[APNOTE13]

リモートデバイス(Arduino+XBee) の温度と

リードスイッチを監視

ABS-9000 DeviceServer

APNOTE13 Rev A.1.0

2011/09/22

オールブルーシステム (All Blue System)

ウェブページ: www.allbluesystem.com

コンタクト:contact@allbluesystem.com

1　　イントロダクション

遠隔地にあるリモートセンサーを手元の PC の Webブラウザから監視するシステムについて説明します。
リモートセンサーは市販の安価な Arduino[1]互換機に温度センサーとリードセンサーを接続して、XBee[2] 無線モジュールを使用してセンサーデータを送信します。このリモートセンサーを PC で実行するWebブラウザからリアルタイムで監視します。

リモートデバイスには温度センサーとリードセンサーが接続されて、温度と接点（ドアや窓の開閉）の状態を監視しています。Web ブラウザからサーバーに設置したリモート監視用のページを開いて、現在のリモートデバイスの温度を温度ゲージに表示します。また、リードスイッチの監視モードを “ON” にすることで、リモートデバイスに接続したリードスイッチの接点が変化したときに、指定したメールアドレスに警報メールを送信することができます。

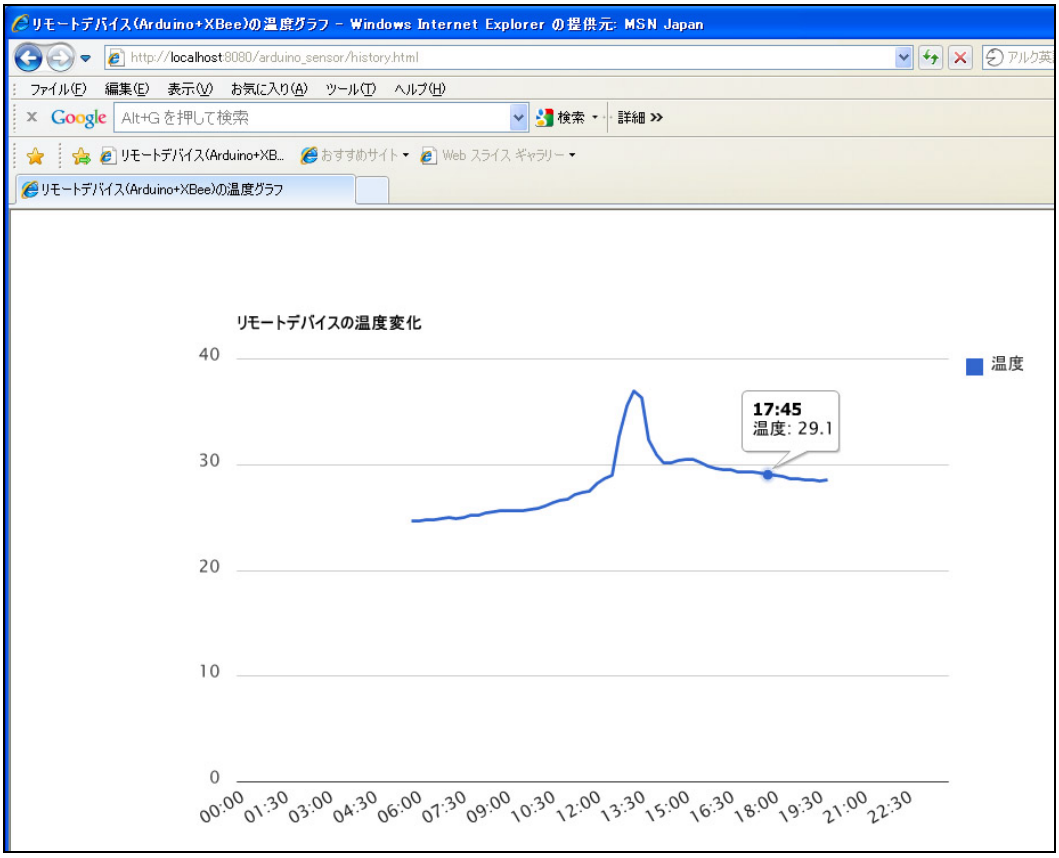
（リモートデバイスの温度と現在の監視モードを、Webブラウザで表示した画面）

リモートデバイスとサーバー間の通信には XBee デバイスを使用します。 リモートデバイスで検出したイベントデータ(リードスイッチの状態変化等) やリモートコマンド(コンフィギュレーション変更、ポート操作、マニュアルサンプリング等)は、全て XBee モジュールのデータパケット中に格納されて送受信されます。Arduino 互換上で動作するプログラムは、オールブルーシステム が提供している “TDCP (Tiny Device Control Program) for Atmega328” を使用しています。TDCP は DeviceServer のライセンスと共に使用する場合にはフリーで製品に搭載できます。(*1 参照)XBee デバイスは、Digi International Inc. 社製の IEEE 802.15.4 RF モジュールを使用します。サーバー側とリモートデバイスの両方で使用しています。

リモートデバイスでは、自動で定期的に I/O 値と A/D 変換値のサンプリングを行ってサーバーに送信するように設定しておくことで、温度変化のデータをサーバーのデータベースに格納しておくことができます。Web ブラウザから温度グラフ表示用のページを開いて、データベース中のサンプリングデータを集計して温度変化グラフを表示するこ

[1] Arduino (http://www.arduino.cc/)
[2] XBee XBee® and XBee□PRO® are registered trademarks of Digi, Inc.

とも出来ます



(定期的にリモートから送信された温度データを 15 分単位で集計表示した画面)

このアプケーションノートで説明したシステムを応用することで、センサーネットワークの I/O 装置をインターネットやイントラネットから操作したり、リモートデータを取得するシステムを構築することが簡単にできます。

センサーデバイス自身がインテリジェントな処理を行いますので、インターネット側に公開された Web サービスからは、デバイス側の I/O ポート値の変化監視やチャタリング抑止用フィルタ処理、定期サンプリングのタイマー処理などを切り離して、リモートデバイスの機能を利用できます。リモートデバイスとセンサーネットワーク間の無線通信の確立やリトライ処理、取得したセンサーデータの保管処理をセンサーネットワーク内の機能として Lua スクリプトで構築します。Web ページの JavaScript等からは、これらのセンサーネットワークの機能を Web API 経由で利用する方法を採用することで、仕様追加や動作環境の変更に柔軟に対応できるシステムを構築できます。

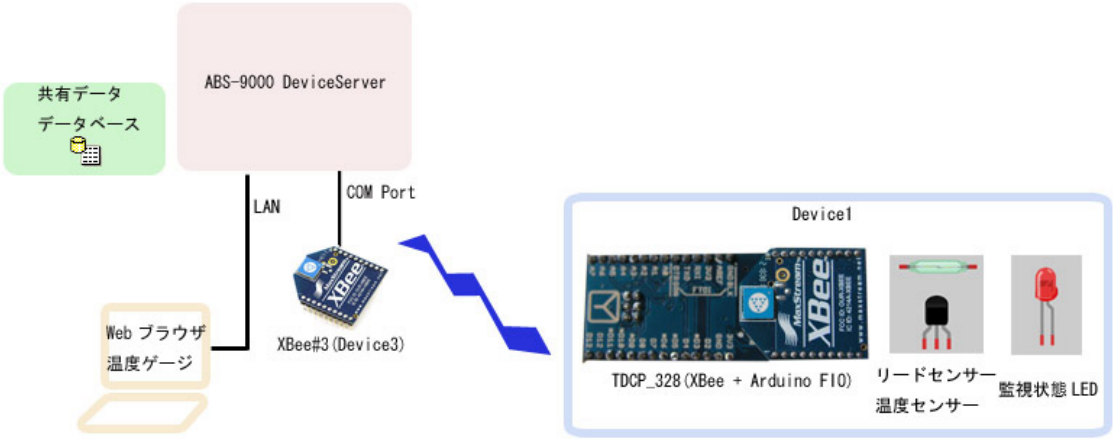
(*1) TDCP(TDCP for Atmega328)プログラムについて

リモートデバイス中のマイクロコントローラ上で動作させるプログラムはオールブルーシステムが提供している“TDCP (Tiny Device Control Program) for Atmega328” を搭載しています。DeviceServer のライセンスと共に使用する場合にはフリーで製品に搭載できます。TDCP 詳細なマニュアル下記で公開しています。

ホームページ http://www.allbluesystem.com

TDCP for Atmega328マニュアル http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf

1.1　　システム全体構成図



このシステムはセンサーとLED が接続されたリモートデバイスと、DeviceServer が動作するサーバーPC、Webブラウザを使用するクライアントPC から構成されています。クライアントPC を省略してサーバーPC のWebブラウザで表示することももちろん可能です。リモートデバイスは、屋内や屋外の離れた場所にに設置されていて、サーバーからリモートコントロールされています。

Webブラウザから、現在のセンサー状態を表示する画面を開くと、Web ページ中に記述された JavaScript によってリモートデバイスにアクセスして現在の温度を取得した後、温度ゲージを表示します。Web ページ中に表示されている監視モードチェックボックスをブラウザから操作すると、監視状態 LED を点灯または消灯させるためのポート操作をリモートコマンドで実行します。

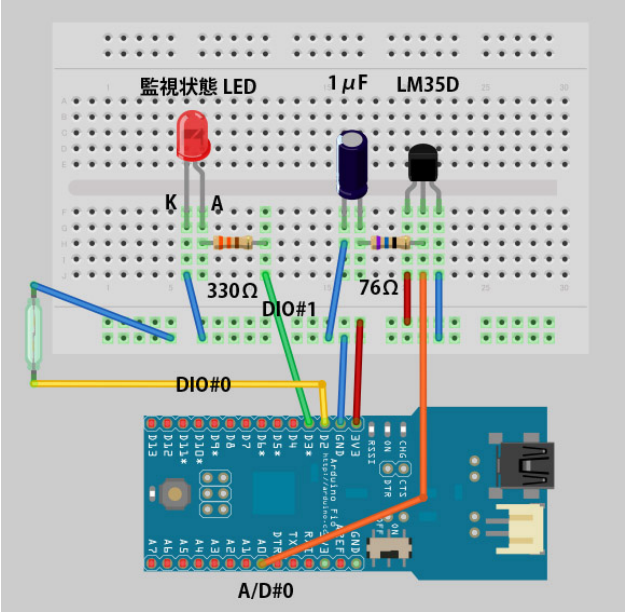
リモートデバイスのリードセンサーが反応すると、サーバーに対してイベントデータが送信されます。サーバーではそのイベントハンドラ中で、監視モード “ON”(監視状態 LED が点灯) になっている時には、予め設定されたメールアドレスに警報メールを送信します。

リモートデバイスで自動サンプリングモードを設定すると、リモートデバイスからは定期的にサンプリングデータ（温度センサーの A/D 変換値）がサーバーに送信されます。サーバーでは、そのサンプリングデータをデータベースに格納します。Web ブラウザから温度グラフ画面を開くと、Web ページに記述された JavaScript によってデータベース中の温度データを集計して温度グラフを表示します。

リモートデバイスとサーバーにはそれぞれ XBee 無線モジュールが接続されています。イベントデータの送信やサーバーからのリモートデバイス操作などは、全てXBee データパケットに格納したデータをやりとりする事でリモートコントロールされています。サーバーとリモートデバイス間をネットワークケーブルやシリアルケーブルで接続する必要がないので、複数のリモートデバイスからデータを取得する場合や、リモートデバイスが移動体の場合でも問題なく動作させることができます。

リモートデバイスとサーバー間の無線通信が一時的に不通になった場合でも、TDCP が持つパケット通信のリトライ機能によって通信安定性を向上させています。またシステムを構成するリモートデバイスがダウンまたは一時的な通信不能になった場合でも、システム全体の動作には影響せず該当デバイスのコントロールやイベント通知が一時的に影響を受けるだけになります。サーバーPC 自身が一時的にダウンした場合でも、特別な設定操作を行わずにサーバー PC復帰後は自動的にイベントの取得やクライアントからのリモート操作を継続できます。

1.2 リモートデバイス配線図



この例では、リモートバイスは Arduino 互換機にXBee デバイス用の I/F が内蔵された Arduino FIO[3] を使用しています。Arduino FIO では オールブルーシステム が提供している “TDCP (Tiny Device Control Program) for Atmega328” のファームウエアが動作しています。

温度センサー(LM35D) は RC ダンパー付きで、A/D#0 ポートに接続しています。
リードスイッチは DIO#0 ポートに接続されています。プロセッサの内部プルアップを有効にしますので、外部のプルアップ抵抗は省略しています。監視状態表示用の LED は電流制限抵抗を介して DIO#1 に接続します。

Arduino FIO と TDCP のポート配置や仕様については、”TDCP for ATmega328 ユーザーマニュアル”
(http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf)を参照してください。

2 システム動作概要

[3] http://www.arduino.cc/en/Main/ArduinoBoardFio

この章では、システムの機能とその動作について詳しく説明します。スクリプトとイベントハンドの内容については、後の章で説明していますので、そちらも参照してください。

2.1 リモートデバイスの温度取得時の処理フロー

リモートデバイスの現在の温度表示用の Web ページをアクセスすることで、リモートデバイスの温度をリアルタイムに取得することができます。

1. クライアントPCの Webブラウザから、リモートデバイスの温度表示用ページを開きます。
   例 "http://server_host_name:8080/arduino_sensor/index.html"
2. クライアントPC では、Web ページ中に記述された JavaScript から サーバーPCで動作している DeviceServer の WebAPI をコールして、"ARDUINO_SENSOR/GET_DATA" スクリプトを実行します。DeviceServer の WebAPI アクセスに関する仕様については、"DeviceServerユーザーマニュアル"（ http://www.allbluesystem.com/DeviceServer/DeviceServer_users.pdf ） 中の "WebAPI 機能"の章を参照してください
3. "ARDUINO_SENSOR/GET_DATA" スクリプト中から、リモートデバイスに対してリモートコマンド "force_sample" を実行します。この時、サーバーPC に接続された XBee デバイス(Device3) から リモートデバイスに接続された XBee デバイス(Device1) に対して、リモートコマンド文字列を格納した XBee データパケットが送信されます。
4. リモートデバイスに接続した XBee デバイス(Device1)でデータパケットを受信すると、Arduino 互換機上で動作している TDCP ファームウエアでパケットが解析されて、TDCP コマンド "force_sample" が実行されます。現在の I/O ポート値の取得とA/D 変換を実行して、サンプリング結果をXBee データパケットに格納して、TDCP コマンド送信元 XBee デバイス(Device3) に送信します。TDCP コマンドとデータパケットの詳しい仕様については、"TDCP for ATmega328 ユーザーマニュアル"（ http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf ）を参照してください
5. DeviceServer は XBee デバイス(Device3) を経由してリモートコマンド"force_sample" の結果を受け取ります。"force_sample" リモートコマンドを実行した "ARDUINO_SENSOR/GET_DATA" スクリプトでは、得られた A/D 変換値を元に温度を計算して、"ARDUINO_SENSOR/GET_DATA"スクリプト実行をリクエストした Web ページ中の JavaScript にリターン値としてその温度データを返します。
6. Web ページ中のJavaScript 中では、取得した温度データを元に Google Chart API[4] を利用して温度ゲージを表示します。

