[APNOTE14]

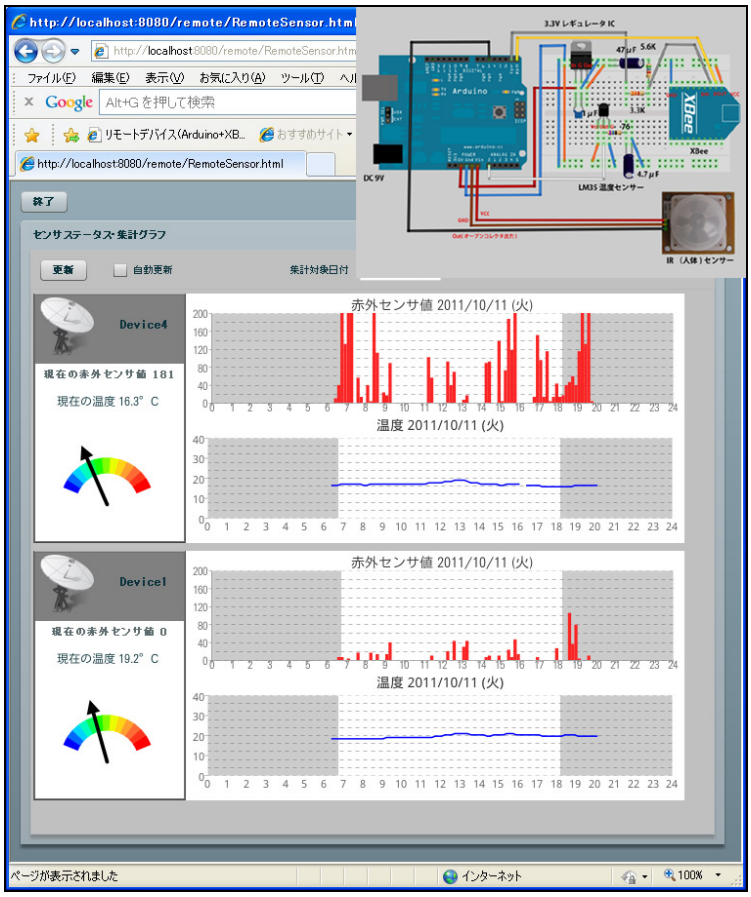
温度と赤外線センサーデータを Arduino+XBee で収集監視

(リモート在宅監視システム)

ABS-9000 DeviceServer

APNOTE14 Rev A.1.1

2012/03/16



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

屋外や屋内に設置した複数のセンサーデバイス(温度、赤外線センサー)の計測データを、定期的に無線でサーバーに送信して、集計グラフを表示するシステムの構築例を説明します。赤外線センサーからの検出が一定期間以上無かった場合に、在宅アラームとしてメールを送信する機能もあります。

各センサーデバイスでは、Arduino[1] ボードに搭載されたプログラムによって、定期的なサンプリングデータ送信とサーバーからのリモートコマンドが実行されます。サーバーとセンサーデバイス間の通信には XBee[2] デバイスを使用します。サンプリングデータの送信や、センサーデバイスのリモートコマンドとリプライデータはすべて 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 ボード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 for ATmega328 (Tiny Device Control Program)” 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。XBee デバイスはサーバー側に１つと、センサーデバイス毎に一つずつ使用しています。

センサーデバイスで計測された温度と赤外線センサーのサンプリングデータは、XBee デバイス経由でサーバーPC の DeviceServer に送信されて内部のデータベースに格納されます。その後 クライアントPC のWeb ブラウザからデータベース中のサンプリングデータを集計してグラフを表示します。

(*1) TDCP for ATmega328プログラムについて

センサーデバイス中のArduino ボード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 for ATmega328 (Tiny Device Control Program)” 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 詳細とセンサーデバイス中に使用したベースボードのマニュアルがフリーで公開しています。下記のWebページを参照して下さい。

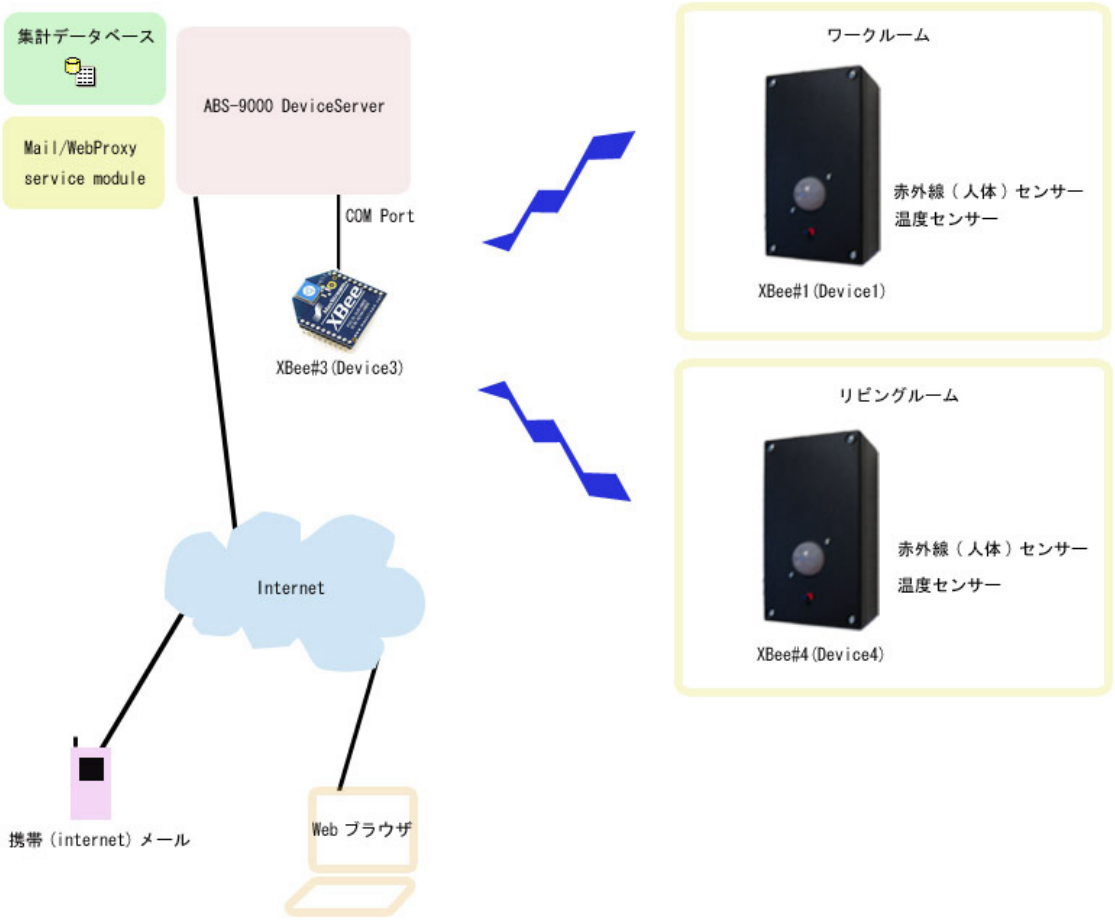
TDCP 紹介とファームウエアダウンロード　http://www.allbluesystem.com/

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

1.1　　システム全体構成図

[1] Arduino http://www.arduino.cc/

[2] XBee XBee® and XBee□PRO® are registered trademarks of Digi, Inc.



