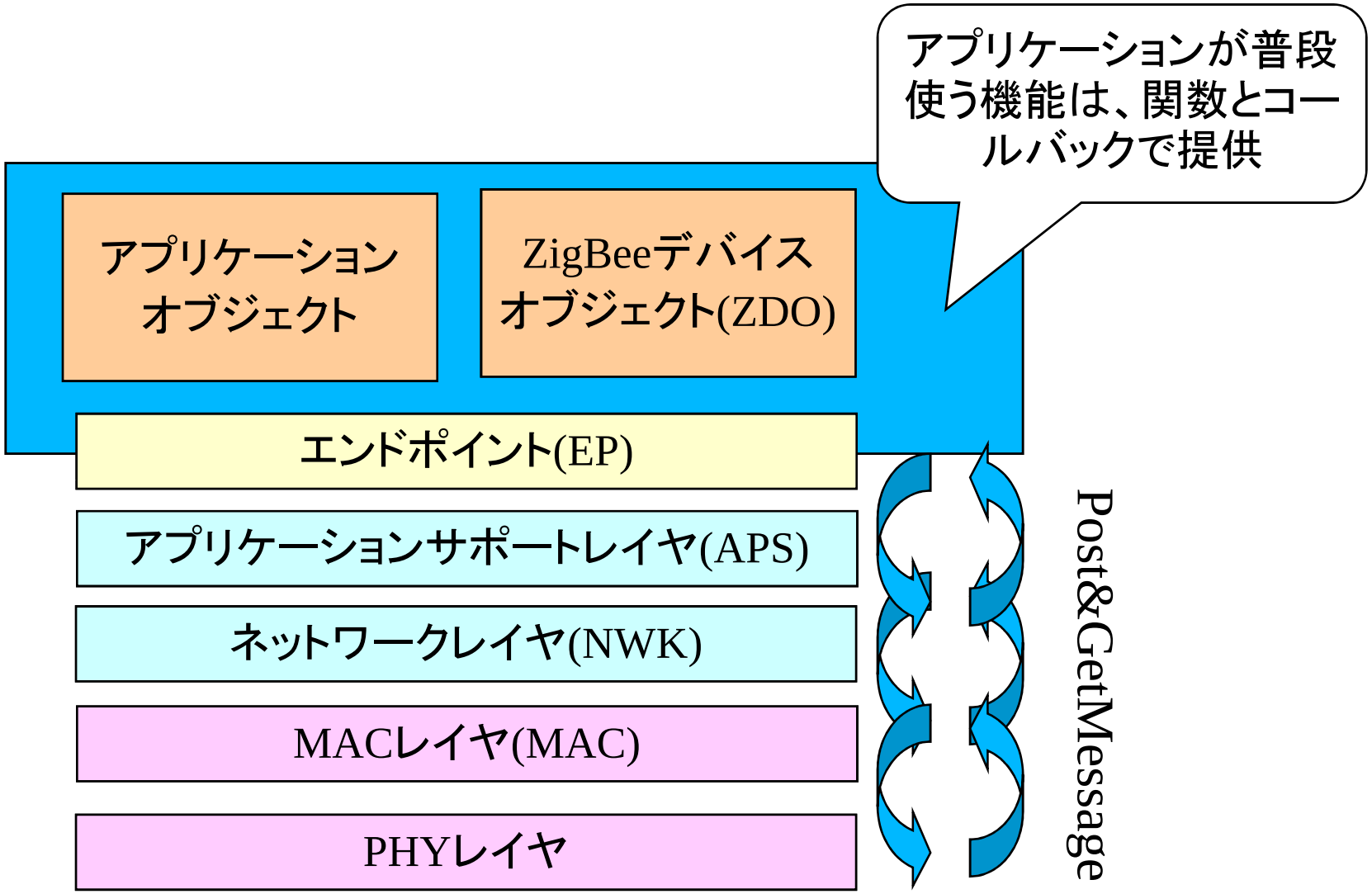
コマンドの引数を設定する

コマンドをPostMessage関数でFIFOに格納する。
宛先レイヤは NWK層, 発行元はAPL層, コマンドは NLME-START-ROUTER.REQUEST

GetMessage

```
SK_GetMessage(
    SK_LAYER_MAC,
    &nResID,
    &nCmd,
    (SK_VP *)&pPkt )
```





スタック内部の主なテーブル

- 10 Discovered Pan Tables, with 25 bytes for each table.