添付ファイルの Web ページ（index.html）では定期的に上記の動作を繰り返して、Web ブラウザ上で最新の温度データを常に表示できるようになっています。

[4] http://code.google.com/intl/ja/apis/chart/ , http://code.google.com/intl/ja/apis/chart/interactive/terms.html

2.2 リモートデバイスの監視モード設定時の処理フロー

リモートデバイスに接続したリードセンサーが “ON” または “OFF” に変化した時に警報メールを送信する機能があります。監視モードが “ON” の時だけ警報メールを送信することで、不用意にメールを送信しないようにしています。
監視モードの変更は、リモートデバイスの温度表示用 Webページ上のチェックボックスで操作します。
リモートデバイスに接続した、監視モード表示用LED はシステムの現在の監視モードに合わせて点灯または消灯します。

1 クライアントPCの Webブラウザから、リモートデバイスの温度表示用ページを開きます。
例 ”http://server_host_name:8080/arduino_sensor/index.html”

2 監視モードチェックボックスを操作すると、クライアントPC では、Web ページ中に記述された JavaScript から サーバーPCで動作している DeviceServer の WebAPI をコールして、”ARDUINO_SENSOR/WATCH_MODE” スクリプトを実行します。

3 ”ARDUINO_SENSOR/WATCH_MODE” スクリプト中から、最初に DeviceServer 内に作成した監視モードフラグ保存用のデータを更新します。このデータはDeviceServer 内蔵のデータベースに保存されています。次に、リモートデバイスに対してリモートコマンド ”port_bit” を実行します。この時、サーバーPC に接続された XBee デバイス(Device3) から リモートデバイスに接続された XBee デバイス(Device1) に対して、リモートコマンド文字列を格納した XBee データパケットが送信されます。

4 リモートデバイスに接続した XBee デバイス(Device1)でデータパケットを受信すると、Arduino 互換機上で動作している TDCP ファームウエアでパケットが解析されて、TDCP コマンド “port_bit” が実行されます。I/O ポートの指定ビットを更新して、LED を点灯または消灯させます。

＊実際には上記4 の動作の後で、リモートデバイスで実行したコマンドの結果ステータスを、XBee データパケットで送受信するステップ等が実行されます。

2.3 リードスイッチが反応した時の処理フロー

リモートデバイスのリードセンサーが変化した時に警報メールを送信する機能があります。警報メールは監視モードが “ON” の時だけ送信され、連続してリードスイッチが変化した場合でも１分以内の重複した警報メールは送信しないようにしています。

リモートデバイスのI/O ポートの入力値変化(リードスイッチ)の検出は、Arduino 互換機上で動作している TDCP ファームウエアで行います。リードスイッチ動作時のチャタリングは、TDCP 内部でフィルタリング(10ms 間隔のサンプリングとイベント検出)が行われますので、ユーザーが作成するスクリプトではこれらを考慮する必要はありません。

1 リモートデバイスに接続されたリードスイッチの入力が変化すると、TDCP のコンフィギュレーションで設定された XBee デバイスに対して “CHANGE_DETECT” イベントパケットを送信します。このシステムでは、TDCPコマンド “server_addr” コマンドを使用して予めサーバー PC に接続された XBee デバイス(Device3) のアドレスがイベント送信先に設定されています。TDCP コマンドとイベントパケットの詳しい仕様については、”TDCP for ATmega328 ユーザーマニュアル”（ http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf ）を参照してください。

2 DeviceServer は XBee デバイス(Device3) を経由してイベントパケット “CHANGE_DETECT” を受信します。XBee データパケットが解析されて、TDCP デバイスのイベントであることが判明すると、XBEE_TDCP_DATA イベントハンドラスクリプトが実行されます。このスクリプト中で、デバイスの確認とイベント種別をチェックします。その後、現在の監視モードが “ON” の場合にのみ、”ARDUINO_SENSOR/ALARM_MAIL” スクリプトを起動します。”ARDUINO_SENSOR/ALARM_MAIL” スクリプトはメール送信時にメールサーバーに接続して処理を行うため、イベントハンドラ自身の処理時間が長くなるのを防ぐために、別スレッドで実行します。

3 ”ARDUINO_SENSOR/ALARM_MAIL” スクリプトでは、最初にリードスイッチが連続して反応した場合に備えて、１分以内のメール送信を抑止するために、最後にメールを送信した時刻のチェックを行います。その後、警報メールをスクリプト中に記載されたメールアドレスに送信します。メールの送信時刻のチェックは、同時にこのスクリプト自身が並行して実行されている場合を考慮してクリティカルセッションを利用して排他制御します。

2.4 温度変化グラフ表示の処理フロー

当日の温度変化をグラフで表示する機能があります。この機能は２つの部分から構成されています。定期的なサンプリングを行ってサーバー中のデータベースに温度データを登録する機能と、クライアント PC のWeb ブラウザから温度変化グラフ表示用ページを開いたときに、データベースを集計してグラフを表示する部分です。

定期的なサンプリングを行う機能の動作フローを以下に示します。

1 リモートデバイスでは、TDCP のコンフィギュレーションで設定された XBee デバイスに対して定期的に I/O ポートと A/D チャンネルのデータを取得して “SAMPLING” イベントパケットを送信します。このシステムでは、TDCPコマンド “server_addr” コマンドを使用して予めサーバー PC に接続された XBee デバイス(Device3) のアドレスがイベント送信先に設定されています。また、サンプリング間隔は “sampling_rate” コマンドで、600秒(10 分)間隔で送信するように設定されています。TDCP コマンドとイベントパケットの詳しい仕様については、”TDCP for ATmega328 ユーザーマニュアル”（ http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf ）を参照してください。

2 DeviceServer は XBee デバイス(Device3) を経由してイベントパケット “SAMPLING” を受信します。XBee データパケットが解析されて、TDCP デバイスのイベントであることが判明すると、XBEE_TDCP_DATA イベントハンド

ラスクリプトが実行されます。このスクリプト中で、デバイスの確認とイベント種別をチェックします。その後、サンプリング結果をデータベースに保存するために、”ARDUINO_SENSOR/REGISTER_ACTIVITY” スクリプトを起動します。”ARDUINO_SENSOR/REGISTER_ACTIVITY” スクリプトはデータベースに登録処理を行うため、イベントハンドラ自身の処理時間が長くなるのを防ぐために、別スレッドで実行します。

3 ”ARDUINO_SENSOR/REGISTER_ACTIVITY” スクリプトでは、A/D 変換値を温度に変換した後、集計用データベースに登録します。

4 温度変化グラフ表示用ページでは当日分のデータのみを表示するようになっていますが、JavaScript と HTML を修正して任意の日時のデータを表示することも出来ます。このシステムでは、データベース中のサンプリングデータは、過去３０日よりも古いデータは削除してデータが無制限に増えるのを防止しています。
“PERIODIC_TIMER” スクリプトは DeviceServer から１分に一回定期的にコールされますので、このスクリプト中に、１時間に一回毎に過去データ削除用のスクリプト “ARDUINO_SENSOR/SENSOR_DATA_PURGE” を実行するように記述します。

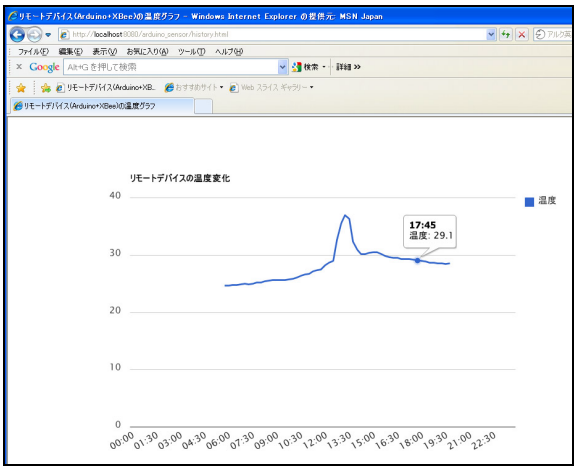
クライアント PC のWeb ブラウザから温度変化グラフ表示用ページを開いたときの動作フローは以下の様になります。

1 クライアントPCの Webブラウザから、温度変化グラフ表示用ページを開きます。
例 ”http://server_host_name:8080/arduino_sensor/history.html”

2 クライアントPC では、Web ページ中に記述された JavaScript から サーバーPCで動作している DeviceServer の WebAPI をコールして、” ARDUINO_SENSOR/SUMMARY_DATA” スクリプトを実行します。

3 ”ARDUINO_SENSOR/SUMMARY_DATA” では、統計データベースをアクセスして、15 分単位に温度データの平均値を計算して、集計結果をリターン値に設定します。

4 Web ページ中のJavaScript 中では、取得した集計を元に Google Chart API を利用して温度グラフを表示します。



3 必要な機材・リソース

3.1 サーバー

必要なシステムやデバイス	説明
サーバーPC (OS:Windows XP 以降)	ABS-9000 DeviceServer の動作する PCが１台必要です。
XBee Explorer USB[5]	サーバー PC と XBee デバイスを 仮想 COM ポート経由で接続するために使用します。
XBee デバイス	サーバー PC 用に Digi International Inc. 社製 XBee IEEE 802.15.4 デバイスが 1台必要です。DeviceServer の COM ポートに接続して各リモートデバイス間との通信を行います。 DeviceServer は、XBee デバイスのファームウエアバージョンの “10CD” 以降にのみ対応しています。(必要に応じて XBee ファームウエアの更新を行ってください)

3.2 リモートデバイス

必要なシステムやデバイス	説明
Arduino FIO TDCP for Atmega328 ファームウエア導入済みのもの	詳しくは ”TDCP for ATmega328 ユーザーマニュアル” (http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf) を参照してください。
XBee デバイス	Arduino FIO用に Digi International Inc. 社製 XBee IEEE 802.15.4 デバイスが 1台必要です。リモートデバイスに設置してサーバー間との通信を行います。 TDCP は、XBee デバイスのファームウエアバージョンの “10CD” 以降にのみ対応しています。(必要に応じて XBee ファームウエアの更新を行ってください)
温度センサー LM35D リードスイッチ LED その他、電流制限用抵抗と温度センサーRC ダンパー用コンデンサと抵抗	パーツとセットアップ詳細は、”リモートデバイス配線図” の章を参照して下さい。

リモートデバイスは、複数同時に使用することも出来ます。この場合には各々のリモートデバイスに接続した XBee デバイスアドレスとノード名(NodeIdentifier) にそれぞれ別の16ビットアドレスと名前を設定します。また添付ファイル中のクライアント側で表示する Web ページ(index.html, history.html) に、複数デバイスを表示できるように変更を加える必要があります。

[5] http://www.sparkfun.com/products/8687

3.3 クライアント

必要なシステムやデバイス	説明
クライアントPC	サーバーPC に Web ブラウザでアクセスして、温度ゲージと温度グラフを表示させます。サーバー PC 上で Web ブラウザを起動する場合には、クライアントPC は不要です。

4 セットアップ

4.1 XBee デバイス初期設定(サーバー側・リモートデバイス共通)

XBee デバイスを DeviceServer とリモートデバイスの TDCPで使用するために初期設定が必要です。
使用するすべての XBee デバイスで下記の設定を行ってください。

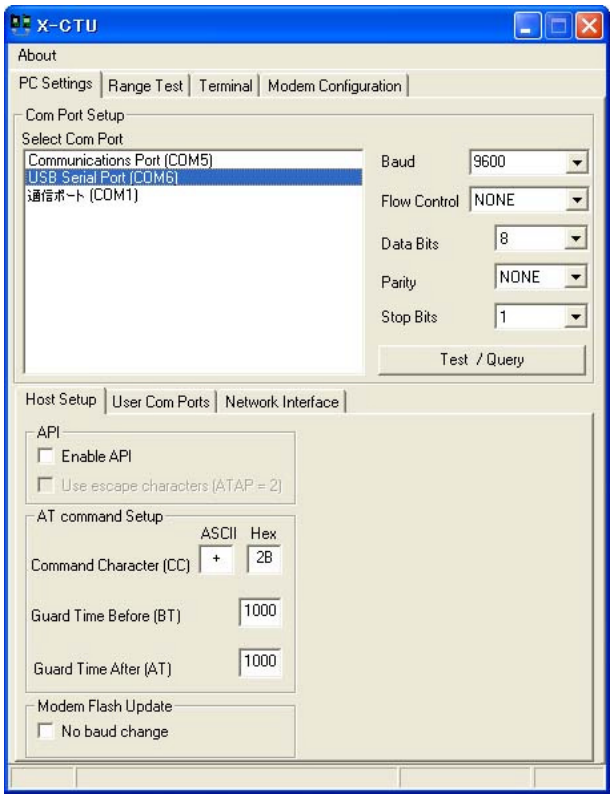
最初に DeviceServerとの接続に必要な最低限の設定を COM ポート経由で行います。Sparkfun Electoronics 社製の XBee Explorer USB などを使用して、仮想 USB ポート経由で接続して設定してください。(これ以外の方法で COM ポート接続する場合も手順は同じです)

リモートデバイス用の XBee デバイスの設定についても、DeviceServer に接続する XBee デバイスを設定するときに使用した XBee Explorer USBを使用して行います。この時には、XBee Explorer USB のソケットから一時的に XBee デバイスを入れ替えて作業を行ってください。

XBee デバイス デフォルト値から変更が必要な設定値	
API モード	1 (default は 0)
PAN(Personal Area Network) ID	任意の値 (default はOx3332) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルではPAN にOxAB90 を使用しています。
16bit Source Address	同一PAN ID 内でユニークな値 (default は 0x0000) この値は、ここで設定しなくても後から XBee 管理プログラムで設定可能ですが、デバイス一覧から選択したデバイスがどのデバイスであるかを見分けることが容易になるように便宜的にここで設定します。 すべてのデバイス間で違った値を設定してください。 (0x0000,0xFFFF,0xFFFE を除く) 例えば、0x0001, 0x0002,0x0003 など

上記３つの 初期設定のコマンドをXBee に送信するために、XBee デバイスを XBee Explorer USB に接続して PC から仮想COM ポート経由でアクセスできるようにします。その後、Digi international Inc. 社製の X-CTU プログラム、または汎用のターミナルエミュレータプログラムなどから AT コマンドをすることで設定を行います。XBee と COM ポートのボーレートは初期設定の 9600 bps にして下さい。ターミナルエミュレータプログラムを使用するときは、プログラムの設定でローカルエコー “ON”, 改行コード受信時の動作を CR(改行) + LF(行復帰) にするとコマンド実行結果が見やすくなります。