このシステムは複数のセンサーデバイスとサーバーから構成されていて、センサーデバイスは監視対象の各部屋に設置されています。センサーデバイスでは、赤外線センサーと温度センサーの計測データを定期的にサンプリングして、サーバーに送信しています。サーバーからのリクエストに応じて、現在の計測データを任意のタイミングで送信することもできます。このアプリケーションノートではセンサーデバイスとして、“Device1” と “Device4” の２つを設置する例で説明しています。

サーバーでは、各センサーデバイスから送信されたサンプリングデータを内部のデータベースに蓄積しています。ネットワークに接続した PC のWeb ブラウザからクライアントプログラム(Flash アプリケーション)を実行して、任意の日付の集計グラフを表示できます。

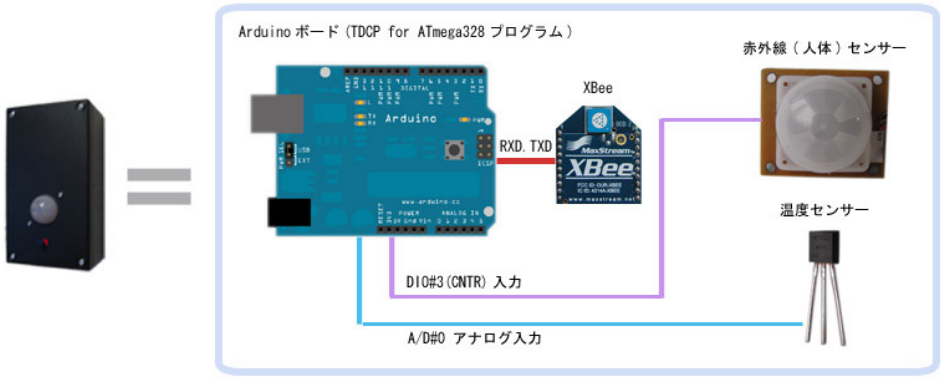
また、一定期間赤外線センサーに反応がなかった場合に、警報メールを電子メールで送信する機能もあります。

センサー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て、センサーデータのサンプリング値の送信やサーバーからのセンサーデバイス操作は、すべて無線でリモートコントロールされます。サーバーとセンサー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、設置場所を自由に決めることができ、設置場所の変更も簡単になります。

センサーデバイスとサーバー間の無線通信がエラーになった場合でも、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。システムを構成するセンサーデバイスの一部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センサーのサンプリング集計値のみに影響が限定されるようになっています。サーバー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復帰後は自動的にサンプリングや監視状態を継続できます。

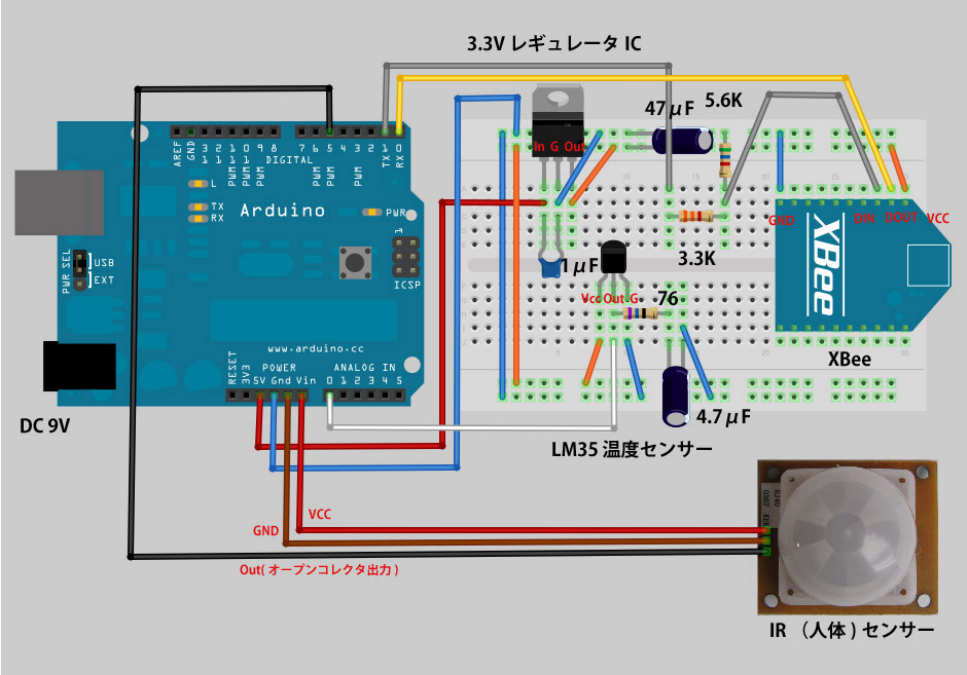
1.2 センサーデバイス構成図



センサーデバイスは 市販の Arduino ボードに、XBee デバイスと赤外線センサー・温度センサーを接続した構成になっています。Arduino ボードでは オールブルーシステム が提供している “TDCP (Tiny Device Control Program) for Atmega328” のファームウエアを動作させています。

Arduino の A/D 変換入力ポートに温度センサーが接続され、デジタル入力(カウンタ入力)ポートに赤外線センサーが接続されています。赤外線センサーは人体の動作を検出したときに断続的に ON-OFF を繰り返す仕様で、これをデジタル入力ポートから取り込んで、サンプリング期間内にセンサーが反応した数をカウントします。

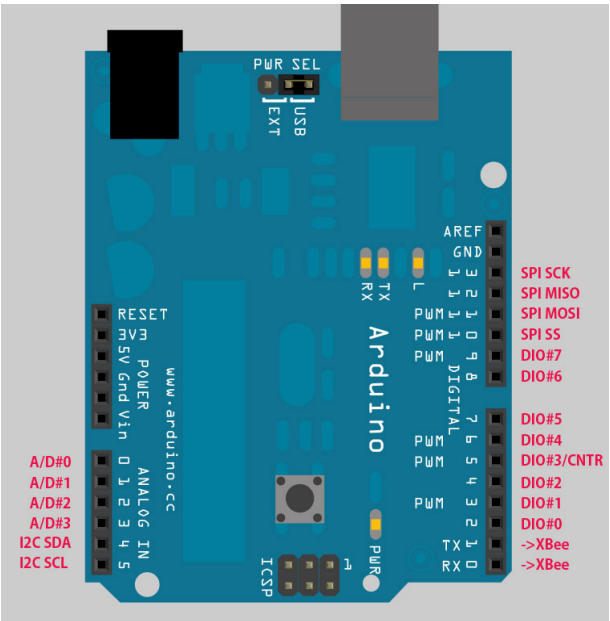
1.3 センサーデバイス配線図



センサーデバイスは、Arduino に XBee デバイスとセンサーを接続した構成になっています。リモート通信用の XBee デバイスの接続用に、外部に 3.3 V レギュレータIC とシリアル信号レベル変換用抵抗を使用しています。市販の XBee シールド等を使用する場合には、上記のレギュレータIC やシリアル信号レベル変換用の抵抗は省略できます。

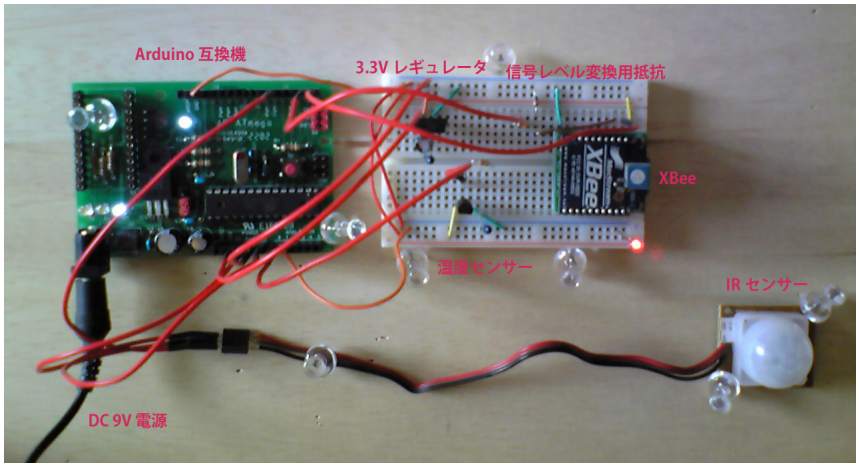
温度センサー(LM35D) は RC ダンパー付きで、A/D#0 ポートに接続しています。赤外線センサーは、デジタル入力ポート DIO#3 に接続しています。赤外線センサーの出力はオープンコレクタのため、DIO#3 の内部プルアップを有効にして動作させます。赤外線センサーの電源は +5V では電圧不足なので、Arduino ボードの電源に接続される 7..9V (Vin)に直接接続します。