MAC Layer

- 16 Routing Tables, with 12 bytes for each table.
- 10 Neighbor Tables, with 40 bytes for each table.

NWK Layer

- 5 Application Object Tables, with 24 bytes for each table.

APS Layer

- 16 Packet Buffers, with 160 bytes for each buffer.
- 8 Command Buffers, with 16 bytes for each buffer.

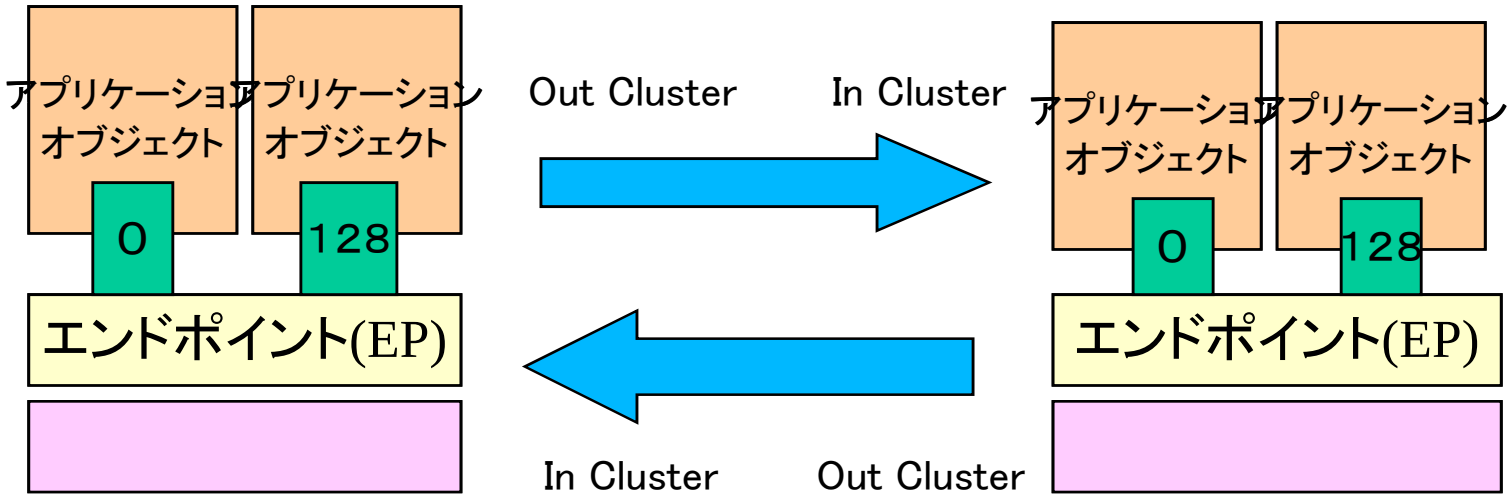
Generic

⇒スタックのリビルドによってサイズの変更が可能

アプリケーションオブジェクトとクラスター

クラスターIDとは

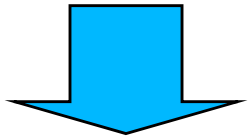
・送受信するデータの「意味」を表す番号です。
・アプリケーションはクラスター番号を見て、その受信したデータが何のためのデータであるかを判断します。



クラスターIDとは

・Endpoint 0番はZDOによって予約されています
・ZDOが利用するクタスター番号の例

0x0000: //NWK_addr_req
0x0001: //IEEE_addr_req
0x0002: //Node_Desc_req
0x0003: //Power_Desc_req
0x0004: //Simple_Desc_req
0x0005: //Active_EP_req
0x0006: //Match_Desc_req
…

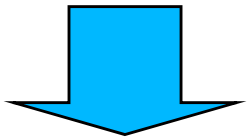


Endpoint 0番に対するクタスター番号0x0001番の送信は、「NWKアドレスを応答せよ」という意味

クラスターIDの連携

・例：SK_ZDO_Simple_Desc_req
・指定したエンドポイントのアプリケーションオブジェクトでサポートされている、Input ClusterとOutput Clusterの一覧を得る