X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。

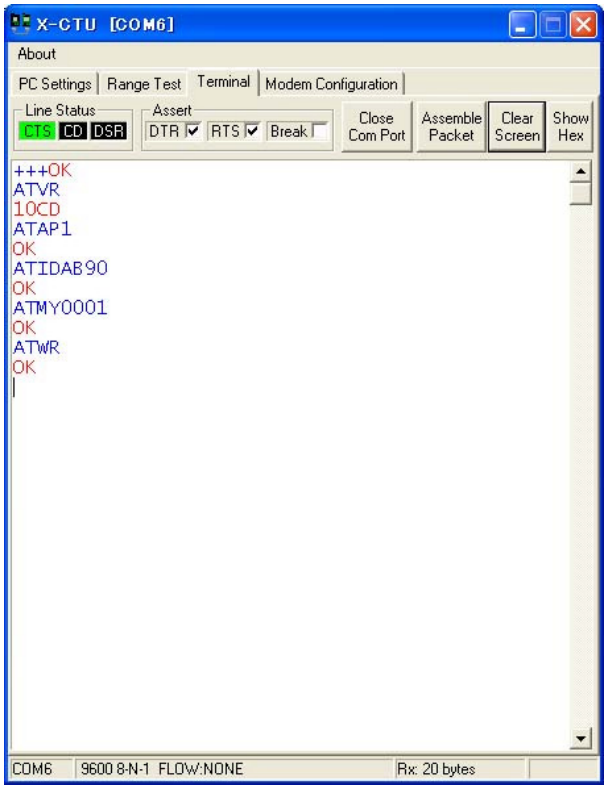


Terminal タブを選択してターミナル画面を表示します。キーボードから、”+++” を入力して、コマンドモードに入ります。コマンドモードに入ると “OK” が表示されますので、続けて以下のコマンド文字列を入力してください。コマンド入力の時間がかかりすぎると、自動的にコマンドモードから抜けてしまいますので、そのときには、”+++” を入力して最初からコマンドを入力し直して下さい。

```
ATVR
ATAP1
ATIDAB90
ATMY0001
ATWR
```

最初に ATVR でファームウエアバージョンを表示しています。”10CD” 以降になっていることを確認してください。

ATAP1 は、API モードを “1” に設定しています。ATIDAB90 は PAN_ID を 0xAB90 に設定しています。もし別の PAN_ID を使用するときには適宜変更してください。次に、ATMY0001 で、デバイスの16 bit Source Address を “0x0001” に設定しています。この部分は、デバイスごとにユニークな値になるように変更して下さい。
最後に、ATWR で、設定値を不揮発メモリに書き込みます。入力時の画面表示は以下のようになります。



X-CTU プログラムを終了します。 その後、XBee Explorer USB に接続する XBee デバイスを切り替えて、使用するすべての XBee デバイスについて同様に初期設定を行って下さい。
このときに、設定した16bit Source Address の値をデバイス機器にマーキングしておくと、後で XBee デバイス管理プログラムでデバイスを選択するときに、識別し易くなります。

システム上の XBee デバイスの設定値例は、以下のようになります。

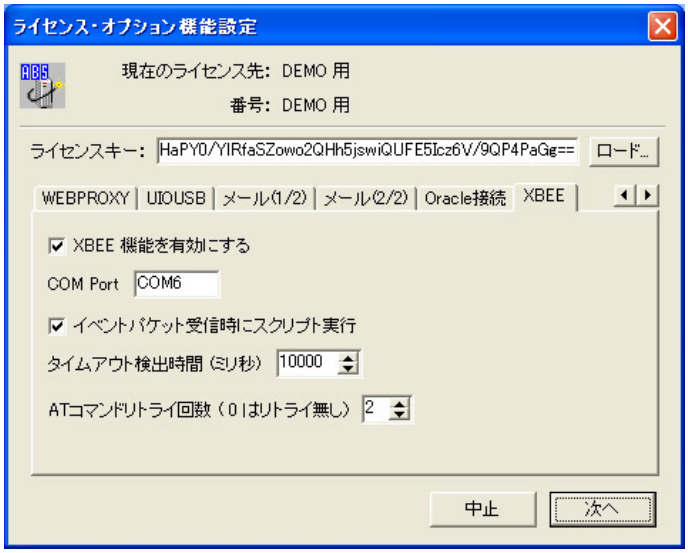
デバイス番号(用途)	API モード(ATコマンド)	PAN_ID(ATコマンド)	16bit Address(ATコマンド)
XBee#1(リモートデバイス)	1(ATAP1)	0xAB90(ATIDAB90)	0x0A01(ATMY0A01)
XBee#3(DeviceServer接続)	1(ATAP1)	0xAB90(ATIDAB90)	0x0C03(ATMY0C03)

4.2 XBee デバイスを DeviceServer に接続

XBee デバイスの初期設定後に、XBee#3(サーバー用) を XBee Explorer USB に接続します。
デバイスが PC に接続されたら、DeviceServer から XBee デバイスを使用するための COM ポートの設定を行います。
サーバー設定プログラム(ServerInit.exe)を起動して、XBEE タブを選択して COM ポート番号を設定して “XBEE 機能を有効にする” にチェックをつけてください。

また、XBee データパケット受信時に XBEE_IO_DATAや XBEE_EVENT_DATA イベントハンドラを実行するために、”イベントパケット受信時にスクリプト実行” にもチェックを付けてください。

サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。



4.3 リモート側デバイス接続

リモートデバイスに内蔵する XBee デバイスを、一時的にサーバー PC の XBee Explorer USB などに接続して初期設定した場合には、取り出してリモートデバイスの Arduino FIOボードにセットします。

リモートデバイスの電源を接続して、サーバー側の XBee と通信できる状態にしておきます。

これ以降の リモートデバイス上の XBee デバイスの設定変更や、リモートデバイスの設定(TDCP プログラムのコンフィギュレーション)はすべて サーバー PC からリモートコマンドで操作できます。

4.4 マスター登録と XBee 詳細設定

リモートデバイスの XBee デバイスと、サーバー PC に接続した XBee デバイスを DeviceServer のマスターファイルに登録します。XBee デバイスの初期設定で設定しなかった NodeIdentifierなどの詳細設定もここで行います。

XBee デバイス管理プログラムを使用して、同一 PAN ID をもつ XBee デバイスを DeviceServer に登録したり、デバイス自身の設定内容を変更します。プログラムメニューから “ALL BLUE SYSTEM” -> “クライアント起動”を選択・実行します。ログインするときは、管理者特権をもったユーザー（例えば DeviceServer セットアップ時に管理者アカウントとして登録したユーザーなど）でログインしてください。デスクトッププログラムが起動したら、“XBee” ツールボタンを選択してXBee デバイス管理プログラムを起動します。

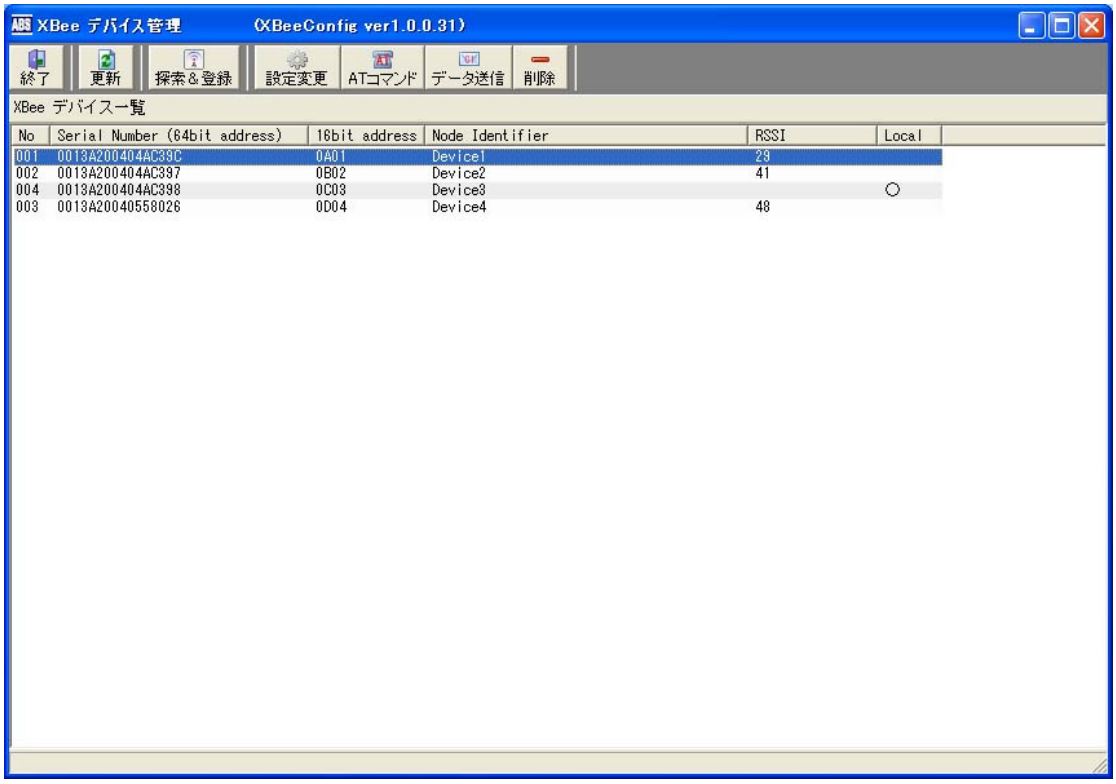
DeviceServer では 登録済みの XBee デバイスをマスターファイルに記録しています。

XBee デバイス登録は、”探索＆登録” ボタンを押すことで、同一 PAN ID のリモートデバイスを見つけて、自動的にマスターファイルに登録します。既に、登録済みのXBee デバイスの項目は最新の情報でマスターファイルの内容が更新されます。DeviceServer に COMポートで直接接続された XBee デバイスについても同様に自動登録されます。

XBee デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

登録確認ダイアログが表示されますので、”OK” を押します。

DeviceServer の COM ポートに直接接続された XBee デバイスで “Node discover” が実行され、付近にある同一 PAN ID の XBee デバイス情報を取得して、自動的にマスターファイルに登録が行われます。XBee デバイス管理プログラムのデバイス一覧には、登録済みのXBee デバイスが表示されます。



(Node Identifier 項目は、詳細設定修正後に 再度 “探索＆登録” ボタンを押すことで上記のように表示されます)

XBee デバイスの詳細設定を変更するために、XBee デバイス管理プログラムのデバイス一覧から対象デバイスを選択して、”設定変更” ツールボタンを押します。初期設定時に 16 bit Source Address を設定したときは、その値がデバイス一覧に表示されていますので、変更対象の XBee デバイスを確認できます。

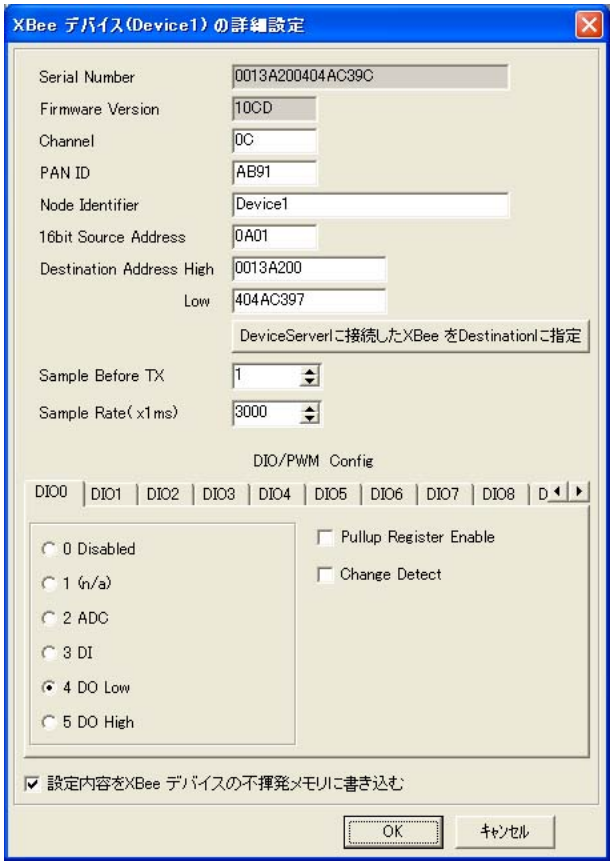
選択したXBee デバイスと通信を行って、現在のデバイス情報を取り込みます。

もしエラーが発生したときは、選択したXBee デバイスとの間で通信ができない状態になっていますので、通信経路

や電源を確認してください。

XBee デバイスの現在の設定値を取り込んだ後、詳細設定変更ダイアログが表示されます。
ここでは各 XBee デバイスの Node Identifier の設定を行って下さい。Node Identifier に指定できる文字は ASCII で 20文字までです。また、ダイアログに表示されている 16 bit Source Address が、対象のデバイスであるかどうかの確認も行ってください。ここで 16 bit Source Address を任意の値に変更することもできます。

デバイス番号 (16 bit Source Address)	デバイス名 (Node Identifier)
XBee#1 (0x0A01) リモートデバイス	Device1
XBee#3 (0x0C03) DeviceServer 接続	Device3



項目を修正した後、 ”設定内容を XBee デバイスの不揮発メモリに書き込む” にチェックを付けて “OK” を押してください。
今回のシステムでは XBee デバイス上の I/O や ADC、サンプリング機能は使用しませんので、”Node Identifier” 以外の項目を設定する必要はありません。

XBeeデバイスの Node Identifier を変更したときは、XBee デバイス管理プログラムのデバイス一覧に表示されている、マスターファイルも更新しておく必要があります。XBee デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

デバイスのNode Identifier または 16bit Source Address以外の詳細設定を変更する場合には、マスターファイルの更新は必要ありません。

4.5 リモートデバイス TDCP 設定

リモートデバイスの TDCP ボードの設定を行うために、下記のTDCP コンフィギュレーション値を設定します。

TDCP プログラム設定項目

項目名称	設定内容	設定用 TDCPコマンド
イベントデータ送信先 XBee アドレス	サーバーPC に接続した XBee デバイスのシリアル番号(16bitアドレス)	“server_addr,<16bitAddr(*1)>”
Arduino DIO ポートの入出力モード設定	bit#1(監視モード LED) を出力モードにしてその他のビット(bit#0: リードスイッチ)は全て入力モードに設定する。	dio_config,02
DIOがで入力モードに設定されているポートの、ビット毎のプルアップ設定	全ての入力ポートのプルアップを有効にする。	pullup,FF
DIO ポート値変化を、検出するビットを指定する	bit #0 のビット値を監視して、変化があった場合には “CHANGE_DETECT” イベントを送信する。	change_detect,01
A/D 変換のリファレンス電源を選択する	プロセッサ内部のリファレンス電源 1.1V を使用する	adc_vref,3
定期的に自動サンプリングを行う間隔（秒）を設定する	１０分毎に、”SAMPLING” イベントを送信する	sampling_rate,600
コンフィギュレーション保存	TDCP 設定内容を プロセッサ内蔵の EEPROM に保存	“config_save”
CPU リセット	TDCP プログラムのリセット。 コンフィギュレーションで保存された新しい app_modeで再起動する	“reset”(*2)

(*1) サーバーPC に接続した XBee のシリアル番号や16 ビットアドレスは XBee デバイス管理プログラムからデバイス一覧で確認するか、スクリプト中から xbee_my_serial_number() 関数を使用して取得できます。
(*2) リセットコマンド実行時は、リモートデバイスからのリプライパケットは常に返らないので、スクリプトから xbee_tdcp_command() 関数を使用してリセットコマンドを送信する場合には、no_result パラメータに true を指定して下さい。

設定用 TDCP コマンドは、スクリプト中に全ての設定用 TDCPコマンドを記述して実行する方法と、１コマンドごとに XBee デバイス管理プログラムから送信する方法のどちらか方法で実行できます。