TDCP ファームウエア動作時の Arduino ボードのピン配置と機能を下記に示します。



TDCP の詳しい仕様については、”TDCP for ATmega328 ユーザーマニュアル”

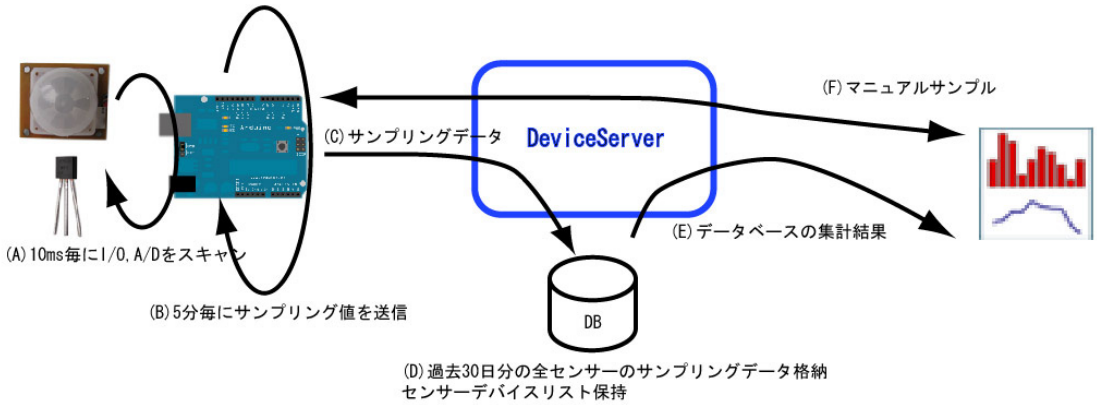
（ http://www.allbluesystem.com/TDCP/TDCP328_Users.pdf ）を参照してください。



ブレッドボードを使用して、センサーデバイスの動作を試験している状態です。この後、下記の様なプラスチックケースに回路を組み込んでセンサーデバイスを作成します。上記の写真ではXBee デバイスのピン間隔をブレッドボードに合わせるためにピッチ変換用の基板を間に挟んでいます。

センサーデバイスの個数分だけ同じ構成のハードウエアを作成します。

1.4 センサーデータの処理フロー



ここでは、システム全体の動作フローについて説明します。

センサーデバイスでは、10ms 間隔でデジタル入力ポート(赤外線センサー)と A/D 変換入力を監視しています。デジ

タル入力ポートでは、デジタル入力の変化数をカウントしています。(A)

サンプリング間隔の 10 分ごとに、現在のカウンタ値(赤外線センサーの変化数)と A/D 変換入力値(現在の温度センサー値)をサンプリングデータとして取得します。(B)

サンプリングデータは センサーデバイス内の XBee モジュールからイベントデータとして送信され、サーバーPC に接続された XBee デバイスで受信します。(C)

サーバー側では XBee データパケット受信時にイベントハンドラスクリプト(XBEE_TDCP_DATA)が実行されます。イベントハンドラスクリプト中で赤外線センサーのカウンタ値と温度を計算した後、サーバー(DeviceServer) のデータベースに登録します。(D)

クライアントプログラム（Flash アプリケーション）を実行して、サーバーのデータベースに蓄積されたセンサーデータの集計を行ってグラフを作成します。(E)

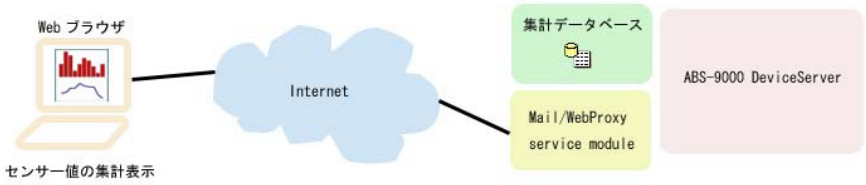
クライアントプログラムでは集計のグラフの他に、現在の温度と赤外線センサーのカウンタ値が表示されます。
この時、クライアントからマニュアルサンプリングのリクエストがネットワーク経由でサーバーに送信されます。サーバーでは対応するリクエストコマンド(TDCPコマンド) を、各センサーデバイスの XBee モジュールに対してデータパケットに入れて送信します。TDCPコマンドを受信したセンサーデバイスでは、マニュアルサンプリングを実行した後、現在のセンサーデータをリプライパケットに格納して、XBee モジュール経由でサーバーに送信します。
サーバー はリクエストを発行したクライアント（Flash アプリケーション）に、センサーデバイスから受信したマニュアルサンプリングデータをネットワーク経由で渡します。

同様の仕組みで、センサーデバイスの設定値変更や I/O ポートの操作を、クライアントから何時でも実行することができます。(F)

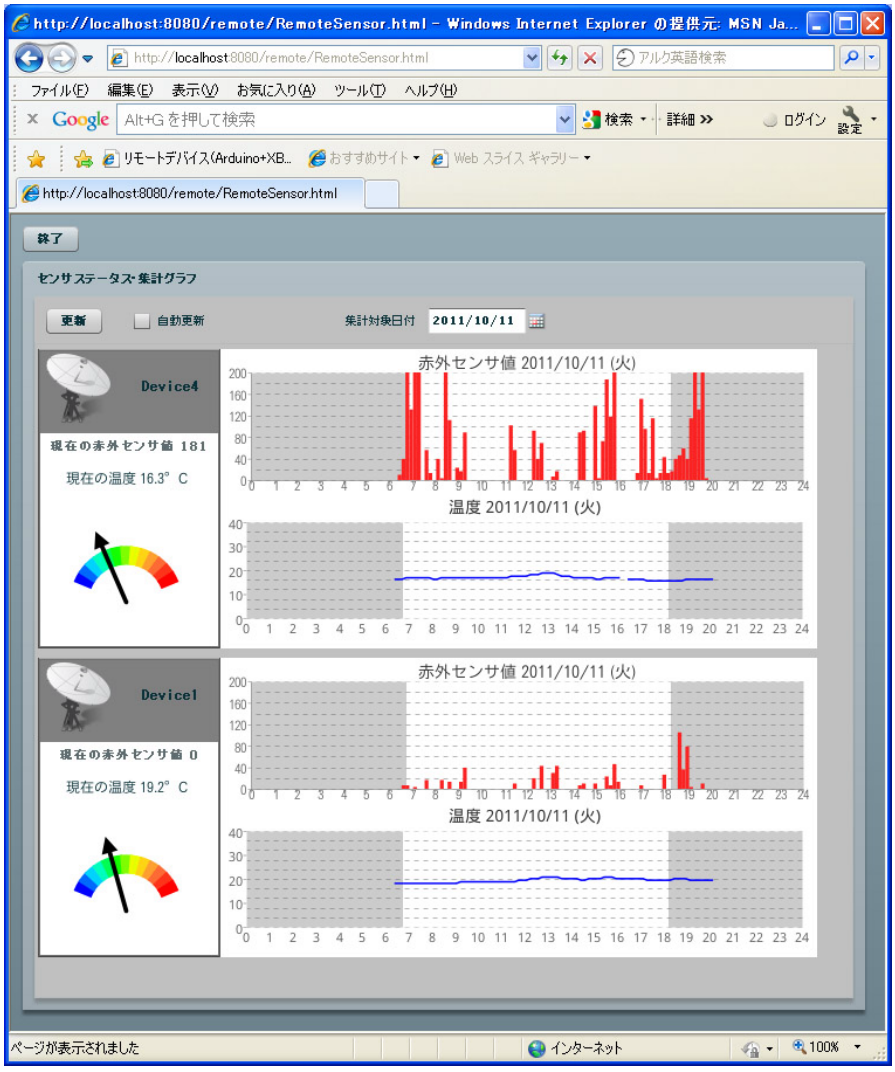
2 アプリケーション説明

このシステムでは、集計グラフを表示するためのアプリケーションと、在宅アラームメール送信の２つの機能があります。ここでは、それぞれのアプリケーションの説明をします。