・例：SK_UB SK_ZDO_Match_Desc_req
・指定したクラスターの一覧をサポートするアプリケーションオブジェクトを応答する

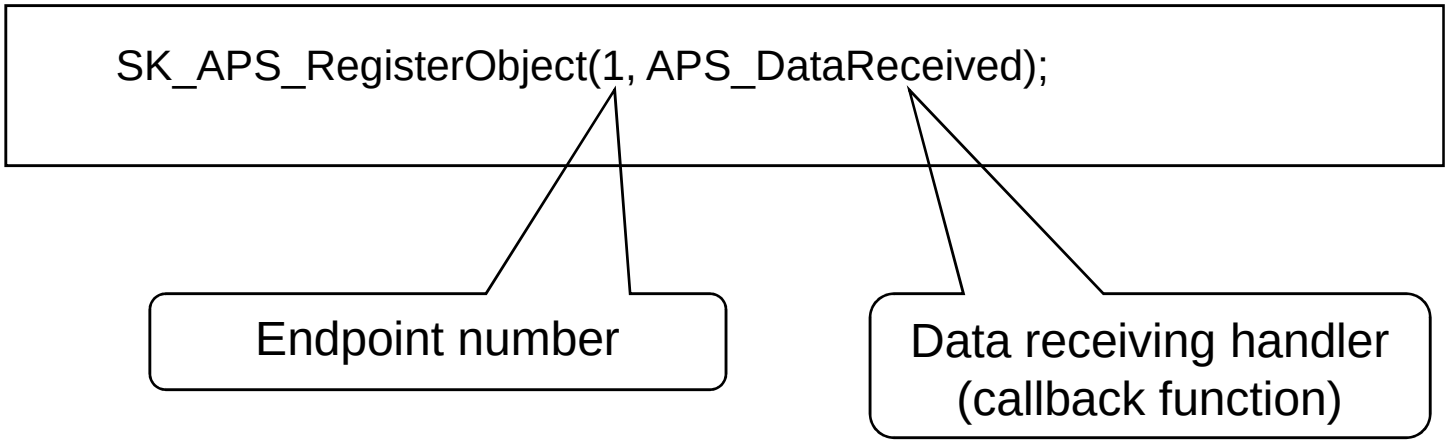


未知のZigBee端末にどんな機能があるかを知ることができる

クラスター番号とデータ構造の対を公開する＝インターフェイスの公開

アプリケーションオブジェクト①

・ZigBeeでは、アプリケーションのことをアプリケーションオブジェクトと呼びます。
・アプリケーションオブジェクトは、動作前にZigBeeスタックに対して「登録」が必要です。
・登録はSK_APS_RegisterObject関数によって行います。



アプリケーションオブジェクト②

- プロファイルとして登録するには、ProfileID, クラスターIDリストを与える必要があります。
- 登録はSK_APS_StandardRegisterObject関数によって行います。

```
SK_APS_APPLICATION_OBJECT app;

app.m_Callback = APS_DataReceived;
app.m_ApplicationID = 0x0000;
app.m_ProfileID = 0x7f01;
app.m_DeviceID = 0x0000;

app.m_InputClusterCount = 0x02;
app.m_InputClusterList[0] = 0x0004;
app.m_InputClusterList[1] = 0x0054;

app.m_OutputClusterCount = 0x02;
app.m_OutputClusterList[0] = 0x0001;
app.m_OutputClusterList[1] = 0x0002;

SK_APS_RegisterStandardObject(1, &app);
```