下記に、それぞれの方法で設定を行う場合について記述しますが、どちらかの方法で設定してください。

複数のリモートデバイスがある場合には対象デバイスを切り替えてすべてのリモートデバイス中の TDCP プログラムの設定を行ってください。

4.5.1 ARDUINO_SENSOR/SETUP_DEVICE スクリプト作成

下記のスクリプトを作成して、リモートデバイスの TDCP プログラムの設定を行います。

```
file_id = "SETUP_DEVICE"
--[[
          TDCP_328 コントローラ初期設定スクリプト
]]
local device = "Device1"
log_msg("start..",file_id)
local stat,serial,addr16,result

stat,serial,addr16 = xbee_my_serial_number()
if (not stat) or (addr16 == "") then error() end
log_msg("DeviceServer's XBee address is " .. addr16,file_id)

stat,result = xbee_tdcp_command(device,"server_addr," .. addr16)
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"dio_config,02")
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"pullup,FF")
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"change_detect,01")
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"adc_vref,3")
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"sampling_rate,600")
if (not stat) or (result[2] ~= "1") then error() end

stat,result = xbee_tdcp_command(device,"config_save")
if (not stat) or (result[2] ~= "1") then error() end
```

```
stat,result = xbee_tdcp_command(device,"reset",true)
if (not stat) then error() end
```

local device = "Device1" は、リモートデバイスに接続した XBee デバイスの NodeIdentifier を Lua ローカル変数 “device” に設定しています。local 宣言をしないで変数を使用するともできます。ローカル宣言しておくと、使用した変数がスコープ（スクリプト全体、if, while 文などのブロック）から抜ける時に自動的に削除されますので、意図しない変数を不注意で使用するなどの不具合防止に役立ちます。

--[[と]] で囲まれた部分はコメント行です。また行中の -- より右側の部分もコメントです。

log_msg() は、ログにメッセージを出力するための関数です。メニューから “All Blue System” -> “ログコンソール” を選択してログ出力を画面にも表示している場合には、この関数で指定したメッセージがリアルタイムに出力されます。ログコンソールを起動していなかった場合でも、ログサーバーには全てのログが記録されています。ログサーバーで保存されているログを後で確認する場合には、ログコンソールを起動して”ログファイル切り替え” ボタンを押します。この操作でメモリ中に溜まっているログがファイルに出力されます。その後、”ログフォルダを開く” ボタンを押して、確認したいログファイルをエディタで開いて過去のログを表示できます。

セミコロン”;” は文の終端を表しています。スクリプト中の文の終端は Lua の構文から自動的に判断されますので、セミコロンを記述しなくてもエラーにはなりません。

file_id = "SETUP_DEVICE" は、log_msg() 関数でログにメッセージを出力するときのモジュール名を統一させるための変数です。この変数を使用しないで log_msg() の第二パラメータに文字列をその都度指定しても構いません。

stat,serial,addr16 = xbee_my_serial_number() は、サーバーと USB エキスプローラで接続した XBee デバイスのシリアル番号(64ビットアドレス) と 16 ビットアドレスを取得する関数です。

stat,result = xbee_tdcp_command(device,"server_addr," .. addr16) は、リモートデバイスに “server_addr,<addr16>” (<addr16> は、サーバーPC に接続したXBee の16ビットアドレス) のコマンドを送信しています。これによって リモートデバイスの設定値が変更されます。 xbee_tdcp_command() は実行結果ステータスと、リプライパラメータの２つの値を受信します。実行結果ステータス値が True またはFalse の論理値が返されるので、それをif文でチェックしてエラーが発生したときに、スクリプト動作を中止するようにしています。また、リモートデバイスとサーバー間の通信は成功してリプライデータを受信したものの、送信したTDCP コマンド自身の実行に失敗(TDCP コマンドパラメータエラー、TDCP コマンド実行時エラー)していた場合には、xbee_tdcp_command() の実行結果は True で、第２リプライパラメータに “0” が返ります。第２リプライパラメータが ”1” の場合にはTDCP コマンド実行が成功したことを意味します。

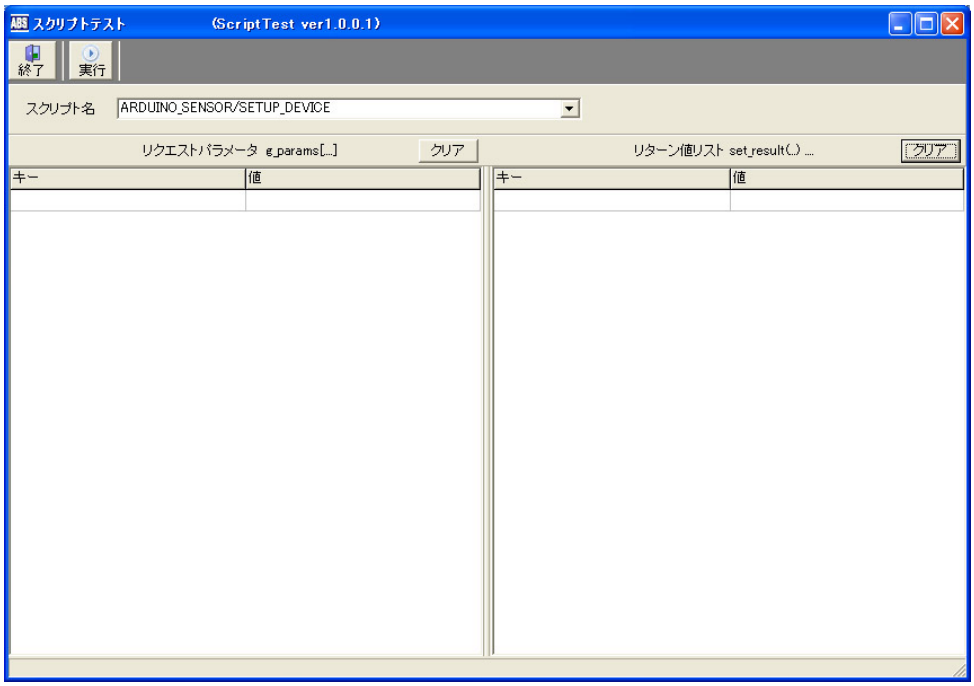
全ての TDCP 設定値を同様に、xbee_tdcp_command() 関数を使用して設定します。

最後の `stat,result = xbee_tdcp_command(device,"reset",true)` は、リモートデバイスをリセットします。このときリプライステータスは一切返ってこないので、`xbee_tdcp_command()` の第３パラメータに true を指定してリプライ受信を行わないようにします。パラメータを指定しないと、リプライを正常に受信するまで自動的にリトライ操作（デフォルトで２回）が行われて、リセットコマンドが複数回実行されることになりますので注意してください。

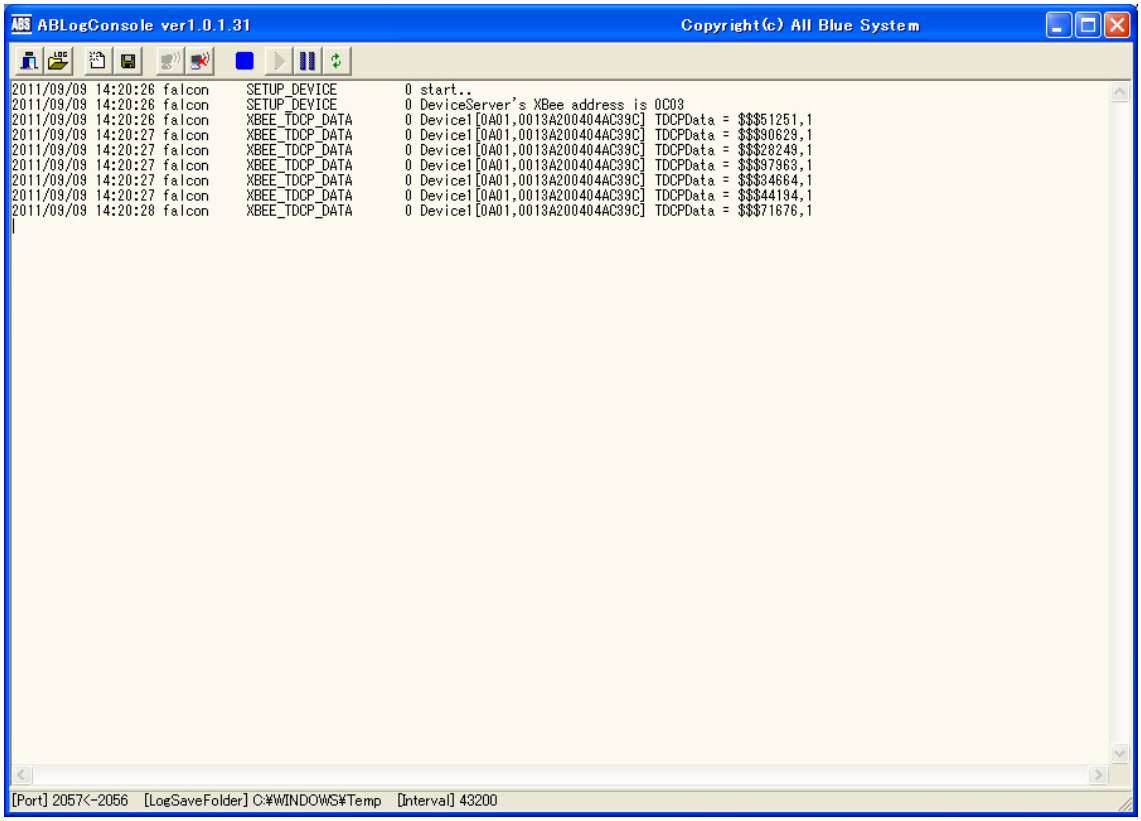
`error()` は、スクリプト実行中にエラーを検出した場合などに、スクリプトの実行を中止するときにコールする関数です。`error()` 関数を実行するとサーバーのログ中にエラー発生が記録されます。もしエラー状態にしないでスクリプトを途中で終了したい場合には、`error()` の代わりに `do return end;` の様に記述します。

ファイル名(SETUP_DEVICE.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR” に保管します。

スクリプト実行は、Web の “ScriptControl プログラム” または DeviceServerのクライアントプログラムから実行します。下記は、DeviceServer のクライアントプログラムから実行した画面例です。

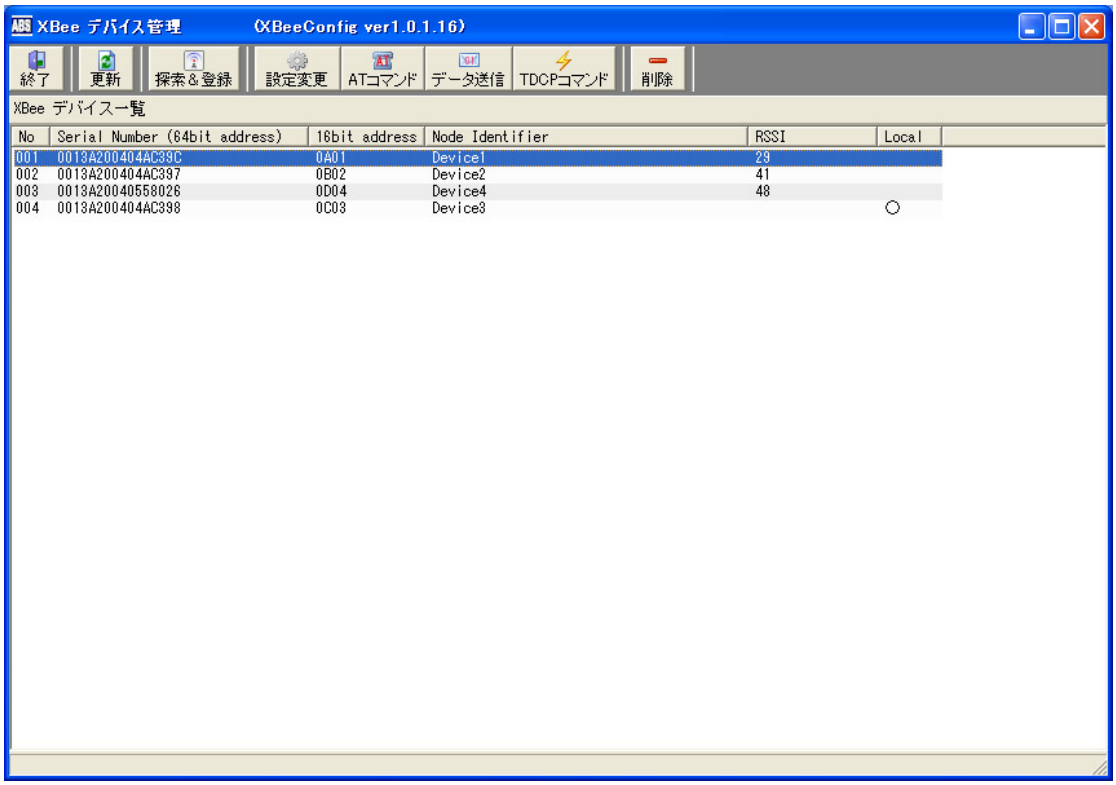


“実行”ボタンを押すとスクリプトがサーバーで実行され、リモートデバイスの TDCP プログラムの設定が行われます。ログには、下記の様なコマンド実行ごとのリモートデバイスからの応答パケット受信が記録されます。”$$$<文字列>” の後のカンマに続けて “1” が返っていればコマンド実行が成功したことを示しています。

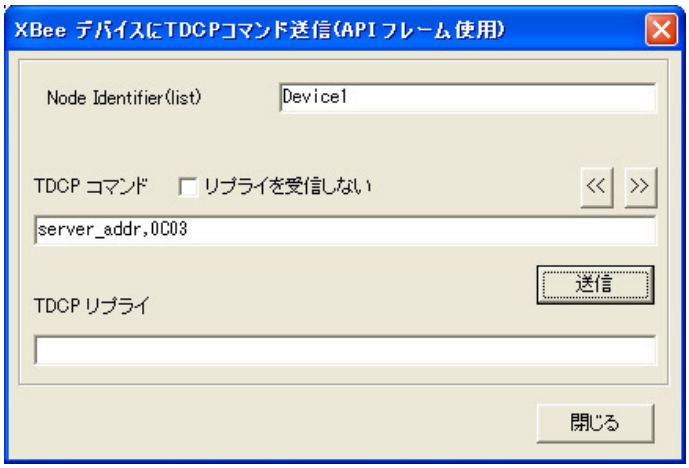


4.5.2 XBee デバイス管理プログラムから TDCP コマンド実行

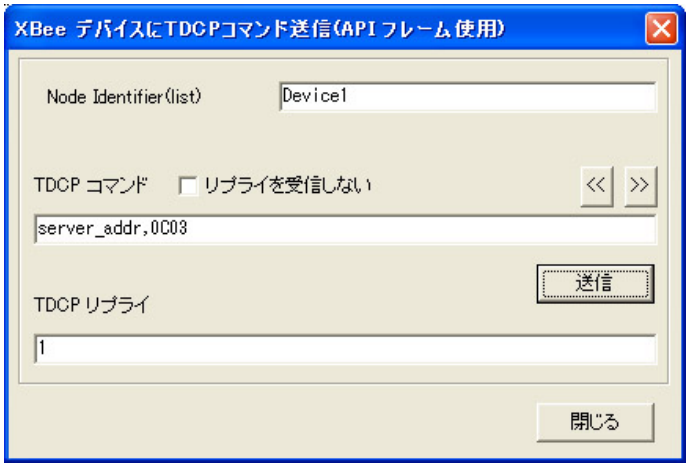
DeviceServerのクライアントプログラムを起動して、XBee デバイス管理プログラムを開きます。