2.1 Web ページ(Flash アプリケーション)から集計グラフ表示



このアプリケーションでは、インターネットやLAN に接続したPC のWebブラウザから、現在のセンサーデータの状態と指定した日付のセンサーデータの集計グラフを表示します。



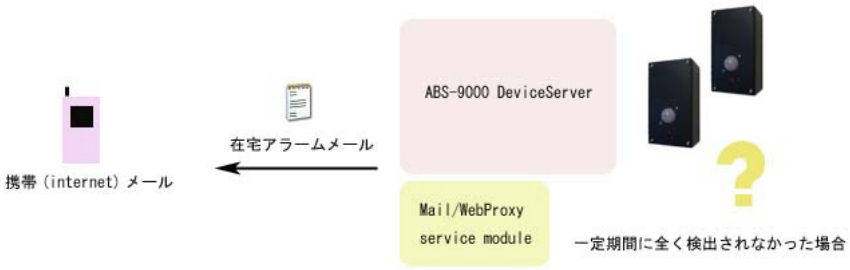
集計アプリケーションのメイン画面では、複数のセンサーデバイスを一度に確認することが出来ます。

各センサーデバイスの左パネルには現在の赤外線センサーと温度センサーの計測値が表示されます。画面上の ”更新”ボタンを押すと、全てのセンサーデバイスの表示内容が最新の情報に更新されます。センサーデバイスの計測値は、サーバーからリモートセンサーデバイスに対してマニュアルサンプリングを実行した結果が表示されています。

右パネルには ”集計対象日付” で指定した日付の集計グラフが表示されます。赤外線カウント値がバーグラフで表示されて、温度変化は折れ線グラフで表示されます。各データポイントは 10 分間隔でデータベース中のサンプリングデータを集計（平均値）した結果です。集計対象日付を変更するとサーバーで再集計を行った後、グラフが描画されます。

自動更新にチェックを入れると、約１分ごとに ”更新” ボタンを押したのと同様に画面が最新の情報に定期的に更新されます。

2.2 赤外線(人体)センサーに一定期間反応が無い場合にアラームメール送信



一定期間、センサーデバイスの赤外線(人感)センサーからの検出が無かった場合に、アラームメールを送信する機能です。これは、在宅中で生活している間には赤外線センサーにある程度の反応があることが期待できますが、もし一定期間中にセンサーが一度も反応しなかった場合に異常事態として検出する機能です。

このアプリケーションノートでは、定期的にサンプリングされるセンサーデバイスの赤外線センサーのカウンタ値が、２時間を越えて一度も検出されなかった場合に、ACTIVITIES_CHECK スクリプトが実行されてスクリプト中で指定したメールアドレスにアラームメールを送信します。

アラームメール受信	
メール件名	不在警報
メール本文	一定期間IRセンサーからの検出がありませんでした

アラームメールの送信条件は AM7:00 から PM 10:00 の間に限定して、就寝時間中など活動していない時間帯を除外するようにスクリプトを作成しています。赤外線センサーデータのチェック間隔やアラームメールの内容はスクリプトを変更することで自由に設定できます。

このアプリケーションの詳しいセットアップ方法とスクリプトの説明は、”スクリプト作成” の章を参照して下さい。
主に関連するスクリプトファイル(PERIODIC_TIMER, ACTIVITIES_CHECK,REGISTER_ACTIVITY)

3 必要な機材・リソース

3.1 サーバーPC

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[3]	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 センサーデバイス

センサーデバイスは、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。

必要なシステムやデバイス	説明
Arduino CPU ボード 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 用に Digi International Inc. 社製 XBee IEEE 802.15.4 デバイスが 1台必要です。センサーデバイスと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	LM35DZ アナログ出力可能であればどの様なデバイスでも構いませんが、デバイスを変更した場合には、スクリプト中の温度計算部分の変更が必要になります。
赤外線（人体）センサー	焦電型赤外線センサモジュール（デジタル出力タイプ、検出時のデータホールド機能無し） 今回使用したのは、秋月電子通商 “SE-10” です。これ以外でもデジ

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

	タル出力可能なタイプであれば使用できます。

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	サーバーに Web ブラウザでアクセスして、集計グラフ表示用アプリケーション(Flash) を実行します。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・センサーデバイス共通)

XBee デバイスを DeviceServer とセンサーデバイスの 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)

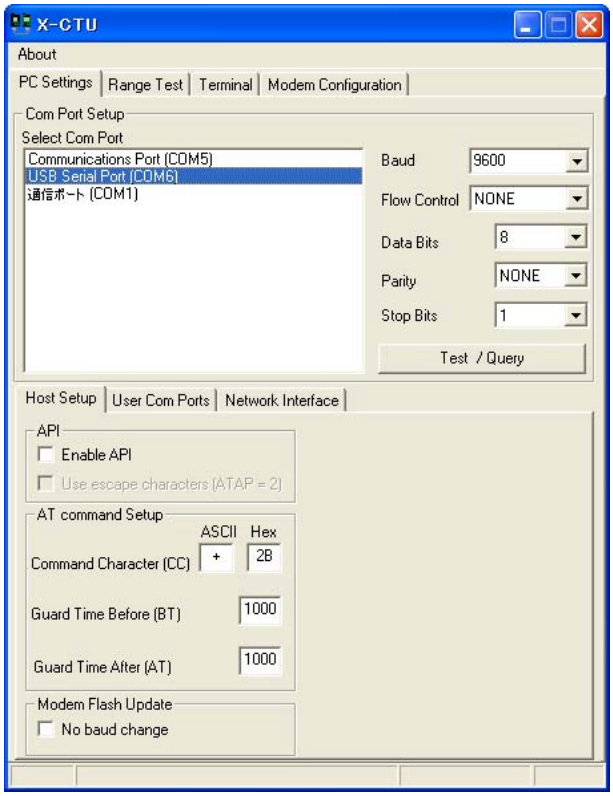
センサーデバイス用の XBee デバイスの設定についても、DeviceServer に接続する XBee デバイスを設定するときに使用した XBee Explorer USBを使用して行います。この時には、XBee Explorer USB のソケットから一時的に XBee デバイスを入れ替えて作業を行ってください。

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0x3332) 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する場合や、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0xAB90 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 コマンドを使用して設定します。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。

