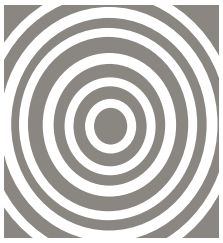
TinyOS/nesCプログラミングの初歩

トピックス

- アプリケーション: MyApp_Timer
  - アプリケーションディレクトリの内容
  - 詳細と方法
- コンフィグレーション
  - ワイアリング
- モジュール
  - StdControl
  - Timer
  - Leds
- Programmer's Notepadでプログラミング



MyApp_Timer – トップレベルモジュール

アプリケーションモジュールは*MyAppM.nc*ファイルにあります。

- Programmer's Notepadでコピー＆ペーストしてください。
- 次の表のパラメータを使用して保存してください。

注意: モジュールファイルはC言語のようなコードになります。
このコードの機能はタイマを開始して赤のLEDを反転させます。

ファイル名	MyAppM.nc
ファイルの種類	All files (*.*)
ファイルの形式	ファイル形式は変更しない

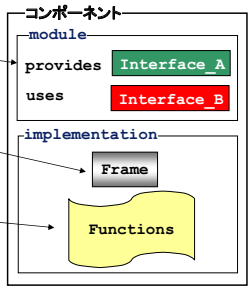
nesC モジュール – 最低限のモジュール (review)

インタフェースの実装

- Cコードで書かれる、ワイアリングしない

```
module ModuleName {
   provides interface StdControl;
}
implementation {
   // ========== FRAME ========
   int state;

   // ========== FUNCTIONS =====
   command result_t StdControl.init() {
      return SUCCESS;
   }
   command result_t StdControl.start() {
      return SUCCESS;
   }
   command result_t StdControl.stop() {
      return SUCCESS;
}
```



MyApp_Timer – トップレベルモジュール

1 of 3

```
/**
 * This module shows how to use the Timer and LED components
**/
module MyAppM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
```

MyAppMと名前を付けられたモジュールです。予めインタフェースをキーワードprovidesとusesで宣言します。

MyAppMモジュールはStdControlインタフェースを提供します。

- MyAppMは提供するインタフェースを実装しなければなりません。
- これはMyAppコンポーネントの初期化や開始をするのに必要です。

MyApp_Timer – トップレベルモジュール

1 of 3

```
/**
 * This module shows how to use the Timer and LED components
**/
module MyAppM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
```

タイマのインタフェース名はTimerです。

LEDのインタフェース名はLedsです。

多くのnesCアプリケーションでは、ある処理を一定間隔毎に行うことが一般的です。

- タイマはその機能を実現します。
- LEDの1つをアクティブにするのにLedsインタフェースを使用します。

nesC インタフェース – StdControl インタフェース (review)

StdControl インタフェースはトップレベルアプリケーションで最低限実装しなければなりません。

- StdControl インタフェースはTinyOSアプリケーションの基本的な機能である初期化、開始、停止を提供します。

StdControl は電源スイッチをオンオフするコンポーネントに似ています。

- また、ブート時の初期化処理を行います。

StdControl 内で連続した処理を行ってはいけません。

- 代わりにタイマを使用してください。

MyApp_Timer – Timer インタフェース

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
```

ここでTimer インタフェースは2つのコマンドを定義しています。

- start() と stop() コマンド

そして、1つのイベント

- fired() イベント

"result_t"とは

- コマンドやイベントが返すステータス値のデータタイプです。
- ステータス値はSUCCESS あるいは FAILのどちらかになります。

MyApp_Timer – Timer インタフェース (続き)

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
```

start()コマンドはタイマの種類とタイムアウトする時間間隔を指定するのに使用します。

- 引数のintervalはミリ秒です(正確には1/1024秒)。

タイマのタイプ

- TIMER_ONE_SHOT
  - 指定した時間の経過後1回のみ
- TIMER_REPEAT
  - stop() コマンドで停止するまで継続

MyApp_Timer – Timer インタフェース (続き)

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
```

アプリケーションが時間経過を知る方法
- イベントを受け取ります。

イベントとは
- インターフェースの実装から何か起こったときに合図されます。
- この場合、**fired()** イベントは指定された時間が経過したときに合図されます。

これは両方向インタフェースの例です。
- インタフェースはコマンドの呼び出しとイベントハンドラの両方を提供することが出来ます。
- インタフェースを使用するモジュールはこのインタフェースが使用するイベントを実装しなければなりません。

MyApp_Timer – トップレベルモジュール

```
implementation {
  /**
   * Initialize the components.
   *
   * @return Always returns <code>SUCC
   **/
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

/**
   * Start things up.  This just sets the rate for the clock
   * component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.start() {
    // Start a repeating timer that fires every 1000ms
    return call Timer.start(TIMER_REPEAT, 1000);
  }
```

MyAppM モジュールは**StdControl**インタフェースが提供する **StdControl.init()**, **StdControl.start()**, **StdControl.stop()** コマンドを実装します。

MyApp_Timer – トップレベルモジュール

```
implementation {
  /**
   * Initialize the components.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

/**
   * Start things up.  This just sets the rate for the clock
   * component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.start() {
    // Start a repeating timer that fires every 1000ms
    return call Timer.start(TIMER_REPEAT, 1000);
  }
```

実装された**StdControl**インタフェースの**init()** コマンドはサブコンポーネントのLedsを初期化するのに **Leds.init()**を呼び出します。

MyApp_Timer – トップレベルモジュール

```
implementation {
  /**
   * Initialize the components.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

/**
   * Start things up.  This
   * component.
   *
   * @return Always returns <code
   **/
  command result_t StdControl.sta
    // Start a repeating timer that fires every 1000ms
    return call Timer.start(TIMER_REPEAT, 1000);
  }
```

start() コマンドは1000ミリ秒の繰り返しタイマ("**TIMER_REPEAT**")を **Timer.start()**を呼び出して作成します。

MyApp_Timer – トップレベルモジュール

```
/**
   * Halt execution of the application.
   * This just disables the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop() {
    return call Timer.stop();
  }


/**
   * Toggle the red LED in response to the <code>Timer.fired</code>
   * event.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

stop()でタイマを停止します。

MyApp_Timer – トップレベルモジュール

```
/**
   * Halt execution of the application.
   * This just disables the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop() {
    return call Timer.stop();
  }


/**
   * Toggle the red LED in response to the <code>Timer.fired</code>
   * event.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

Timer.fired() イベントを実装します。**MyAppM**が使用するインタフェースのイベントは実装しなければなりません。

MyApp_Timer – トップレベルモジュール

```
/**
   * Halt execution of the application.
   * This just disables the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop() {
    return call Timer.stop();
  }


/**
   * Toggle the red LED in response to the <code>Timer.fired</code>
   * event.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

Timer.fired() イベントが起動され、**Leds.redToggle()**で赤のLEDを反転させます。

MyAppM.nc – Key Lesson

ほとんどのTinyOSアプリケーションは**Timer.fired()**で処理が始まります。
- Cのwhile loopを含む**main()**と比較してください。
- **StdControl.init()** の中でループすると全てのタスクがブロックされてシステムはフリーズします。
- TinyOSでの適切な処理はリピートタイマ(**REPEAT_TIMER**)を開始して **Timer.fired()**でロジックを実装することです。