“Device1” を選択して、”TDCPコマンド” ボタンを押します。下記のダイアログが表示されて、リモートのXBee デバイスとそれに接続された TDCP ファームウエアが動作しているCPU ボードに TDCP コマンドを送信できます。コマンド実行結果は、TDCP リプライ エディットボックスに返ります。



前述の 設定用 TDCP コマンドを入力して “送信ボタン”を押します。



TDCPリプライ文字列の最初のカラムに “1” が返った場合にはコマンド実行が成功したことを示します。コマンド実行が失敗した場合には “0” が返ります。ただし、“reset” コマンド実行時にはリプライパケットは返りませんのでコマンド送信時に “リプライを受信しない” にチェックを付けて下さい。

すべての 設定用 TDCP コマンドを同様に実行して下さい。“Device1” の設定が終了したら”閉じる” ボタンでデバイスリスト画面に戻ります。

5 スクリプト作成(温度データとリードスイッチ監視)

システム全体の動作をコントロールするスクリプトを記述します。

この章では最初に、温度データの取得とリードスイッチの監視を行う機能についてのみセットアップを行います。
後の章で自動サンプリング機能を追加して、温度グラフを表示するための機能を説明します。

⚠ 注意

スクリプト中に日本語を記述するときは、スクリプトファイルを UTF-8N 形式で保存してください。Shift_JISや UTF-8 BOM付き形式などで保存すると、DeviceServer でエラーが発生します。Windows付属のワードパッドやメモ帳ではこの形式で保存できませんので、別途 UTF-8N 形式で保存可能なエディタソフト（*1）を使用してください。
（*1）TeraPad などのソフトウエアがよく使用されています。

5.1 SERVER_START スクリプト作成

今回のアプリケーションでは、クライアントPCの Web ブラウザからDeviceServer をアクセスする場合にユーザー認証を省略しています。Webブラウザで表示するページ中の JavaScript からは、予め DeviceServer に作成されたセッション情報を利用することで簡単な仕組みにしています。

SERVER_START スクリプトに、セッション情報を自動的に作成するための記述を行います。
SERVER_START スクリプトは、DeviceServer 起動時に最初に１回だけ実行されるスクリプトで、ここにセッションを作成するための動作を記述します。 Webページ上の JavaScript で使用するセッション情報はここで作成したセッションと同一の文字列(セッショントークン)を指定してください。

```
file_id = "SERVER_START"
log_msg("start..",file_id)
-----------------------------------------------------
-- 認証を省略してアクセスするためのセッションを作成する
-----------------------------------------------------
if not create_session("1234",true) then error() end
```

`create_session("1234",true)` 関数で、サーバーにセッションを作成します。第一パラメータはセッショントークン文字列で、英数字を使用した任意の文字列を指定してください。通常は、ログイン操作でユーザー認証を行った後、自動で生成されたユニークな文字列をセッショントークンとして使用しています。ここではWeb API 経由でアクセスするときにユーザー認証を省略できるように、強制的にセッショントークン文字列を指定してセッションを作成しています。`create_session()` 関数の第二パラメータに True を指定することで、セッションを使用しない場合の自動ログアウト（自動セッション削除）の対象から外すことができます。

ここで作成したセッショントークンと同じ文字列を、Web API を使用して JavaScript 等からアクセスするときに指定します。

ファイル名(SERVER_START.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

5.2 XBEE_TDCP_DATA スクリプト作成

サーバーPC でリモートデバイスからイベントデータ、リクエスト応答パケットなどを受信したときに実行されるスクリプトを作成します。

DeviceServer をインストールしたときに、初期ファイルとして XBee データパケット内容をログに出力する機能のみが記述されていますので、これに追加記述します。

リモートデバイスのリードスイッチが反応して、I/O 値が変化したときに送信される “CANGE_DETECT” イベントデータを受信したときにこのスクリプトが実行されます。(他の種類のTDCPイベントデータを受信した場合にも、XBEE_TDCP_DATA スクリプトがコールされます)

イベントデータの種類が “CANGE_DETECT” であることを確認した後、共有データ(データベース)をチェックして、現在の監視モードが “ON” になっているかどうかをチェックします。

監視モードが “ON” の場合だけ､警報メール送信のためのスクリプト “ARDUINO_SENSOR/ALARM_MAIL” をコールします。

```
file_id = "XBEE_TDCP_DATA"
--[[
*******************************************************************************
* イベントハンドラスクリプト実行時間について                                  *
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
XBEE_TDCP_DATA スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値              値                                          値の例
-------------------------------------------------------------------------------
APIType             フレームデータ中のAPI Type(16進数2桁)       81
SourceAddress       フレームデータ中のSourceAddress
                    16bit アドレスの場合(16進数4桁)             0A01
                    64bit アドレスの場合(16進数16桁)            0013A200404AC397
SerialNumber        XBee デバイスの SerialNumber
```

```
                    DeviceServer に保持されたマスターファイルを使用して、
                    SourceAddress から変換した値が設定される。          0013A200404AC397
NodeIdentifier      XBee デバイスの NodeIdentifier。
                    DeviceServer に保持されたマスターファイルを使用して、
                    SourceAddress から変換した値が設定される。          Device1

RSSI                フレームデータ中のRSSI(16進数2桁)                  45
Options             フレームデータ中Options                             00
TDCP_COUNT          TDCP データカラム数                                 2
TDCP_<Column#>      TDCP データ値(ASCII 文字列)
                    TDCP_1 は常にコマンドプリフィックス文字列を表す     "$$$1234"
                    "$$$" で始まり、0文字以上の任意の文字列が後に続く。
                    TDCP_2 はコマンド実行ステータスを表す               "1"
                    "1" はコマンド実行成功、"0" は失敗を示す
                    イベントデータの場合にはイベント名が入る
                    TDCP_3以降のデータはTDCPコマンド毎に決められた、
                    オプション文字列が入る
                    <Column#> には 最大、TDCP_COUNT まで 1から順番に
                    インクリメントされた値が入る。
]]
------------------
-- BEGIN SCRIPT --
------------------


--------------------------------------------------------------------------------
-- リモートデバイスのリードスイッチが変化したときにメールを送信する
--------------------------------------------------------------------------------
if (g_params["TDCP_2"] == "CHANGE_DETECT") and
   (g_params["TDCP_4"] == "31") and
   (bit_and(tonumber(g_params["TDCP_5"],16),0x01) > 0 ) then


          ----------------------------------------
          -- 現在監視中かどうかをチェックする
          ----------------------------------------
          local stat,flag = get_permanent_data("WatchMode")
          if not stat then error() end
          if (flag == "1") then


                    ------------------------------------------------
```

```
                    -- アラーム送信する
                    ------------------------------------------------
                    if not script_fork_exec("ARDUINO_SENSOR/ALARM_MAIL","","") then error() end
          end;
end


------------------
-- END SCRIPT   --
------------------
```

このスクリプトは、リモートデバイスで監視対象の I/O ポートのビット値が変化したときに実行されます。
スクリプトパラメータ　g_params[] 部分にイベントデータが格納されています。

TDCP デバイスイベントデータ(CHANGE_DETECT) の内容は下記の様になっています(TDCP for Atmega328ユーザーマニュアルより抜粋)

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

<my_addr16>　TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。
<app_mode>　TDCP_328用の固定値 "31" 設定されます。
<diff_bits>　変化したビットを 1、変化していないビットを 0にした値が、8bit幅の16進数表記で設定されます。
<dio>　　現在のポート値が 8bit 幅の16進数表記で設定されます。

上記のカンマ区切りの最初のデータ "$$$" は、イベントハンドラでは `g_params["TDCP_1"]` に格納されいます。後は順に "CHANGE_DETECT" が `g_params["TDCP_2"]` の様に格納されています。

`if g_params["TDCP_2"] == "CHANGE_DETECT" then` では、このイベントハンドラスクリプトが実行されたときに、リモートデバイスでイベントが発生していたかを調べています。イベントハンドラでは、イベントごとにあからじめ決められたイベントパラメータが `g_params[]` 配列(文字列をキーとした連想配列)に格納されています。
`g_params["TDCP_2"]` は、TDCP デバイスイベントの２つめのイベントデータを指定していて、イベントの種類を表します。ポートの値が変化した時にはイベント種類は "CHANGE_DETECT" になります。その他にもTDCP には A/D 変換値が決められた範囲よりも変化した時のイベント "ADVAL_UPDATE" などがあります。イベントハンドラスクリプトに渡されるパラメータの詳細は、"TDCP for ATmega328 ユーザーマニュアル"
(http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf) のイベントリファレンスの章を参照してください。

`(bit_and(tonumber(g_params["TDCP_5"],16),0x01) > 0 )` は、ポートのビット#0 が変化しているかどうかをチェックしています。`g_params["TDCP_5"]` には、CHANGE_DETECT イベント発生時に変化しているポートのビット位置が16進数文字列で格納されています。例えば `g_params["TDCP_5"]` の値が "FF" の場合には全てのポートのビットが変化していることを示します。`tonumber(g_params["TDCP_5"],16)` で、この16進数の文字列を数値に変換します。

(bit_and(tonumber(g_params["TDCP_5"],16),0x01) > 0)で算術 AND 演算を行って、ビット位置#0 が変化しているかどうかを、0 またはそれ以外の値になることで判断しています。

local stat,flag = get_permanent_data("WatchMode") はリードスイッチが反応したときに、監視モードが “ON” になっているかどうかを調べています。現在の監視モードを保存するために、キー名 “WatchMode” で データベース中に “1” または “0” の値を保存しています。後述の “ARDUINO_SENSOR/WATCH_MODE” スクリプトでこの監視モード値を更新しています。ここでは、この値をデータベースから取得しています。

if not script_fork_exec("ARDUINO_SENSOR/ALARM_MAIL","","") then error() end は、警報メールを送信するためのスクリプト “ARDUINO_SERVER/ALARM_MAIL” をコールしています。このとき、スクリプトの実行（メール送信）完了を待たずにこの XBEE_EVENT_DATA スクリプトと並行して実行するように script_fork_exec() を使用しています。イベントハンドラは頻繁にコールされるため、スクリプト実行完了までの時間をできるだけ短くするようにします。この様な場合に、実行時間がかかりそうなスクリプトを別スレッドで実行させることができます。
もし XBEE_EVENT_DATA 処理に時間がかかって、同一のイベントが連続して発生した場合でも、XBEE_EVENT_DATA イベントハンドラスクリプト自身が並行して複数同時に実行されますので、通常は問題ありません。

ファイル名(XBEE_TDCP_DATA.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

5.3 ARDUINO_SENSOR/ALARM_MAIL スクリプト作成

警報メールを送信するためのスクリプトを作成します。

メールの宛先を設定している下記の部分は、警報メールの送信先メールアドレスに変更して下さい。
local mail_addr = "監視警報メール宛先 <your_mail_address@your.mail.domain>"

```
file_id = "ALARM_MAIL"
--[[
*********************************************************************
監視中にセンサーが反応したのでアラームメールを送信する
*********************************************************************
]]
log_msg("start..",file_id)
-------------------------------------------------------
-- アラームメールの宛て先を設定する
-------------------------------------------------------
local mail_addr = "監視警報メール宛先 <your_mail_address@your.mail.domain>";


-------------------------------
```

```
-- クリティカルセッション開始
--------------------------------

local cstat,handle = critical_session_enter("LapseTimeCheck",10000);
if not cstat then error() end
------------------------------------------------------------
-- 前回アラーム出力を行ったときからの経過時間を計測する
------------------------------------------------------------
local t = os.time();

local stat,prev_t = get_shared_data("PREV_ALARM_TIME")
if not stat then error() end
if (prev_t ~= "") then
        ------------------------------------------------------------
        -- 前回のアラーム出力から 60 秒以内の場合には送信しない
        ------------------------------------------------------------
        local diff_t = os.difftime(t,tonumber(prev_t));
        if diff_t < 60 then
                log_msg("連続送信キャンセル",file_id)
                critical_session_leave(handle)
                return;
        end;
end;

------------------------------------------------------------
-- 次回アラーム出力時の経過時間計測用に現在時刻を保存する
------------------------------------------------------------
if not set_shared_data("PREV_ALARM_TIME",tostring(t)) then error() end

--------------------------------
-- クリティカルセッション終了
--------------------------------
if not critical_session_leave(handle) then error() end

------------------------------
-- アラームメール送信
------------------------------
local body = {};
table.insert(body,"リードスイッチの変化を検出しました")
```

```
if not mail_send(mail_addr,"","** アラームメール送信 **",unpack(body)) then error() end


------------------------------
-- アラームログ出力
------------------------------
for key,val in ipairs(body) do
          log_msg((val),file_id)
end
```

最初の部分は、警報メールを連続して短時間に複数回送信しないようにするための機能です。

最後にメールを送信した時刻を共有変数 “PREV_ALARM_TIME” に常に保存しています。メール送信前にこの値をチェックして、現在時刻が “PREV_ALARM_TIME” で示された時刻よりも１分以上経過している場合だけ、警報メール送信を行うようにします。

“PREV_ALARM_TIME” 共有変数は、`local stat,prev_t = get_shared_data("PREV_ALARM_TIME")` で取得されて、
`local diff_t = os.difftime(t,tonumber(prev_t));`
`if diff_t < 60 then` の部分で経過時間をチェックしています。
メール送信時には、`if not set_shared_data("PREV_ALARM_TIME",tostring(t)) then error() end` 部分で現在時刻を“PREV_ALARM_TIME” 共有変数に設定して、次のメール送信要求時に備えます。

このスクリプト自身は、リモートデバイスの I/O が変化したときにコールされますが、１秒間に数十回程度リードスイッチが反応するとこのスクリプトが並行して複数実行されることになります。(10ms よりも早くスイッチが反応した場合には、チャタリング防止のためのTDCPデバイス自身のフィルタ処理によってイベントは発生しません)
このとき、“PREV_ALARM_TIME” 共有変数 のチェックと更新操作を行う部分を、他のスレッド間と排他制御して動作を確実にする必要があります。

`local cstat,handle = critical_session_enter("LapseTimeCheck",10000);` は、排他制御をするために “LapseTimeCheck” という名前でクリティカルセッションを開始します。パラメータで指定する名前は別の名前を指定しても構いません。10000 は、クリティカルセッションに 10秒(10000ms)以内に入れなかった場合にエラーを検出するためのタイムアウト時間です。この関数の実行が成功した場合には、これ以降 `critical_session_leave()` 関数をコールするまで、同じ名前のパラメータ“LapseTimeCheck” を指定してクリティカルセッション開始を待つスクリプトが、並行して実行されることはありません。

```
------------------------------
-- クリティカルセッション終了
------------------------------
if not critical_session_leave(handle) then error() end
```

の部分で、排他制御を終了してクリティカルセッションを抜けます。

critical_session_enter()をコールしてから、critical_session_leave() を実行するまでの間に、何らかのエラー条件を検出してスクリプトの動作を中止する場合にも、必ずcritical_session_leave() を実行してから error() をコールしてください。現在のバージョンの Lua 言語には “try-finally” 文が用意されていませんので、ユーザー側でクリティカルセッション内から抜け出る全ての箇所に、critical_session_leave()を配置してください。