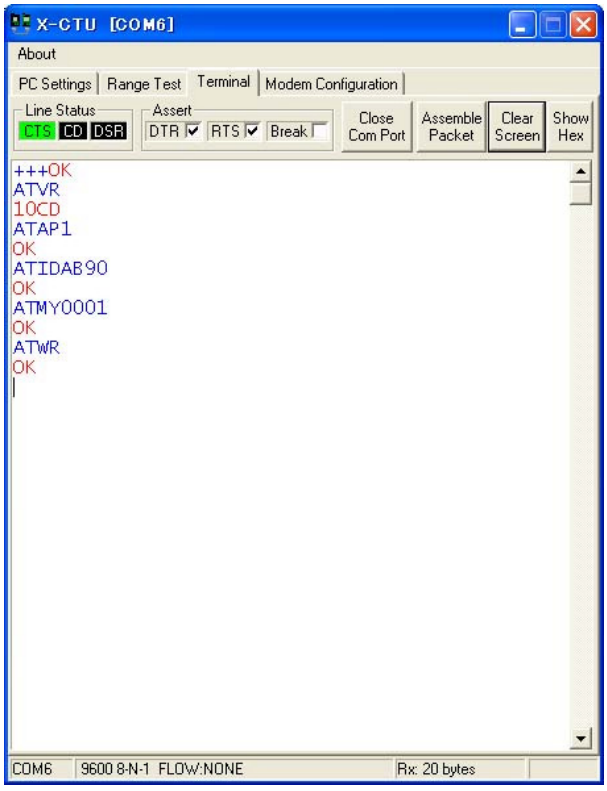
```
ATVR
ATAP1
ATIDAB90
ATMY0001
```

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。

このアプリケーションノートで説明している 各 XBee デバイスの設定値は以下のようになっています

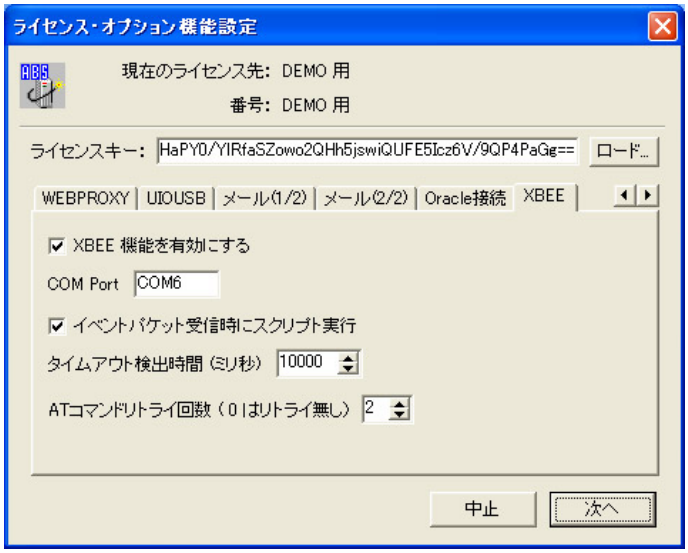
デバイス番号(用途)	API モード(ATコマンド)	PAN_ID(ATコマンド)	16bit Address(ATコマンド)
XBee#1(センサーデバイス)	1(ATAP1)	0xAB90(ATIDAB90)	0x0A01(ATMY0A01)
XBee#3(DeviceServer接続)	1(ATAP1)	0xAB90(ATIDAB90)	0x0C03(ATMY0C03)
XBee#4(センサーデバイス)	1(ATAP1)	0xAB90(ATIDAB90)	0x0D04(ATMY0D04)

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 センサー側デバイス接続

センサー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、取り出してセンサーデバイスの Arduino ボードにセットします。
センサー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 センサーデバイス上の XBee デバイスの設定変更や、デバイスの詳細設定(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

センサーデバイスの 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 に登録の後、デバイス自身の詳細設定内容をリモート操作で変更します。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。

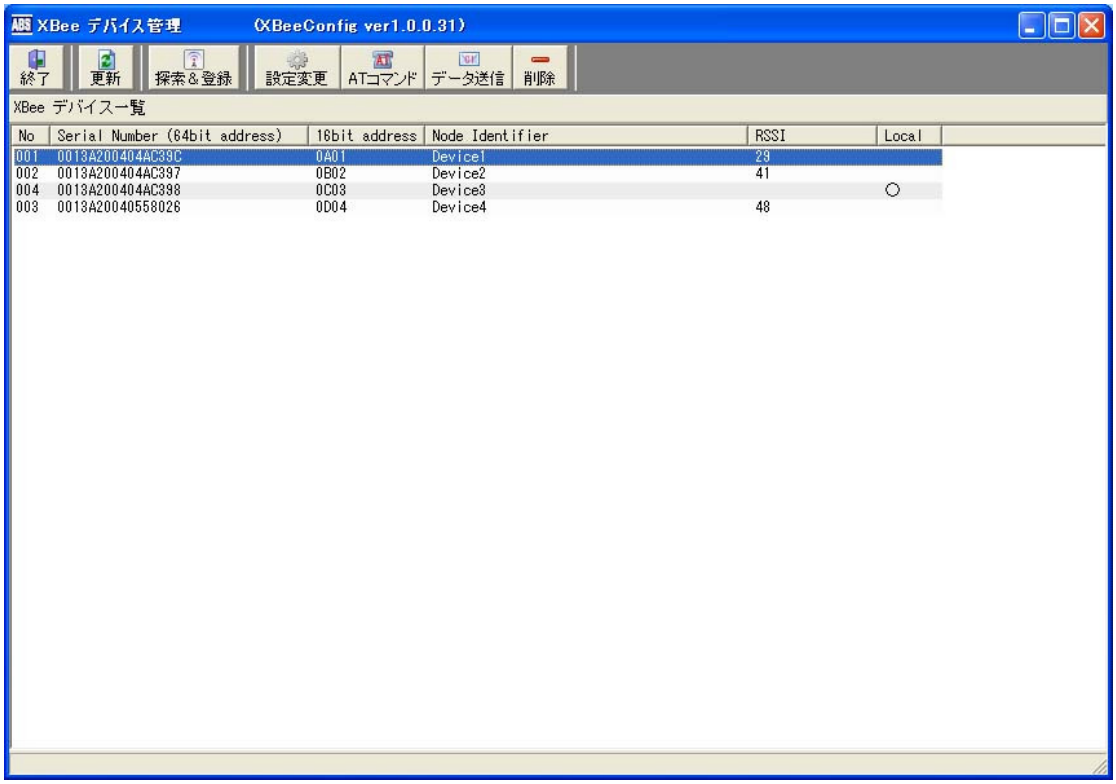
DeviceServer では 登録済みの XBee デバイスをマスターファイルに記録しています。

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。



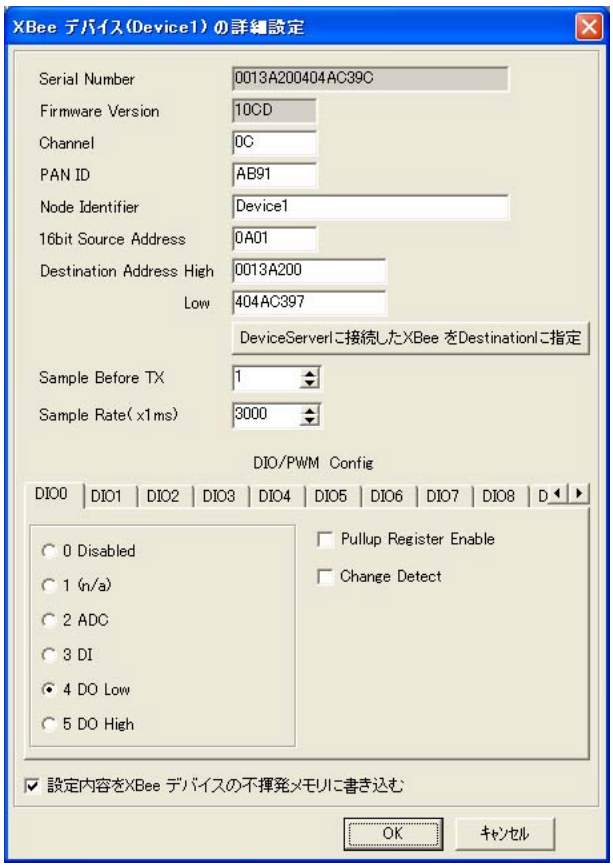
(Node Identifier 項目は、詳細設定修正後に 再度 “探索＆登録” ボタンを押すことで上記のように表示されます)

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) センサーデバイス	Device1
XBee#3 (0x0C03) DeviceServer 接続	Device3
XBee#4 (0x0D04) センサーデバイス	Device4



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 センサーデバイス TDCP 設定

センサーデバイスの TDCP ボードの設定を行います。温度センサー用の A/D 変換入力と赤外線センサーのカウンタ入力をサーバーに定期的に送信するために、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	全て入力モードに設定する。	dio_config,0
DIOが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
A/D 変換のリファレンス電源を選択	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	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の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 HOMESENSOR/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,0")
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,"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を防止できます。