プロファイルとしてのアプリケーション

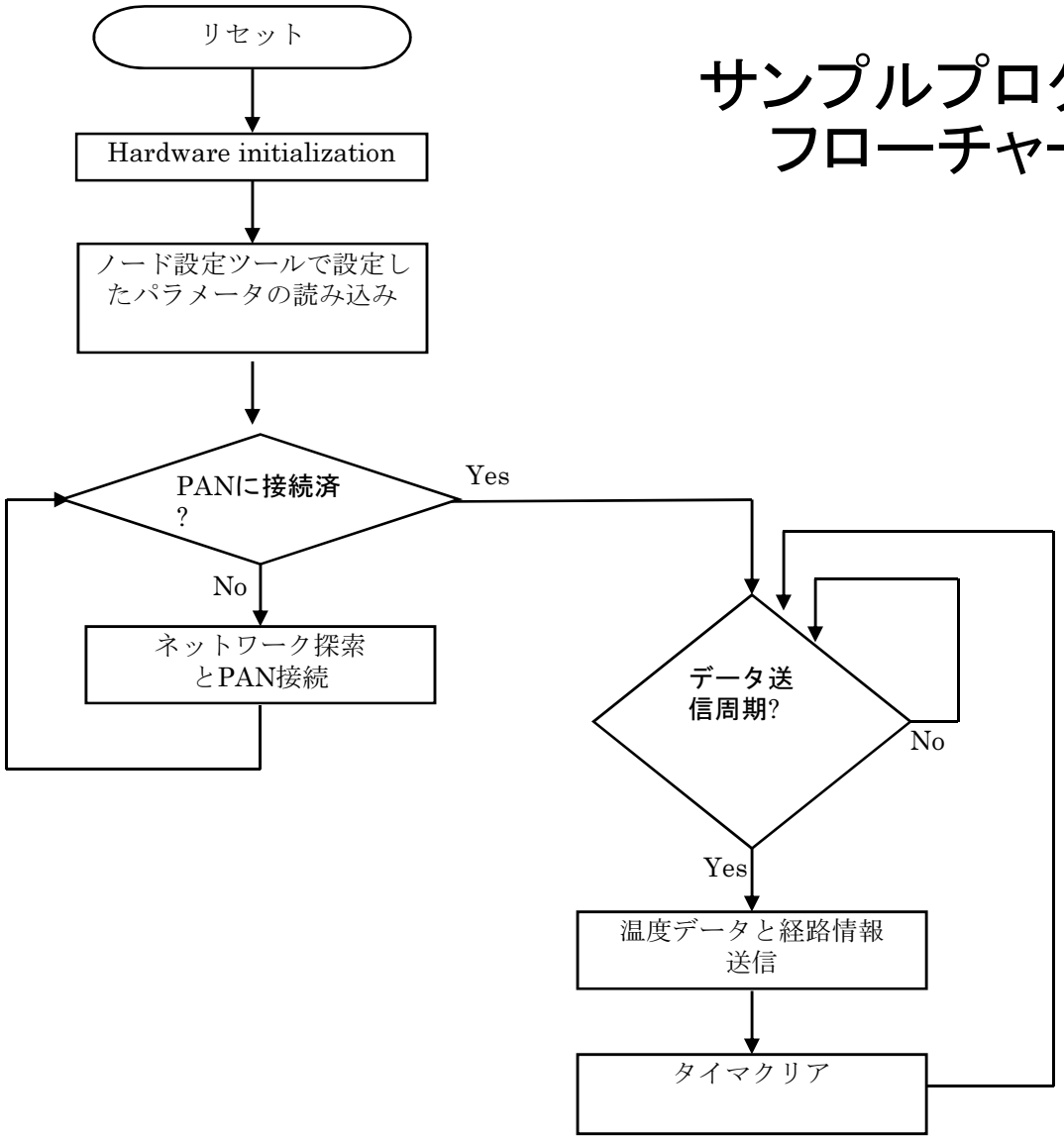
1. Profile IDを決める
    1. パブリックプロファイルとの重複は×
    2. 認証を取得する場合、ZigBee Allianceから固有のProfile IDの割り当てを受ける必要があります
2. エンドポイントを決める
3. 入力クラスター、出力クラスター番号のリストを決める
4. 各クラスターの処理内容を定義する
5. SK_APS_APPLICATION_OBJECT構造体を上記内容でセットアップする
6. SK_APS_StandardRegisterObject関数にてスタックに登録し、実行開始する。

サンプルプログラム

サンプルプログラムの主な機能

- ノード設定ツールにより挙動を設定できる
  - 起動時に特定アドレスから設定値を読み込んでいる
- 定期的に内蔵の温度センサ情報を送信
- ネットワーク構成をネットワークビューアで表示できる
  - 温度情報とともに経路情報をコーディネータに送信
- コーディネータは受信したパケットの情報をシリアルポートへ出力
- シリアルポートからハイパーターミナル等を使用することでコマンドの発行が可能