自動的にクリティカルセッションのミューテックスハンドルが開放されます

スクリプト中で critical_session_enter()をコールしてから、スクリプトを終了（エラーによって強制的に終了する場合を含む）するまでの間に critical_session_leave()をコールしなかった場合には、Windows のリソースを無駄に消費するのを防ぐために、DeviceServer は強制的にcritical_session_leave()をコールして、ログに警告メッセージを出力します。

if not mail_send(mail_addr,"","** アラームメール送信 **",unpack(body)) then error() end は、body テーブル変数に入った文字配列をメールで送信しています。

ファイル名(ALARM_MAIL.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

5.4 ARDUINO_SENSOR/GET_DATA スクリプト作成

温度表示用の Web ページに記述された JavaScript からコールされて、現在のリモートデバイスの温度を取得するためのスクリプトを作成します。

```
file_id = "GET_DATA"
--[[
*********************************************************************************
現在の温度と監視モードを取得する
スクリプトリターンパラメータ
          "Temperature": 現在の温度が文字列形式で入る 例: "23.334"
          "WatchMode": 監視モード   "0" 監視外, "1" 監視中
*********************************************************************************
]]

local device_name = "Device1"
local adc_vref = 1.1
local temp

--[[
---------------------------------------------------------------------------------
リモートデバイスで "force_sample" コマンドを実行して、現在の ADC 値を取得する
```

```
force_sample リプライ (詳しくは "TDCP for ATmega328 ユーザーマニュアル" を参照の事)

          result[1] : TDCP reply prefix
          result[2] : command result status "0":Fail "1":success
          result[3] : app_mode "0".."10":TDCP  "31": TDCP for ATmega328
          result[4] : DIO
          result[5] : freq
          result[6] : counter
          result[7] : A/D #0
          result[8] : A/D #1
          result[9] : A/D #2
          result[10]: A/D #3
-------------------------------------------------------------------------------------------
]]
local stat,result = xbee_tdcp_safe_retry(device_name,"force_sample")
if not stat then error() end

-----------------------------------------------
-- TDCP コマンド実行結果が成功したかどうかと、
-- 対象デバイスが TDCP_328 であることを確認
-----------------------------------------------
if (result[2] == "1") and (result[3] == "31") then
          ----------------------------------------
          -- 温度計算  LM35 出力(A/D#0) : 10mv/℃
          ----------------------------------------
          temp = tostring(100 * adc_vref * tonumber(result[7]) / 1024);
          if not script_result(g_taskid,"Temperature",temp) then error() end;
else
          error()
end;

--[[
-------------------------------------------------------------------------------------------
現在の監視モードを取得する
共有データ Key       :"WatchMode"
           Value     :"1" 監視中
                     :"0" またはデータ無し          監視外
-------------------------------------------------------------------------------------------
```

```
]]


local value
stat,value = get_permanent_data("WatchMode")
if not stat then error() end
if value == "" then value = "0" end
if not script_result(g_taskid,"WatchMode",value) then error() end;


-- log_msg("現在のリモート温度 = " .. temp .. " 監視モード = " .. value,file_id)
```

`local adc_vref = 1.1` は、A/D 変換値から温度を計算するときに使用するローカル変数の宣言です。Atmega328P の A/D リファレンス電圧を内部の 1.1V を使用するためにこの値になっています。TDCPデバイスの内部リファレンス電圧の設定は、前述の “ARDUINO_SENSOR/SETUP_DEVICE” スクリプト中で指定した `stat,result = xbee_tdcp_command(device,"adc_vref,3")` で行っています。

`local stat,result = xbee_tdcp_safe_retry(device_name,"force_sample")` は、リモートデバイスに対してマニュアルサンプリングを実行して、現在の I/O ポート値、A/D 変換値を取得しています。コマンド実行に成功すると、`result` 配列変数にサンプリング結果が入ってきます。サンプリングデータの詳細は上記スクリプト中のコメント行または、”TDCP for Atmega328 ユーザーマニュアル” を参照して下さい。このとき、`result[2]` にはリモートコマンド“force_sample” の実行結果ステータスが入っていて、値が “1” の時にコマンド実行が成功しています。`result[7]` には 温度センサーが接続された A/D チャンネル#0 の値が１０進数で入ります。

`temp = tostring(100 * adc_vref * tonumber(result[7]) / 1024)` は、摂氏温度を計算しています。使用している温度センサー(LM35D)の仕様(1℃毎の出力変化=10mv)と、A/D リファレンス電圧を元に温度を計算しています。

`if not script_result(g_taskid,"Temperature",temp) then error() end;` は、計算した摂氏温度をスクリプトリターンパラメータに設定しています。キー名 “Temperature” に対応する値は　温度データを文字列形式にしたものになります。JavaScript からこのスクリプトのリターンパラメータを取得するときに、キー名 “Temperature”を指定して温度データを得ます。

`stat,value = get_permanent_data("WatchMode")` は、現在の監視モードを取得しています。
`if not script_result(g_taskid,"WatchMode",value) then error() end;` は、取得した監視モードをスクリプトリターンパラメータに設定しています。キー名 “WatchMode” に対応する値は　現在の監視モード(“1” または “0”)になります。JavaScript からこのスクリプトのリターンパラメータを取得するときに、このキー名を指定して監視モードを得ます。

ファイル名(GET_DATA.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

5.5 ARDUINO_SENSOR/WATCH_MODE スクリプト作成

温度表示用の Web ページに記述された JavaScript から監視モードチェックボックスを操作された時にコールされて、現在の監視モードを変更するためのスクリプトを作成します。

```
file_id = "WATCH_MODE"
--[[
*******************************************************************************
監視モードを更新する。
リモートデバイスのLED を、現在の監視モードに応じて点灯または消灯状態にする
スクリプトパラメータ
          "WatchMode": 監視モード   "0" 監視外, "1" 監視中
*******************************************************************************
]]

local device_name = "Device1"

--[[
-------------------------------------------------------------------------------
現在の監視モードを設定する
共有データ Key       :"WatchMode"
           Value     :"1" 監視中
                     :"0" またはデータ無し          監視外
-------------------------------------------------------------------------------
]]
if (g_params["WatchMode"] == "1") or (g_params["WatchMode"] == "0") then
          if not set_permanent_data("WatchMode",g_params["WatchMode"]) then error() end
else
          log_msg("パラメータが間違っています",file_id)
          return
end;

-------------------------------------------------------------------------
-- リモートデバイスの DIO#1 を監視モードに応じて High または Low に設定する
-------------------------------------------------------------------------
if not xbee_tdcp_safe_retry(device_name,"port_bit,1," .. g_params["WatchMode"]) then error() end

-----------------------------------------------------
-- 連続メール送信防止用のフラグをクリアする
-----------------------------------------------------
```

```
if not set_shared_data("PREV_ALARM_TIME","") then error() end
```

`if (g_params["WatchMode"] == "1") or (g_params["WatchMode"] == "0") then` は、JavaScript のスクリプトパラメータで、キー名 “WatchMode” とその値が正しく指定されているかどうかを調べています。

`if not set_permanent_data("WatchMode",g_params["WatchMode"]) then error() end` は、Webページのチェックボックスの状態によって指定された監視モードを設定しています。

`if not xbee_tdcp_safe_retry(device_name,"port_bit,1," .. g_params["WatchMode"]) then error() end`
は、監視モードの値 (“1” または ”0”) に応じて、リモートデバイスの LED を ON または “OFF” にするために I/O ポートを操作しています。”port_bit” コマンドをリモートデバイス上で実行して I/O bit1 の値を更新しています。

`if not set_shared_data("PREV_ALARM_TIME","") then error() end` は、警報メールを連続して送信しないようにチェックするための内部フラグをクリアしています。これによって、監視モード変更直後には何時でも警報メールが送信可能になるようにしています。

ファイル名(WATCH_MODE.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

6 Web クライアントプログラム設定

インターネットまたはLAN 上の Webブラウザから、現在のリモートデバイスの温度を表示するために設置する Webページの説明をします。

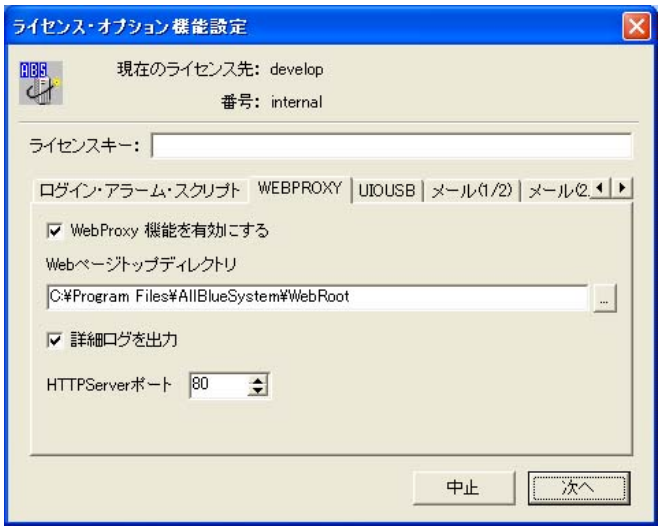
6.1 DeviceServer の WebProxy セットアップ

DeviceServer には HTTP サーバー機能と、Webブラウザ中で動作する JavaScript やFlash アプリケーションからコマンド操作をするためのWeb API の機能を持った WebProxy があります。DeviceServer インストール時に WebProxy のセットアップを行っていない場合は、下記の手順で設定を行います。

最初に、DeviceServer が動作している PCで既に HTTP サーバープログラムが動作していないことを確認してください。ポート番号80(http) でWebProxy が動作しますので、マイクロソフト社製 HTTPサーバー(IIS)や、Apache などのHTTPサーバープログラムを既に使用している場合には、WebProxy で設定するポート番号を変更してください。

DeviceServer と Webサーバープログラムの動作する PC を分離して設置することもできます。

サーバー設定プログラム(ServerInit.exe) プログラムをメニューから選択して実行します。”WEBPROXY” 設定タブの内容を下記のようにして、”次へ” ボタンを続けて押していってサーバーの設定を完了してください。DeviceServer が再起動して WebProxy 機能が有効になります。



6.2 温度表示用 Web ページ設定

WebProxy 設定で指定した “Webページトップディレクトリ” 項目が DeviceServer の動作するPC に Webブラウザでアクセスした時のルートディレクトリになります。

デフォルト設定では DeviceServer が動作しているPC の Webブラウザから ”http://localhost/index.html” をアクセスすると、”C:¥Program Files¥AllBlueSystem¥WebRoot¥index.html” ファイルにアクセスすることになります。”HTTPServerポート” を 8080 に設定した場合には、Webブラウザの URL は ”http://localhost:8080/index.html” を指定します。

ここでは、Webページトップディレクトリ (”C:¥Program Files¥AllBlueSystem¥WebRoot”) 以下に arduino_sensor フォルダを作成して、温度表示用 HTML ファイル (”C:¥Program Files¥AllBlueSystem¥WebRoot¥arduino_sensor¥index.html”) を作成します。また、index.html から参照される JavaScript ファイル device_server.js も同一のディレクトリに配置します。

DeviceServer 自身の HTTPサーバーのWebページトップディレクトリ以下にファイルを配置しないで、別の HTTP サーバーで管理しているフォルダにページを配置することもできます。このような HTTP サーバーが動作している PC と DeviceServer の PCを分けて設置する場合には、device_server.js ファイル内に記述されている DeviceServer をアクセスするホスト名指定を変更してください。具体的には DeviceServer の動作している PC のホスト名または IPアドレス(インターネット上に配置する場合にはドメイン名とホスト名)を device_server.js 中の “server_host_url” 変数に設定してください。

index.html ファイルの内容は下記の様になります。

```
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<title>リモートデバイス(Arduino+XBee)の温度とリードスイッチを監視</title>
<script type='text/javascript' src='https://www.google.com/jsapi'></script>
```

```
<script type='text/javascript' src='device_server.js'></script>
<script type='text/javascript'>

// 表示更新間隔(ms)
var updateInterval = 10000;
setInterval("on_timer()",updateInterval);



// Google Chart API 初期化
google.load('visualization', '1', {packages:['gauge']});
google.setOnLoadCallback(setupChart);

var data;
var chart;
var options;
var temperature = 0;

// 定期的に DeviceServerのスクリプトを実行して、最新のデバイスの温度と監視モードを取得する
function on_timer(){
        script_exec("ARDUINO_SENSOR/GET_DATA","update_sensor_data");
};

// 温度ゲージ表示
function setupChart() {
        data = new google.visualization.DataTable();
        data.addColumn('string', 'Label');
        data.addColumn('number', 'Value');
        data.addRows(1);
        data.setValue(0, 0, '温度');
        data.setValue(0, 1,temperature);
        chart = new google.visualization.Gauge(document.getElementById('chart_div'));
        options = {max:50,min:0,width: 200, height: 200, redFrom: 32, redTo: 50,
                                yellowFrom:25, yellowTo: 32,greenFrom:15, greenTo: 25,
                                majorTicks:["0","10","20","30","40","50"],
                                minorTicks: 5};
        //
        chart.draw(data, options);

        // 最新のデバイスの温度と監視モードを取得する
```

```
            script_exec("ARDUINO_SENSOR/GET_DATA","update_sensor_data");
}

// ARDUINO_SENSOR/GET_DATA スクリプトの終了時にコールされるイベントハンドラ
// リターンパラメータにリモートデバイスの温度と監視モードが入っている
function update_sensor_data(event){
            if (event.Result != "Success"){
                        log("script error!");
                        return;
            }

            // 小数点以下１位までに丸めて温度を表示
            temperature = Math.round(parseFloat(event.ResultParams.Temperature) * 10) / 10;
            data.setValue(0, 1,temperature);
            chart.draw(data, options);

            // 監視モードの状態をチェックボックスに反映させる
            mode_chk.checked = (event.ResultParams.WatchMode == "1");
}

// 監視モードチェックボックスが操作されたため、
// 共有データのフラグとリモートデバイスの LED を最新の状態に更新する
function update_watch_mode(){
            var params = new Object;
            params["WatchMode"] = mode_chk.checked ? "1" : "0";
            script_exec_params("ARDUINO_SENSOR/WATCH_MODE",params);
}
</script>
</head>
<body>
  <div id='chart_div' style='height: 200px; width: 200px;'></div>
  <p>
    <input id='mode_chk' type="checkbox" onclick="update_watch_mode()" >
              <label>リードスイッチ監視</label>
    </input>
  </p>
</body>
</html>
```

index.html 中から引用される device_server.js ファイルの内容は下記の様になります。

```
//サーバーホスト URL 設定// DeviceServer 自身の HTTP サーバーを使用する場合には空にする
//var server_host_url = "http://your_DeviceServer_public_url:80";
var server_host_url = "";

//
// DeviceServer設定
// SERVER_START スクリプトで作成した、認証省略時アクセス用のセッションをここで指定する

var session_token = "1234";



// DeviceServer の LogService にログメッセージを出力
function log(msg){
    var url =        server_host_url + "/command/log" +
                                       "?message=" + encodeURIComponent(msg) +
                                       "&module=" + encodeURIComponent("WebProxy(js)");
    var script  = document.createElement("script");
    script.type = "text/javascript";
    script.src  = url;
    document.body.appendChild(script);
}

// DeviceServer のスクリプトを実行する
// callback パラメータを省略するとリターンパラメータを受け取らない
function script_exec(name,callback){
          if (callback == undefined){
          var url =  server_host_url + "/command/json/script" +
                                       "?session=" + encodeURIComponent(session_token) +
                                       "&resultrecords=0" +
                                       "&callback=" + encodeURIComponent("default_callback") +
                                       "&name=" + encodeURIComponent(name);
          } else {
          var url =  server_host_url + "/command/json/script" +
                                       "?session=" + encodeURIComponent(session_token) +
                                       "&callback=" + encodeURIComponent(callback) +
                                       "&name=" + encodeURIComponent(name);
          }
```