--[[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メッセージ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。

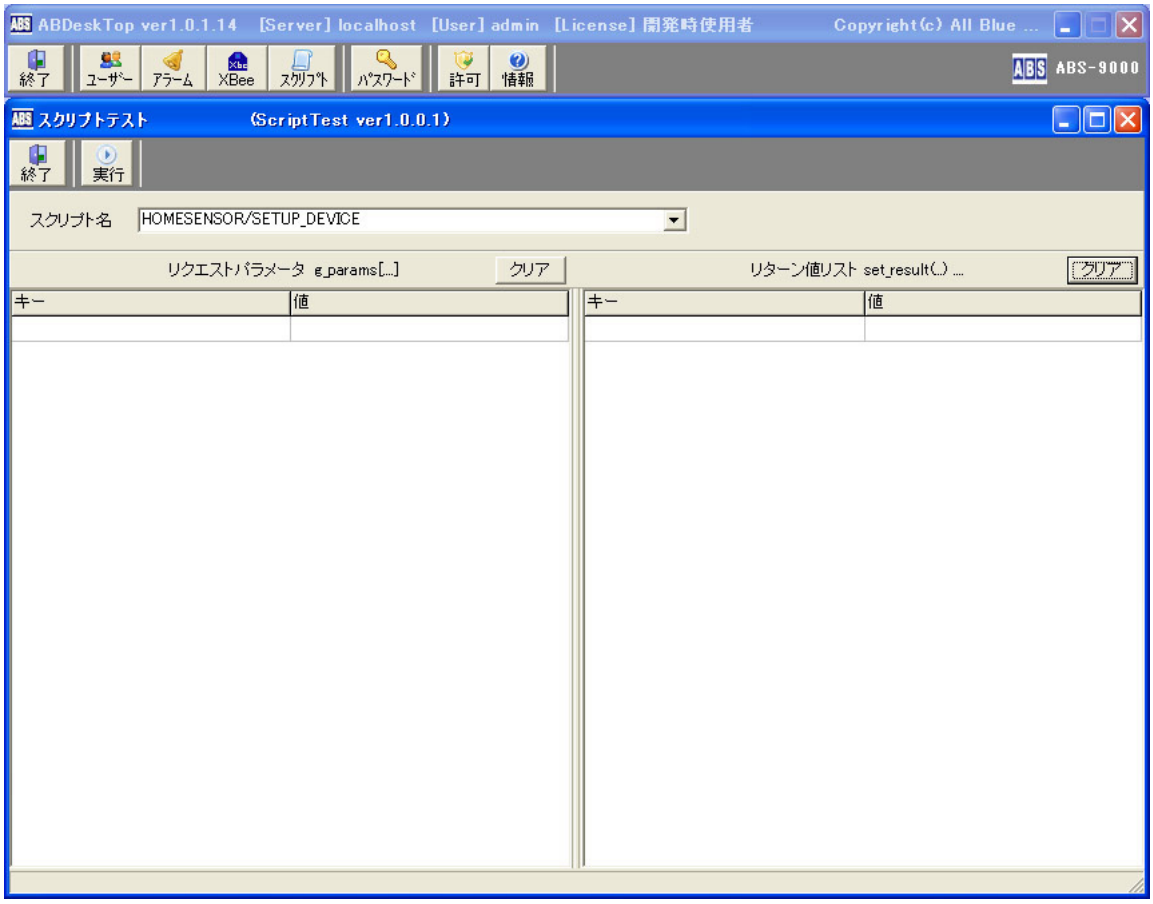
最後の stat,result = xbee_tdcp_command(device,"reset",true) は、リモートデバイスをリセットします。このときリプライステータスは一切返ってこないので、xbee_tdcp_command() の第３パラメータに true を指定してリプ

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。

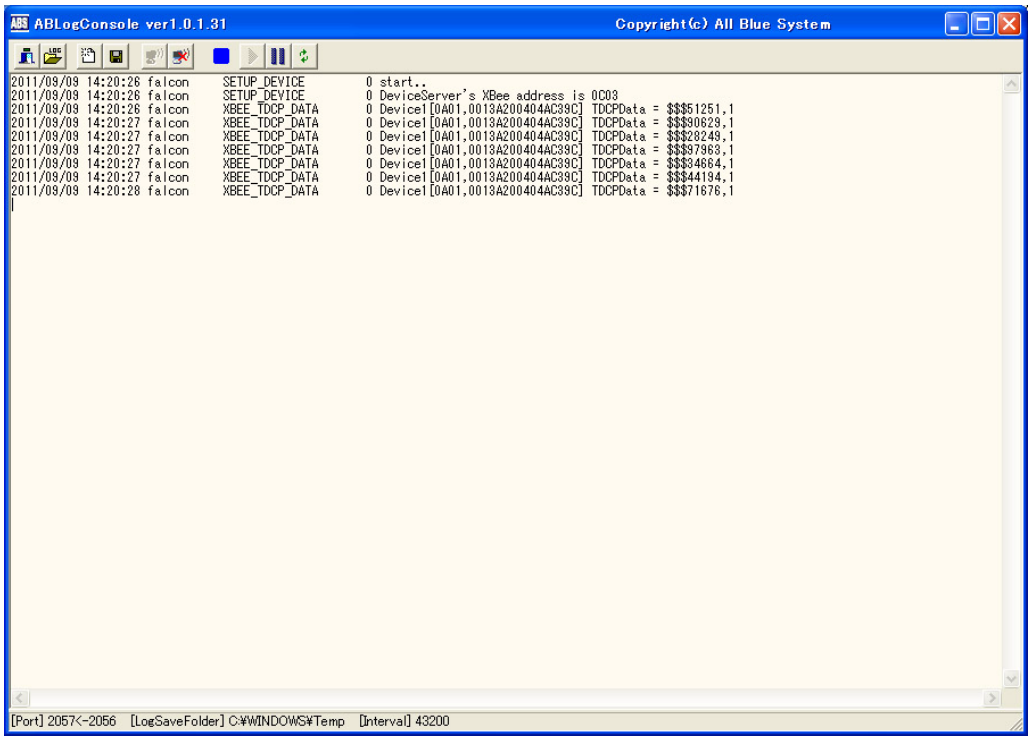
`error()` は、スクリプト実行中にエラーを検出した場合などに、スクリプトの実行を中止するときにコールする関数です。`error()` 関数を実行するとサーバーのログ中にエラー発生が記録されます。もしエラー状態にしないでスクリプトを途中で終了したい場合には、`error()` の代わりに `do return end;` の様に記述します。

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¥HOMESENSOR” に保管します。

スクリプト実行は、Webブラウザから実行する “ScriptControl(Flash)アプリケーション” 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。