サンプルプログラム フローチャート



スタックの初期化

・Hardware_Initialize() を呼び出してハードウェアを初期化
・SK_Base_Init()関数を、MACアドレスと共に呼び出して、ZigBeeスタック全体を初期化

```
// =============================================================-
//   Hardware initialization
// =============================================================-
Hardware_Initialize();


// =============================================================-
//   Protocol stack initialization
// =============================================================-
// Specify a MAC address (Extended)
//
SK_Base_Init(0x00000000, 0x00000001);
```

アプリケーションの基本構造

```
while(1) {
//heartbeat
SK_Base_Main();

//Process input from PC (UART)
Interface();

//Body of periodical scanning
ScanMain();

//Body of sample application
AppMain();
}
```

SK_Base_Main()	ZigBeeスタックメイン
Interface()	シリアルポート処理
AppMain()	アプリケーション
ScanMain()	PAN探索と接続

各関数は待ち時間なく終了しメインループを回す必要があります

SK_Base_Main関数

メインループの中でSK_Base_Main() 関数を周期的に呼び出す必要があります。

→ビジーループによって処理の完了を待ってはいけません

→時間がかかる計算で関数が意図せずブロックしてしまう可能性に注意

```
//Main loop
while(1) {
        SK_Base_Main();
        SampleApplication();
}
```

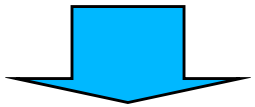
PAN探索と接続

ScanMain関数の中で周期的に接続を確認
接続済の場合、gnSK_NWK_isJoiningフラグが1になります

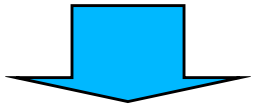
```
if( gNodeParams.m_DeviceType == DEVICE_ROUTER &&
    gnSK_NWK_isJoining == FALSE  ){
            SK_ZDO_NetworkDiscovery(
                  gnSK_APS_apsChannelMask, 4, ZDO_Discovery);
            SK_SLEEP(1000, SCAN_MAIN, 3);
      }
```