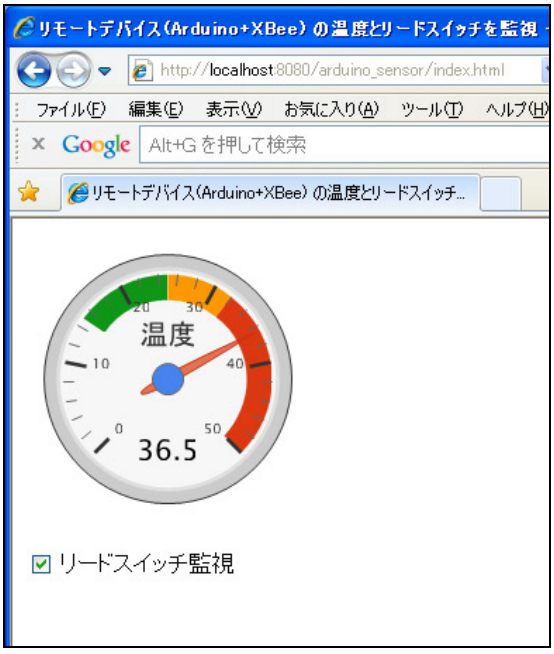
```
    var script  = document.createElement("script");
    script.type = "text/javascript";
    script.src  = url;
    document.body.appendChild(script);
}

// DeviceServer のスクリプトを実行する。
// 第２パラメータにスクリプトパラメータを指定する
// callback パラメータを省略するとリターンパラメータを受け取らない
function script_exec_params(name,params,callback){
          if (callback == undefined){
          var url =  server_host_url + "/command/json/script" +
                              "?session=" + encodeURIComponent(session_token) +
                              "&resultrecords=0" +
                              "&callback=" + encodeURIComponent("default_callback") +
                              "&name=" + encodeURIComponent(name);
          } else {
          var url =  server_host_url + "/command/json/script" +
                              "?session=" + encodeURIComponent(session_token) +
                              "&callback=" + encodeURIComponent(callback) +
                              "&name=" + encodeURIComponent(name);
          }

          for(key in params){
                    url = url + "&" + encodeURIComponent(key) + "=" + encodeURIComponent(params[key]);
          }

    var script  = document.createElement("script");
    script.type = "text/javascript";
    script.src  = url;
    document.body.appendChild(script);
}

// script_exec() でコールバックを指定しない場合のデフォルトコールバック関数
function default_callback(data){
          if (data.Result != "Success"){
                    log("script error!");
                    return;
          }
```

```
}


// 連想配列の要素数を取得
function array_len(arr){
        var idx = 0;
        for(k in arr){
                idx++;
        }
        return(idx);
}
```

7 アプリケーションを起動する

DeviceServer が動作しているPC もしくは、ネットワーク接続されたクライアントPC から Webブラウザを起動して、前項で設置した温度表示用 Webページをアクセスします。

例えば、WebProxy のポート番号が 8080 に設定されていた場合には、"http://localhost:8080/arduino_sensor/index.html" をアクセスします。



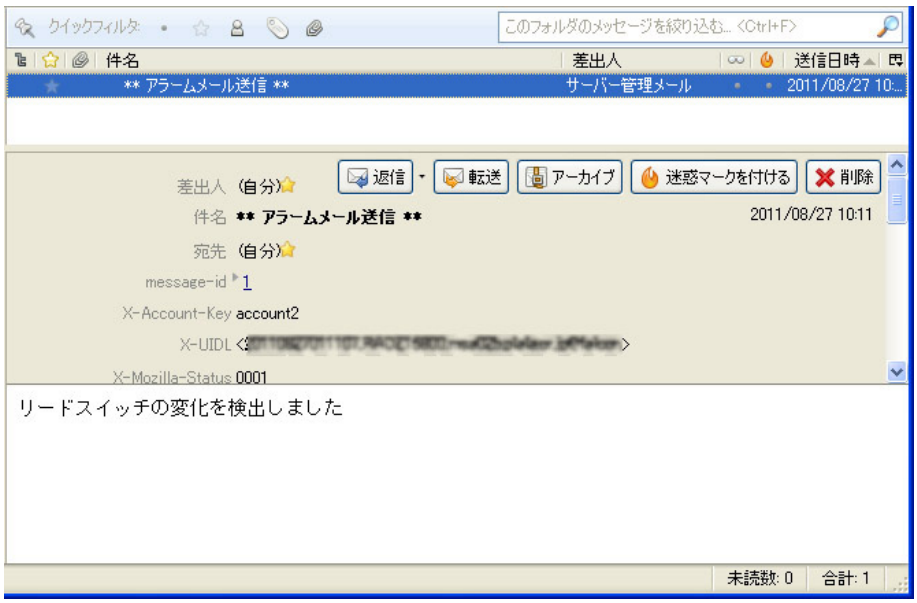
ページを開くと同時に、サーバー経由でリモートデバイスから最新の温度データを取得して温度計を表示します。このページを画面に表示しておくと、10 秒毎に１回の割合で自動的に最新のリモートデバイスの温度データを取得して、温度表示を更新するようになっています。リモートデータ取得の頻度は、index.html 中に記載された下記の JavaScript 部分で設定しています。

```
// 表示更新間隔(ms)
```

```
var updateInterval = 10000;
setInterval("on_timer()",updateInterval);
```

リードスイッチ監視チェックボックスは、現在の監視モードを表しています。チェックボックスを操作してチェックを付ける（監視モード “ON”）と、リモートデバイス上の LED が点灯します。同様にチェックボックスを外すと LED が消灯するのが確認できます。この監視モードは、DeviceServer で管理・保存していますので、サーバー再起動時でも最後に設定した監視モード状態を保持しています。

監視モードが “ON” の状態の時に、リードスイッチを反応させると下記の様な警報メールが送信されます。



8 温度グラフ表示用に機能を拡張する

温度グラフを表示するために、サーバーで動作するスクリプトとグラフ表示用の Web ページを追加します。
既存のスクリプト(温度データとリードスイッチ監視用に設置したスクリプト) を上書きして、機能を拡張します。

主に２つの機能を追加します。１つがリモートデバイスから定期的(10分毎) にサンプリングデータを自動送信してサーバー中のデータベースに保管する機能で、もう一つが 温度グラフを表示させるための JavaScriptで記述された Web ページです。

温度表示を行うために設定したスクリプトと HTML ファイルの中で、修正または追加が必要なものについて説明します。

8.1 XBEE_TDCP_DATA スクリプト修正

リモートデバイスから定期的にサンプリングデータを受信したときに実行される XBEE_TDCP_DATA イベントハンド

ラスクリプトに処理を追加します。

温度計表示の機能では、リードスイッチの変化時の処理のみでしたが、ここに自動サンプリング時に発生する
“SAMPLING” イベントの処理を追加します。リモートデバイスのコンフィギュレーション設定
(“ARDUINO_SENSOR/SETUP_DEVICE” スクリプト)を行ったときに、既に下記のTDCP 設定が実行されています。

```
stat,result = xbee_tdcp_command(device,"sampling_rate,600")
```

これは、10 分毎に “SAMPLING” イベントを発生するための設定ですが、このイベントが発生したときの処理はまだ
XBEE_TDCP_DATA イベントハンドラ中に記述されていなかったので、ここに処理を追加します。

```
file_id = "XBEE_TDCP_DATA"
--[[
*******************************************************************************
* イベントハンドラスクリプト実行時間について                                  *
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
XBEE_TDCP_DATA スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値              値                                              値の例
-------------------------------------------------------------------------------
APIType             フレームデータ中のAPI Type(16進数2桁)           81
SourceAddress       フレームデータ中のSourceAddress
                    16bit アドレスの場合(16進数4桁)                 0A01
                    64bit アドレスの場合(16進数16桁)                0013A200404AC397
SerialNumber        XBee デバイスの SerialNumber
                    DeviceServer に保持されたマスターファイルを使用して、
                    SourceAddress から変換した値が設定される。      0013A200404AC397
NodeIdentifier      XBee デバイスの NodeIdentifier。
                    DeviceServer に保持されたマスターファイルを使用して、
                    SourceAddress から変換した値が設定される。      Device1
```

```
RSSI                フレームデータ中のRSSI(16進数2桁)                    45
Options             フレームデータ中Options                              00
TDCP_COUNT          TDCP データカラム数                                  2
TDCP_<Column#>      TDCP データ値(ASCII 文字列)
                    TDCP_1 は常にコマンドプリフィックス文字列を表す      "$$$1234"
                    "$$$" で始まり、0文字以上の任意の文字列が後に続く。
                    TDCP_2 はコマンド実行ステータスを表す                "1"
                    "1" はコマンド実行成功、"0" は失敗を示す
                    イベントデータの場合にはイベント名が入る
                    TDCP_3以降のデータはTDCPコマンド毎に決められた、
                    オプション文字列が入る
                    <Column#> には 最大、TDCP_COUNT まで 1から順番に
                    インクリメントされた値が入る。
]]
------------------
-- BEGIN SCRIPT --
------------------
--------------------------------------------------------------------------------
-- リモートデバイスのリードスイッチが変化したときにメールを送信する
--------------------------------------------------------------------------------
if (g_params["TDCP_2"] == "CHANGE_DETECT") and
   (g_params["TDCP_4"] == "31") and
   (bit_and(tonumber(g_params["TDCP_5"],16),0x01) > 0 ) then
          ----------------------------------------
          -- 現在監視中かどうかをチェックする
          ----------------------------------------
          local stat,flag = get_permanent_data("WatchMode")
          if not stat then error() end
          if (flag == "1") then
                    ------------------------------------------------
                    -- アラーム送信する
                    ------------------------------------------------
                    if not script_fork_exec("ARDUINO_SENSOR/ALARM_MAIL","","") then error() end
          end;
end


--------------------------------------------------------------------------------
-- サンプリングイベントを受信した場合は統計用データベースに登録するための
-- スクリプトを実行する。
```

```
--------------------------------------------------------------------------
if g_params["TDCP_2"] == "SAMPLING" and g_params["TDCP_4"] == "31" then
          local stat2,addr16,serial,nodename = xbee_find_device(g_params["TDCP_3"]);
          local key_list = list_to_csv("RemoteDeviceName","TemperatureLevel")
          local val_list = list_to_csv(nodename,g_params["TDCP_8"])
          stat2 = script_fork_exec("ARDUINO_SENSOR/REGISTER_ACTIVITY",key_list,val_list)
          if not stat2 then error() end
end
```

TDCP デバイスイベントデータ(SAMPLING) の内容は下記の様になっています(TDCP for Atmega328ユーザーマニュアルより抜粋)

$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<freq>,<counter>,<adc0>,<adc1>,<adc2>,<adc3>

<my_addr16>　　TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode>　　TDCP_328用の固定値 “31” 設定されます。

<dio>　　現在のポート値が8bit 幅の16進数表記で設定されます。

<freq>　　DIO#3(PORTD bit5) のポート値が１秒間に何回変化したかを示す周波数値が、10 進数表記で設定されます。周波数値は 1 秒毎に更新されて”SAMPLING”イベント発生時には最新の値が入ります。カウント可能な周波数の最大値は 65535 (2^16 - 1)で、これ以上の値になった場合には オーバーフローを表す –1 が設定されます。

<counter>　カウント期間(前回の “SAMPLING” イベント発生後から今回の “SAMPLING” イベント発生までの間) に、DIO#3(PORTD bit5) のポート値が何回変化したかを示すカウンタ値が、10 進数表記で設定されます。カウント可能な最大数は 2147483647 (2^31 - 1)で、これ以上の値になった場合には オーバーフローを表す –1 が設定されます。また、カウント期間中に一度でも １秒間に 65535 回以上ポートが変化した場合には、同様にオーバーフローを表す –1 が設定されます。

<adc0>.. <adc3>　　A/D 入力変換値が 10 進数表記で設定されます。

上記のカンマ区切りの最初のデータ “$$$” は、イベントハンドラでは `g_params["TDCP_1"]` に格納されています。後は順に “SAMPLING” が `g_params["TDCP_2"]` の様に格納されています。

`if g_params["TDCP_2"] == "SAMPLING" and g_params["TDCP_4"] == "31" then`

では、このイベントハンドラスクリプトが実行されたときに、リモートデバイスでイベントが発生していたかを調べています。イベントハンドラでは、イベントごとにあからじめ決められたイベントパラメータが `g_params[]` 配列(文字列をキーとした連想配列)に格納されています。`g_params["TDCP_2"]` は、TDCP デバイスイベントの２つめのイベントデータを指定していて、イベントの種類を表します。自動サンプリングが実行されてときのイベント種類は “SAMPLING” になります。その他にもTDCP には A/D 変換値が決められた範囲よりも変化した時のイベント “ADVAL_UPDATE” などがあります。イベントハンドラスクリプトに渡されるパラメータの詳細は、”TDCP for ATmega328

ユーザーマニュアル”（ http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf ）のイベントリファレンスの章を参照してください。

`local stat2,addr16,serial,nodename = xbee_find_device(g_params["TDCP_3"]);` は、サンプリングイベントを送信してきた XBee デバイスのアドレスからデバイス名(Node Identifier) を取得する関数です。このアプリケーションノートの設定例では、`nodename` 変数に XBee デバイス名 “Device1” が入ります。

```
local key_list = list_to_csv("RemoteDeviceName","TemperatureLevel")
local val_list = list_to_csv(nodename,g_params["TDCP_8"])
stat2 = script_fork_exec("ARDUINO_SENSOR/REGISTER_ACTIVITY",key_list,val_list)
```

は、サンプリングデータ中の <adc0> (A/D チャンネル#0) の値とデバイス名をパラメータに指定して、スクリプト “ARDUINO_SENSOR/REGISTER_ACTVITY” を実行しています。スクリプトパラメータはキー名リスト(CSV形式) と値リスト(CSV 形式) を指定します。また、“ARDUINO_SENSOR/REGISTER_ACTVITY” スクリプトではデータベースへの登録を行うためのスクリプトの処理時間を考慮して、別スレッドで実行することで XBEE_TDCP_DATA イベントハンドラ自身は直ぐに処理が完了するようにしています。