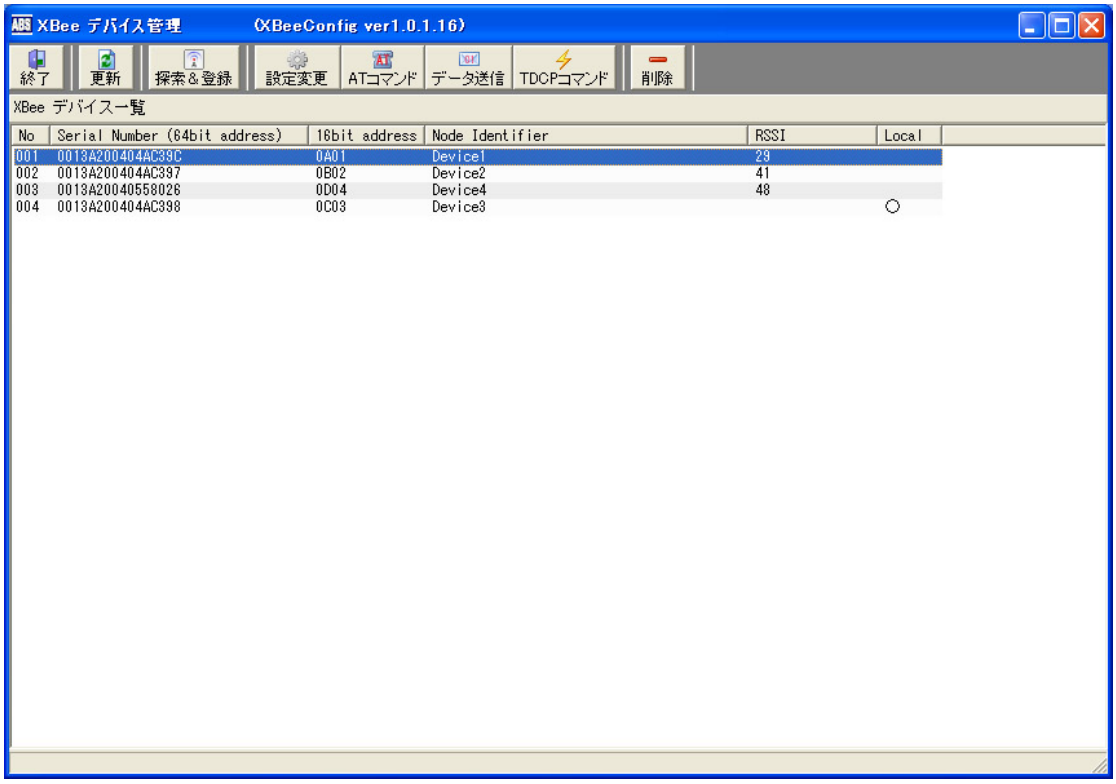


Device1 の設定が終了したら、スクリプト中の “Device1” を “Device4” に変えて同様にスクリプトを実行して、TDCP のコンフィギュレーションを行います。接続する、全てのセンサーデバイスについて同様の手順で行ってください。

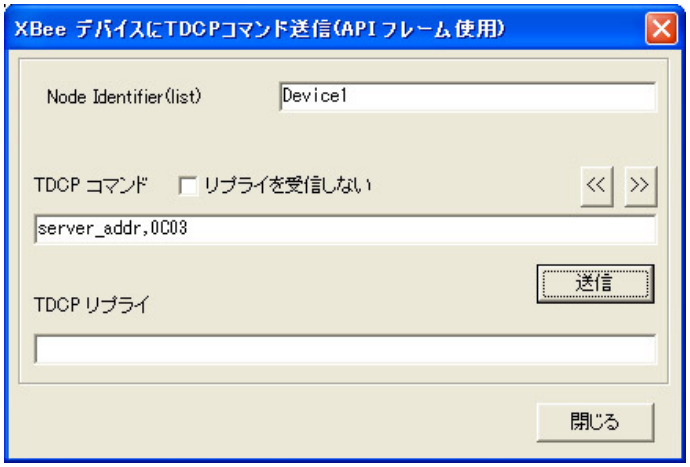
4.5.2 XBee デバイス管理プログラムから TDCP コマンド実行

スクリプトファイルを作成しないで、各 XBee デバイスに接続した TDCP プログラムにコマンドを送信して、コンフィギュレーション項目を一つずつの設定する方法について説明します。

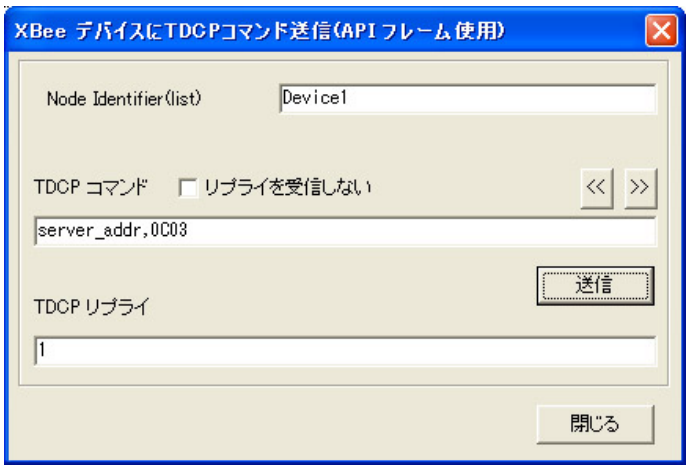
DeviceServerのクライアントプログラムを起動して、XBee デバイス管理プログラムを開きます。



“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。同様の手順で “Device4” の設定を行ってください。

5 スクリプト作成

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。

⚠ 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 XBEE_TDCP_DATA スクリプト作成

サーバーPC でセンサーデバイスからサンプリングデータやイベントデータ、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必要な機能を追加します。

```
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。
]]
```

```
--------------------------------------------------------------------------
-- 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 stat,addr16,serial,nodename = xbee_find_device(g_params["TDCP_3"]);
          local key_list =
list_to_csv("SerialNumber","RemoteDeviceName","ActivityLevel","TemperatureLevel")
          local val_list = list_to_csv(serial,nodename,g_params["TDCP_7"],g_params["TDCP_8"])

          stat = script_fork_exec("HOMESENSOR/REGISTER_ACTIVITY",key_list,val_list)
          if not stat 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
```

こに部分は、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どのような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、イベント種類を表します。自動サンプリングが実行された時のイベントは “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 stat,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("SerialNumber","RemoteDeviceName","ActivityLevel","TemperatureLevel")
local val_list = list_to_csv(serial,nodename,g_params["TDCP_7"],g_params["TDCP_8"])
stat = script_fork_exec("HOMESENSOR/REGISTER_ACTIVITY",key_list,val_list)
```

は、センサーデバイスの XBee シリアル番号、Node Identifier, 赤外線センサーカウンタ値、<adc0> (A/D チャンネル#0) の値をパラメータに指定して、スクリプト “HOMESENSOR/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。“HOMESENSOR/REGISTER_ACTVITY” スクリプトではデータベースの操作を伴うため、スクリプトの処理時間を考慮して別スレッドで実行させています。このようにすることで 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。

5.2 HOMESENSOR/REGISTER_ACTIVITY スクリプト作成

センサーデバイスからサンプリングデータを受信したときに XBEE_TDCP_DATA スクリプトから呼び出されて、データベースに登録するためのスクリプトを作成します。

スクリプトパラメータには、センサーデバイス名とサンプリング値が渡されてきますので、そのデータを元に温度を計算してデータベースに格納します。同様に赤外線センサーのカウンタ値もデータベースに格納します。スクリプトに渡されるパラメータ詳細はスクリプトのコメントを参照してください。

温度センサーの計測には A/D 変換機能を使用していますので、各センサーデバイスの実際の温度を計算するためには A/D リファレンス電圧を取得する必要があります。センサーデバイスではマイクロプロセッサ内蔵のリファレンス電圧(Vref =1.1) を使用しています。センサーデバイス毎に別の Vref 値を使用する場合には、各々のデバイス毎のリファレンス電圧を使用して温度計算を行います。このシステムではリファレンス電圧値を DeviceServer のデータベースに格納しておきます。システム起動時に一回だけ AVREF_SETUP スクリプトを実行してVref 電圧をデータベ

ースに格納します。

温度計算は、センサーデバイス毎の Vref 電圧値をデータベースから取得して、その値と現在の A/D 変換値を元に摂氏温度を計算します。データベースに温度値を格納する時には、後で集計時に検索しやすいようにデバイスシリアル番号をキーに使用して登録します。赤外線センサーのカウンタ値も同様にデータベースに格納します。

一定期間センサーデバイスの赤外線(人体)センサーからの検出が無かった場合を判断するために、赤外線センサーカウンタ値が 0 よりも多い場合には 共有データ(RESIDENTS_ACTIVITIES_FLAG) にフラグをセットします。このフラグの値は２時間ごとに ACTIVITIES_CHECK スクリプトでチェック＆クリアされています。

クライアントプログラムや他のスクリプトから、現在サンプリングデータを送信しているセンサーデバイス一覧を取得できるように、データベース(キー名 “TDCPDeviceList”) にデバイス名とシリアル番号も登録しておきます。