PAN探索と接続

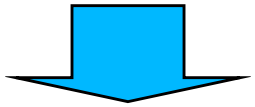
SK_ZDO_NetworkDiscoveryを発行



ZDO_Discoveryコールバックにて、発見したPANが通知される



SK_ZDO_Join関数を呼び出して接続処理を開始する



ZDO_Joinコールバックにて、接続結果が通知される

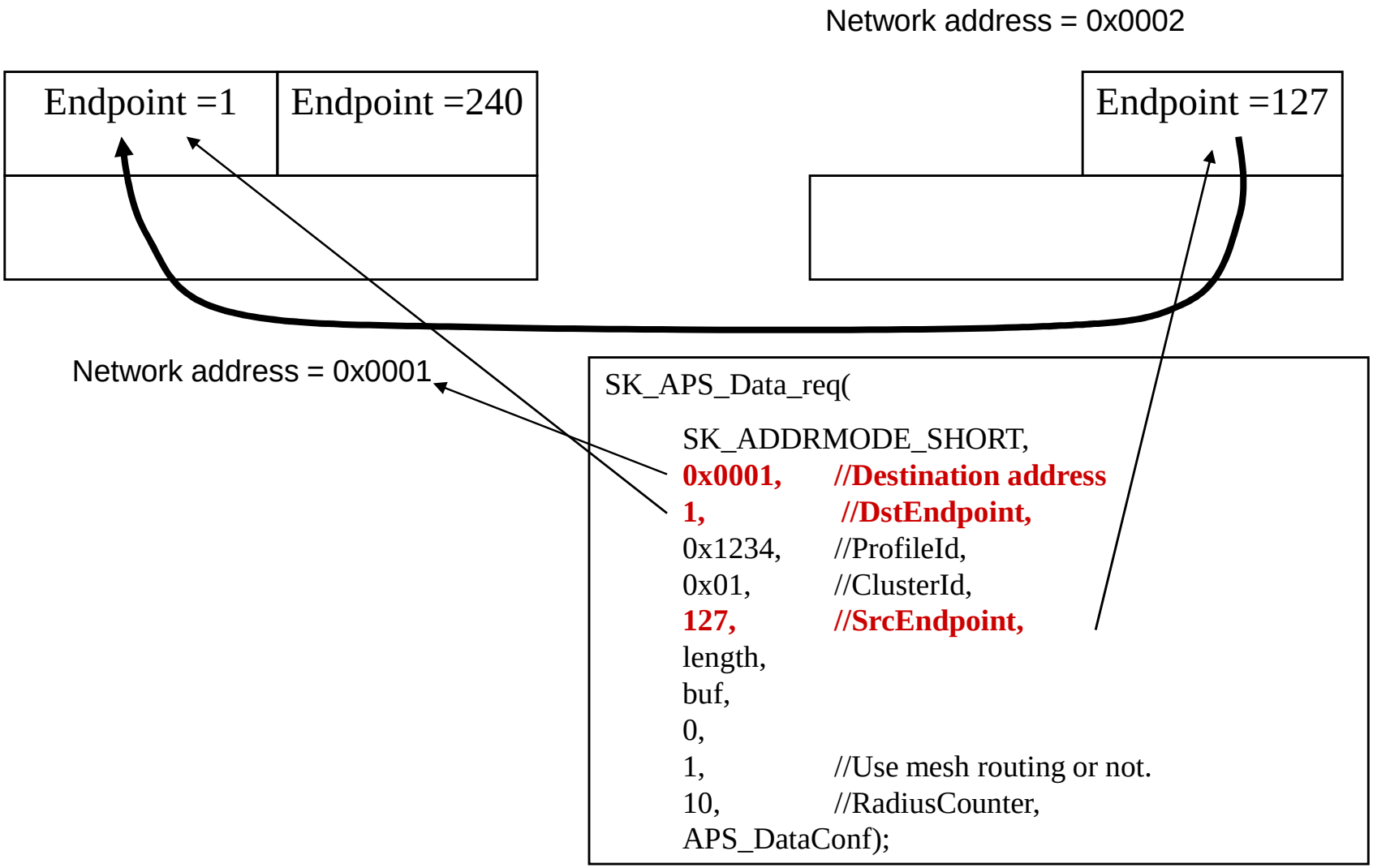
データの送信

AppMain関数内で周期的に温度データと経路情報を送信
データの送信にはAPS層の SK_APS_Data_req 関数を使います

```
SK_ER SK_APS_Data_req(
        SK_UB                   DstAddrMode,
        SK_ADDRESS_UNION        DstAddress,
        SK_UB                   DstEndpoint,
        SK_UH                   ProfileId,
        SK_UB                   ClusterId,
        SK_UB                   SrcEndpoint,
        SK_UB                   asduLength,
        SK_UB*                  Asdu,
        SK_UB                   TxOptions,
        SK_UB                   DiscoverRoute,
        SK_UB                   RadiusCounter,
        void (*WaitProc)(SK_VP) )
```

引数がたくさんありますが、まずはDstAddress と DstEndpointに注目します。

データの送信



データ送信の注意点

- 関数呼び出しからリターンした時点では、実際の送信はされていません。（バッファリングのみ）
- 送信完了と共に第12引数で指定した関数がコールバックされ、送信ステータスが通知されます。
  - 必ず送信完了コールバックを待ってから次の呼び出しを行う必要があります。

0xAD	対応するリンクキーが見つからず暗号化に失敗した場合。またはトラスト・センターによる認証を受ける前に暗号通信を行おうとした場合。
0xB0	何らかの理由で送信要求がタイムアウトした場合
0xC2	未接続状態でデーター送信を行おうとした時のようにデバイスの送信条件が整っていない場合
0xCD	対応するセキュリティキーが見つからず暗号化に失敗した場合
0xD2	ブロード・キャスト・テーブルメモリーがオーバーした場合。
0xD3	送信先までの経路が不明なため送信データが破棄され、代わりに経路発見を起動した場合
0xE9	送信相手からのMAC層ACKの受信に失敗した場合