ファイル名(XBEE_TDCP_DATA.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

8.2 ARDUINO_SENSOR/REGISTER_ACTIVITY スクリプト作成

サンプリングイベント受信時に、XBEE_TDCP_DATA スクリプトから実行されて、温度データを統計用のデータベースに登録するスクリプトを作成します。

```
file_id = "REGISTER_ACTIVITY"
--[[
REGISTER_ACTIVITY スクリプト起動時に渡されるパラメータ
--------------------------------------------------------------------------------
キー値              値                                      値の例
--------------------------------------------------------------------------------
RemoteDeviceName    リモートデバイス名                      Device1
TemperatureLevel    温度センサーの値を A/D 変換した値       230
--------------------------------------------------------------------------------
温度センサーAD値 統計用データベースキー名          "SENSOR_TP_" + <RemoteDeviceName>
統計用データベースに登録する値                     ActivityLevel を数値に変換した値
]]

local temperature;
local key,val,stat;
```

```
local adc_vref = 1.1;
------------------------------------------------------------
-- 温度計算
------------------------------------------------------------
if g_params["RemoteDeviceName"] and g_params["TemperatureLevel"] then
  temperature = (100 * adc_vref * tonumber(g_params["TemperatureLevel"])) / 1024;
  log_msg(string.format("%s temperature = %2.3g",g_params["RemoteDeviceName"],temperature),file_id)
else
  log_msg("*ERROR* parameter error",file_id);
  return;
end;


------------------------------------------------------------
-- サンプリング値をデータベースに登録
------------------------------------------------------------
key = "SENSOR_TP_" .. g_params["RemoteDeviceName"]
stat = add_stat_data(key,tostring(temperature))
if not stat then error() end
```

`if g_params["RemoteDeviceName"] and g_params["TemperatureLevel"] then` は、このスクリプトをコールするときに、必要なパラメータが渡されているかどうかをチェックしています。

`temperature = (100 * adc_vref * tonumber(g_params["TemperatureLevel"])) / 1024;` は、パラメータで渡された A/D 変換値（0 から 1023 の数値）から温度を計算しています。

```
key = "SENSOR_TP_" .. g_params["RemoteDeviceName"]
stat = add_stat_data(key,tostring(temperature))
```

この部分で、温度データをデータベースに格納しています。後でデータベースに格納されたデータを集計をするときに、対象データを絞り込むためのキーを指定しています。ここでは “SENSOR_TP_<Device名>” でリモートデバイス毎にキーを指定しています。このアプリケーションノートではリモートデバイスを１つだけ使用する例で説明していますが、複数のリモートデバイスを使用する場合に備えて、ここでデバイス毎にキーを分けておきます。

ファイル名(REGISTER_ACTIVITY.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

8.3 PERIODIC_TIMER スクリプト作成

PERIODIC_TIMER スクリプトは、DeviceServer で１分ごとに起動されます。

このスクリプト中から、統計データベース中の過去のデータを消去するためのスクリプトをコールします。

共有変数 “TIME_1H” は、1時間(60分)をカウントするために使用しています。（別の名前を使用しても構いません）

```
file_id = "PERIODIC_TIMER"
-------------------------------------------------------------
-- 1時間に一回 SENSOR_DATA_PURGE スクリプトを実行する
-------------------------------------------------------------
stat,val = inc_shared_data("TIME_1H")
if not stat then error() end
if (tonumber(val) == 1) then
          if not script_fork_exec("ARDUINO_SENSOR/SENSOR_DATA_PURGE","","") then error() end
end
if (tonumber(val) > 60) then
          stat = set_shared_data("TIME_1H","")
          if not stat then error() end
end
```

```
stat,val = inc_shared_data("TIME_1H")
if not stat then error() end
if (tonumber(val) == 1) then
```

部分は、一時間毎に処理を行うための判断文です。後の部分の

```
if (tonumber(val) > 60) then
    stat = set_shared_data("TIME_1H","")
    if not stat then error() end
end
```

と合わせて、１時間を計測しています。

```
if not script_fork_exec("ARDUINO_SENSOR/SENSOR_DATA_PURGE","","") then error() end
```

過去のデータを削除するためのスクリプトを別スレッドで実行しています。

ファイル名(PERIODIC_TIMER.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

8.4 ARDUINO_SENSOR/SENSOR_DATA_PURGE スクリプト作成

このアプリケーションノートで説明している温度グラフでは、当日分のグラフのみを表示していますが、HTML(JavaScript) を変更して指定日のグラフを表示するように改造することも簡単にできます。そのときに過去のデータが必要になる場合に備えて過去 1ヶ月分のデータを保持しておきます。

このスクリプトでは、実行した当日から過去30 日以前のデータを統計データベースから削除します。

```
file_id = "SENSOR_DATA_PURGE"
------------------
-- BEGIN SCRIPT --
------------------
-- 30 日以前のデータを削除する
log_msg("start..",file_id)
local now = os.date "*t"
local stat,yyyy,mm,dd = inc_day(-30,now["year"],now["month"],now["day"])
local timestamp = string.format("%4.4d/%2.2d/%2.2d 23:59:59",yyyy,mm,dd)
if not clear_stat_data("SENSOR_TP_",timestamp,"") then error() end
------------------
-- END SCRIPT   --
------------------
```

`local now = os.date "*t"`

`local stat,yyyy,mm,dd = inc_day(-30,now["year"],now["month"],now["day"])`

部分では、yyyy,mm,dd 変数に30 日前の日付（年、月、日）を取得しています。

`if not clear_stat_data("SENSOR_TP_",timestamp,"") then error() end` は、キー名 “SENSOR_TP_” で始まるデータで、過去 30 日以前に登録されたものを全て削除しています。

ファイル名(SENSOR_DATA_PURGE.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

8.5 ARDUINO_SENSOR/SUMMARY_DATA スクリプト作成

温度グラフ表示用の Webページ(JavaScript) から実行されて、データベースに格納されている当日の温度データを１５分毎に集計して結果を返します。

```
file_id = "SUMMARY_DATA"
--[[
*********************************************************************************

温度データを集計する
スクリプトを起動した日を対象に、30 分毎の温度平均を計算する。
サンプリングデータが見つからない場合には、データ値を文字列 "NULL" として返す

*********************************************************************************
]]
```

```
local device_name = "Device1"
local now = os.date "*t"
local summary_from = string.format("%4.4d/%2.2d/%2.2d 0:0:0",now["year"],now["month"],now["day"])

-- 30 分単位で集計
-- stat,datetime,sample,total,mean,max,min = summary_stat_data("SENSOR_TP_" ..
device_name,summary_from,1800,48)
-- 15 分単位で集計
stat,datetime,sample,total,mean,max,min = summary_stat_data("SENSOR_TP_" ..
device_name,summary_from,900,96)
if not stat then error() end

local temp
for key,val in ipairs(datetime) do
        if sample[key] > 0 then
                temp = string.format("%2.3g",mean[key])
        else
                temp = "NULL"
        end
        if not script_result(g_taskid,string.sub(val,12,16),temp) then error() end
end
```

`local now = os.date "*t"`
`local summary_from = string.format("%4.4d/%2.2d/%2.2d 0:0:0",now["year"],now["month"],now["day"])`
は、集計用のライブラリ関数 `summary_stat_data()` で使用する “集計対象開始日付” パラメータに、当日の 午前0時を設定しています。

`stat,datetime,sample,total,mean,max,min =`
`summary_stat_data("SENSOR_TP_" .. device_name,summary_from,900,96)`
は、統計用データベースの集計を行っています。キー名に “SENSOR_TP_Device1” を指定して、”ARDUINO_SENSOR/REGISTER_ACTIVITY” スクリプトで登録した温度データを 15 分毎に集計して、”サンプル数”、”合計値”、”平均値”、”最大値”、”最小値” を計算しています。このとき、集計結果で得られた “サンプル数” は集計間隔(15分)内に見つかったデータの数を示します。リモートデバイスから自動送信される “SAMPLING” イベントの間隔(10分)が、集計間隔よりも小さい場合には必ず1 以上の値になります。その集計対象間隔の時刻にリモートデバイスとサーバー間の通信が一度も成功しなかった場合には 0 になります。

```
if sample[key] > 0 then
        temp = string.format("%2.3g",mean[key])
    else
```

```
        temp = "NULL"
end
```

部分は、該当する集計間隔にデータが無い場合に、その部分のグラフを描画しないようにするために集計結果のデータを “NULL” 文字列を代入しています。Web ページ中の JavaScript でこの “NULL” 文字列が見つかった場合には、そのポイントを描画しない様にしています。

`if not script_result(g_taskid,string.sub(val,12,16),temp) then error() end` は、スクリプトのリターンパラメータに集計結果を格納しています。キー名に、集計間隔の開始日時を指定して、値にはその集計間隔の温度平均値を設定しています。集計用のライブラリ関数 `summary_stat_data()` のリターン値で取得する `datetime` 配列値のフォーマットは “YYYY/MM/DD HH:MM:SS” になっています。このため、`string.sub(val,12,16)` で作成されたキー名は “HH:MM” になります。

ファイル名(SUMMARY_DATA.lua) で DeviceServer のスクリプトフォルダの中に ARDUINO_SENSOR フォルダを作成してその中に保存します。(“C:¥Program Files¥AllBlueSystem¥Scripts¥ARDUINO_SENSOR”)

8.6 温度グラフ表示用 Web ページ設定

Webページトップディレクトリ (”C:¥Program Files¥AllBlueSystem¥WebRoot”) 以下に arduino_sensor フォルダを作成して、温度グラフ表示用 HTML ファイル (”C:¥Program Files¥AllBlueSystem¥WebRoot¥arduino_sensor¥history.html”) を作成します。また、history.html から参照される JavaScript ファイル device_server.js も同一のディレクトリに配置します。

DeviceServer 自身の HTTPサーバーのWebページトップディレクトリ以下にファイルを配置しないで、別の HTTP サーバーで管理しているフォルダにページを配置することもできます。このような HTTP サーバーが動作している PC と DeviceServer の PCを分けて設置する場合には、device_server.js ファイル内に記述されている DeviceServer をアクセスするホスト名指定を変更してください。具体的には DeviceServer の動作している PC のホスト名または IPアドレス(インターネット上に配置する場合にはドメイン名とホスト名)を device_server.js 中の “server_host_url” 変数に設定してください。

history.html ファイルの内容は下記の様になります。

```
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<title>リモートデバイス(Arduino+XBee)の温度グラフ</title>

<script type='text/javascript' src='https://www.google.com/jsapi'></script>
<script type='text/javascript' src='device_server.js'></script>
<script type='text/javascript'>
// Google Chart API 初期化
```

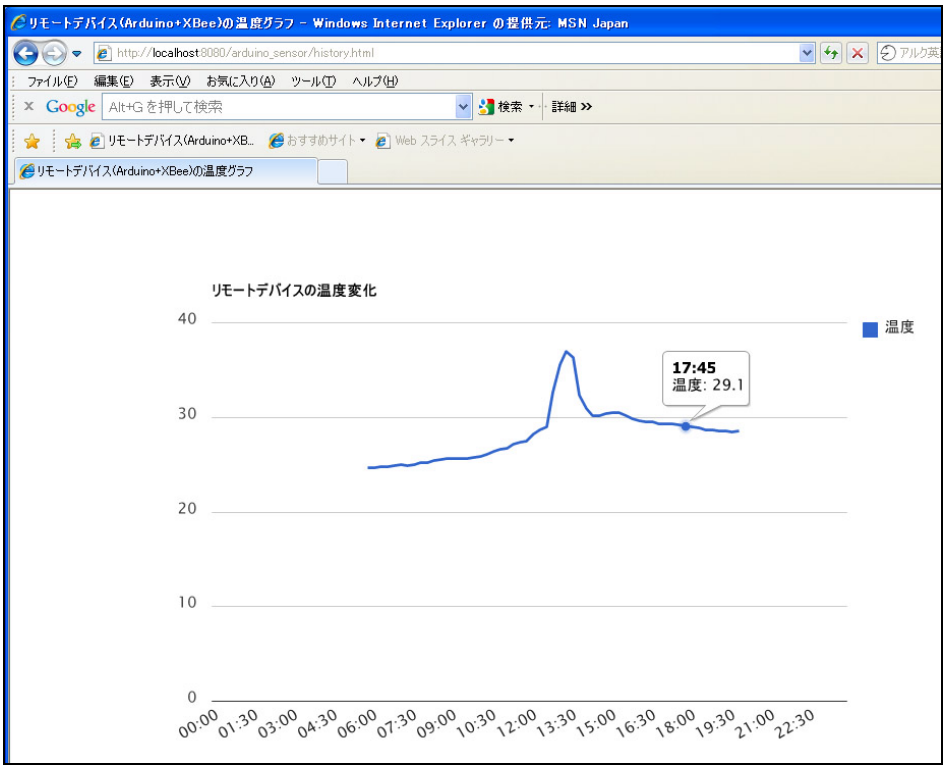
```
google.load('visualization', '1', {packages:['corechart']});
google.setOnLoadCallback(get_datas);

// 集計用スクリプトを起動する。
function get_datas(){
	// 30 分単位で温度データを集計する
	script_exec("ARDUINO_SENSOR/SUMMARY_DATA","draw_chart");
}

// ARDUINO_SENSOR/SUMMARY_DATA スクリプトの終了時にコールされるイベントハンドラ
// リターンパラメータに集計データが入っている
function draw_chart(event){
	if (event.Result != "Success"){
		log("script error!");
		return;
	}

	var data = new google.visualization.DataTable();
	data.addColumn('string', 'TimeRangeFrom');
	data.addColumn('number', '温度');
	data.addRows(array_len(event.ResultParams));

	var idx = 0;
	for(v in event.ResultParams){
		data.setValue(idx, 0, v);
		if (event.ResultParams[v] == "NULL") {
			// 対象期間内にサンプリングデータがひとつも無かった場合
			data.setValue(idx, 1,null);
		} else {
			// 対象期間の平均温度をリターンパラメータから取得
			data.setValue(idx, 1,Number(event.ResultParams[v]));
		}
		idx++;
	}

	var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
	var options =	{ width: 1000, height: 600, title:'リモートデバイスの温度変化',
			  vAxes:[{maxValue:40,minValue:0 }]
			};
```

```
            //
            chart.draw(data, options);
}
</script>
</head>
<body>
  <div id='chart_div' style='height: 600px; width: 1000px;'></div>
</body>
</html>
```

8.7 温度グラフを表示する

DeviceServer が動作しているPC もしくは、ネットワーク接続されたPC から Webブラウザを起動して、前項で設置した温度グラフ表示用 Webページをアクセスします。

例えば、WebProxy のポート番号が 8080 に設定されていた場合には、"http://localhost:8080/arduino_sensor/history.html" をアクセスします。



ページを開くと同時に、集計動作が行なわれて15 分毎の平均グラフを表示します。データが存在しない部分のグラフは描画されません。

9 このドキュメントについて

9.1 著作権および登録商標

9.2 連絡先

オールブルーシステム (All Blue System)

ウェブページ http://www.allbluesystem.com

メール contact@allbluesystem.com

9.3 このドキュメントの使用について

このドキュメントは、ABS-9000 DeviceServer の一般的な使用方法と応用例について解説してあります。お客様の個別の問題について、このドキュメントに記載された内容を実際のシステムに利用するときには、ここに記載されている以外にも考慮する事柄がありますので、ご注意ください。特に安全性やセキュリティ、長期間にわたる運用を想定してシステムを構築する必要があります。

オールブルーシステムでは ABS-9000 DeviceServer の使用や、このドキュメントに記載された内容を使用することによって、お客様及び第三者に損害を与えないことを保証しません。 ABS-9000 DeviceServer を使用したシステムを構築するときは、お客様の責任の下で、システムの構築と運用が行われるものとします。

10 更新履歴

REV A.1.0 2011/9/22

初版作成