データの受信

```
SK_APS_APPLICATION_OBJECT app;

app.m_Callback = APS_DataReceived;
app.m_ApplicationID = 0x0001;
app.m_ProfileID = 0x1234;
app.m_DeviceID = 0x9876;
app.m_InputClusterCount = 0;
app.m_OutputClusterCount = 0;

//Bind to endpoint 1
SK_APS_RegisterStandardObject(1, &app);
```

- 自分宛のデータを受信する度に、アプリケーションオブジェクト登録時に指定した関数がコールバックされます。
- 引数は1つで、SK_APSDE_DATA_INDICATION構造体へのポインタが渡されます。
- 中継データはコールバックされません。

サンプルプログラムにおける クラスターの使われ方

```
BANK_main void APS_DataReceived(SK_VP pResPkt) {
        SK_APSDE_DATA_INDICATION *AdInd;

        AdInd = (SK_APSDE_DATA_INDICATION *)pResPkt;

        if( AdInd->m_ClusterId == 1 ){

        } else if( AdInd->m_ClusterId == 2 ){
        ・・・
```

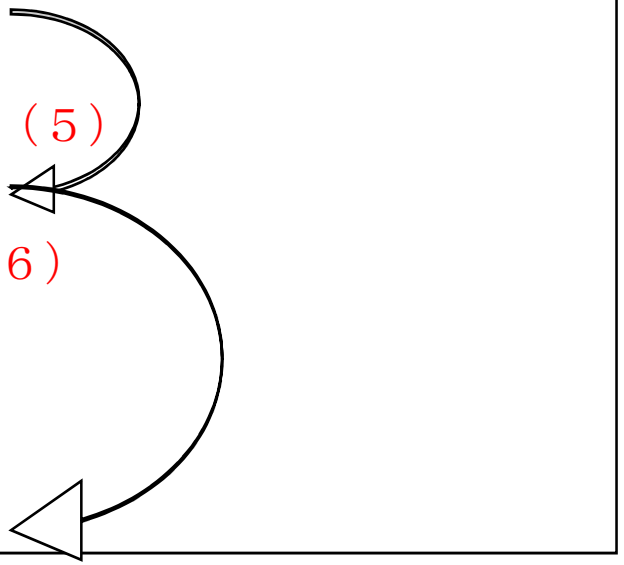
0x0001	温度データ
0x0002	ネットワークビューワ用経路情報
0x0003	ノード設定ツールのオンラインモードにおけるパラメータ設定
0x0004	ノード設定ツールのオンラインモードにおける遠隔コマンド実行
0x00FF	メニューコマンドのメッセージ送信

同期処理

```
SK_STATESTART(MSAMPLE);  （１）

void MacroSample() {
        SK_UB counter;      （２）
        SK_STATEADD(MSAMPLE, 1);      （３）

        while(1){
                counter++;      （４）
                SK_SLEEP(100, MSAMPLE, 1);（５）
                SK_print_hex(counter, 2);

                if( counter == 10 ) {
                        SK_print("Ten count!!");      （６）
                        break;
                }
        }
        SK_STATEEND(MSAMPLE);      （７）
}
```

同期処理

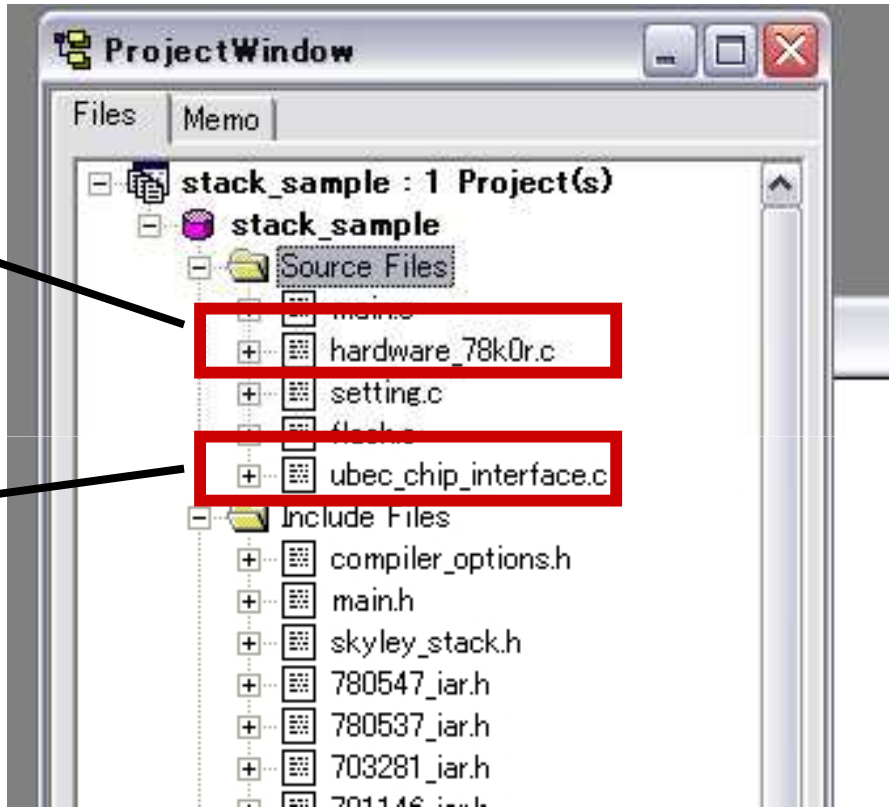
```
SK_STATESTART(MSAMPLE);  （１）

void MacroSample() {
        static SK_UB counter;      （２）
        SK_STATEADD(MSAMPLE, 1);      （３）

        while(1){
                counter++;      （４）
                SK_SLEEP(100, MSAMPLE, 1);  （５）
                SK_print_hex(counter, 2);
                if( counter == 10 ) {
                        SK_print("Ten count!!");      （６）
                        break;
                }
        }
        SK_STATEEND(MSAMPLE);      （７）
}
```



プロジェクト構成

- MCUの初期化、クロック設定
- UART、CSIの初期化
- タイマ割り込み設定
- A/D、RTC等

- 無線ICの初期化
- 無線ICとMCU間の通信制御
- 無線ICの省電力制御



ファイル構成

レイヤ名	ファイル名	内容
PHY層	prot_phy.c	UZ2400の初期化、レジスタ設定、RFのデータ送受信、スリープ制御などUZ2400に依存する関数が記述されています。
MAC層	prot_mac.c	MAC層のメインソースファイルで、IEEE 802.15.4のMAC機能を実現するプログラムが記述されています。
	prot_util_mac.c	IEEE 802.15.4のPIB値を設定・取得する関数、802.15.4フレームヘッダの解析・構築用関数、GTS関連処理が含まれます。
Network層	prot_nwk.c	ネットワーク層のメインソースファイルで、ネットワーク層の機能を実現するプログラムが記述されています。
	prot_routing_nwk.c	ネットワーク層機能のうち、データを受信した際の上位層へのディスパッチと転送計算が記述されています。
	prot_parse_nwk.c	ネットワーク層フレームヘッダの解析・構築用関数が記述されています。
	prot_tables_nwk.c	ネットワーク層が保持する各種テーブルの初期化とテーブル操作が記述されています。

ファイル構成

APS層	prot_aps.c	APS層のメインソースファイルで、APS層の機能を実現するプログラムが記述されています。
	prot_header_aps.c	APS層フレームヘッダの解析・構築用関数が記述されています。
ZDO層	prot_zdo.c	ZDO層のメインソースファイルで、ZDOサービスのクライアント側処理が記述されています。
	prot_server_zdo.c	ZDOサービスのサーバ側処理が記述されています。
AF層	prot_af.c	AF層のメインソースファイルで、AF層が提供する各種DescriptorやCapability情報を構築するための関数が記述されています。
Security層	prot_sec.c	ペイロードの暗号化・復号化用関数、各種セキュリティヘッダの解析・構築用関数が記述されています。
	rijndael-alg-fst.c	AES 128bit CCM* 暗号化・復号化を行う関数が記述されています。
フレームワーク	prot_base.c	SKSTACKの初期化とハートビード関数が記述されています。
	skyley_base.c	メッセージボックスの実現と、メモリの確保・開放を行う関数が記述されています。
I/O	uart_interface.c	UART入出力関数が記述されています。

株式会社スカイリー・ネットワークス

http://www.skyley.com/

info@skyley.com

製品についてのお問い合わせはこちらまで

sales@skyley.com