```
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      監視用リモートデバイス名                  LivingRoom
SerialNumber          監視用リモートデバイスのシリアル番号      0013A200404AC397
ActivityLevel         赤外センサーで関知した動作レベル(カウンタ値)        120
TemperatureLevel      温度センサーの値を A/D 変換した値         35
-------------------------------------------------------------------------------
カウンタ値      統計用データベースキー名              "SENSOR_IR_" + <SerialNumber>
温度センサーAD値 統計用データベースキー名             "SENSOR_TP_" + <SerialNumber>
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
-------------------------------------------------------------
-- 最初のデータ登録時に１回だけ、
-- A/D リファレンス電圧をデータベースに登録する
-------------------------------------------------------------
stat,val = get_shared_data("AVREF_SETUP")
if not stat then error() end
if val == "" then
```

```
          if not inc_shared_data("AVREF_SETUP") then error() end
          if not script_exec("HOMESENSOR/AVREF_SETUP","","") then error() end
end
-------------------------------------------------------------
-- A/D リファレンス電圧値取得
-------------------------------------------------------------
stat,val = get_permanent_data("AVref_" .. g_params["RemoteDeviceName"])
if not stat then
          log_msg("*ERROR* adc_vref read error",file_id)
          error()
end
adc_vref = tonumber(val)
-------------------------------------------------------------
-- 温度計算
-------------------------------------------------------------
if g_params["SerialNumber"] and g_params["RemoteDeviceName"] and g_params["ActivityLevel"] and
g_params["TemperatureLevel"] then
          temperature = (100 * adc_vref * tonumber(g_params["TemperatureLevel"])) / 1024;
          log_msg(string.format("%s IR-count = %s temperature = %2.3g",
            g_params["RemoteDeviceName"],g_params["ActivityLevel"],temperature),file_id)
else
          log_msg("*ERROR* parameter error",file_id);
          error();
end;
-------------------------------------------------------------
-- 最後に取得した温度と赤外線カウンタ値を共有データに保存して
-- 他のシステムから簡単に利用できるようにする
--
-- キー名  赤外線カウンタ値  "IR_<DeviceName>"
--         温度値            "TP_<DeviceName>"
-------------------------------------------------------------
if not set_shared_data("IR_" .. g_params["RemoteDeviceName"],g_params["ActivityLevel"]) then error() end
if not set_shared_data("TP_" .. g_params["RemoteDeviceName"],string.format("%2.3g",temperature)) then
error() end
-------------------------------------------------------------
-- デバイス名とシリアル番号をデータベースに登録
-------------------------------------------------------------
stat = add_permanent_strlist("TDCPDeviceList",g_params["SerialNumber"] .. "," ..
g_params["RemoteDeviceName"],true)
```

```
if not stat then error() end
------------------------------------------------------------
-- サンプリング値をデータベースに登録
------------------------------------------------------------
key = "SENSOR_IR_" .. g_params["SerialNumber"]
val = tonumber(g_params["ActivityLevel"])

stat = add_stat_data(key,val)
if not stat then error() end

key = "SENSOR_TP_" .. g_params["SerialNumber"]
stat = add_stat_data(key,tostring(temperature))
if not stat then error() end
------------------------------------------------------------
-- ACTIVITIES_CHECK で、IR センサーが
-- 0 以上のカウンタ値を取得したかどうかをチェックするための
-- フラグを設定する
------------------------------------------------------------
if val > 0 then
          if not set_shared_data("RESIDENTS_ACTIVITIES_FLAG","1") then error() end
end
```

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

5.3 HOMESENSOR/AVREF_SETUP スクリプト作成

前述の REGISTER_ACTIVITY スクリプトから1 回だけコールされて、センサーデバイスごとのリファレンス電圧値を DeviceServer のデータベースに格納するためのスクリプトを作成します。

センサーデバイスの A/D リファレンス電圧値(Vref)をスクリプト中に記入してデータベースに登録します。データベースに登録した Vref 値は A/D 変換値から温度を計算するときに、他のスクリプトやクライアントプログラムから使用されます。

```
file_id = "AVREF_SETUP"
--[[
TDCP デバイスの A/D 変換に使用される A/D Vref 値を
データベースに保存する。クライアントプログラム(Flash) から
この値を取得して、A/D ポートに接続された各種センサーの
検出値を計算するときに利用する。
```

```
A/D Vref 値が変化した場合には必ずこのスクリプトを修正した後、手動で実行してください。
DeviceServer の再起動時に、REGISTER_ACTIVITYスクリプト中から
一回だけこのスクリプトが自動で実行されます。



データベースに保存するパラメータ
--------------------------------------------------------------------------------
キー値                  値                                                  値の例
--------------------------------------------------------------------------------
AVref_<DeviceName>   <DeviceName> で指定された XBee デバイス(Node Identifier)
                     が接続されたTDCP デバイスの A/D Vref 値を指定する      "5.00"
                     デバイスが複数ある場合には全ての A/D Vref 値を
                     同様に指定する
--------------------------------------------------------------------------------
]]
log_msg("start..",file_id)

-- A/D Vref 値をクライアント側で利用するためにデータベースに保存する
if not set_permanent_data("AVref_Device1","1.10") then error() end
if not set_permanent_data("AVref_Device4","1.10") then error() end
```

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

5.4 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。

このスクリプト中に、データベース中の古いセンサーデータを消去する SENSOR_DATA_PURGE スクリプトと、一定期間、センサーデバイスの赤外線(人体)センサーからの検出が無かった場合を判断するための ACTIVITIES_CHECK スクリプトを２時間ごとに起動するように記述します。

```
file_id = "PERIODIC_TIMER"
--[[
********************************************************************************
＊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アラームデバイスで、
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。
*******************************************************************************
]]
-------------------------------------------------------------------------
-- 2時間に一回 ACTIVITIES_CHECK,SENSOR_DATA_PURGE スクリプトを実行する
-------------------------------------------------------------------------
stat,val = inc_shared_data("TIME_2H")
if not stat then error() end
if (tonumber(val) == 1) then
          if not script_fork_exec("HOMESENSOR/ACTIVITIES_CHECK","","") then error() end
          if not script_fork_exec("HOMESENSOR/SENSOR_DATA_PURGE","","") then error() end
end

if (tonumber(val) > 120) then
          stat = set_shared_data("TIME_2H","")
          if not stat then error() end
end
```

共有変数 TIME_2H は、2時間(120分)をカウントするために使用しています。別の共有変数名を使用しても構いません。

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。

5.5 HOMESENSOR/SENSOR_DATA_PURGE スクリプト作成

PERIODIC_TIMER スクリプトから２時間ごとにコールされて、過去30 日よりも古いサンプリングデータを削除します。データベースサイズが際限なく大きくなるのを防ぐためにクリプトを設定します。ただし、このスクリプトで削除対象となったサンプリングデータは集計できなくなります。スクリプトを変更して保存対象となる日数を自由に変更することもできます。

```
file_id = "SENSOR_DATA_PURGE"
-- 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_",timestamp,"") then error() end
```

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

5.6 HOMESENSOR/ACTIVITIES_CHECK スクリプト作成

PERIODIC_TIMER スクリプトから２時間ごとにコールされて、一定期間、センサーデバイスの赤外線(人体)センサーからの検出が無かった場合にアラームメールを送信します。

```
file_id = "ACTIVITIES_CHECK"
log_msg("start..",file_id)
local mail_addr = "不在警報メール宛先 <your_mail_address@your.mail.domain>"
local flag_name = "RESIDENTS_ACTIVITIES_FLAG"
-----------------------------------------------------------------------
-- 前回このスクリプトを実行してから、今回このスクリプトを実行する
-- までの間に、IR カウンタの値を全く検出していない場合には、警告メールを
-- 送信する。このスクリプトの実行間隔は PERIODIC_TIMERスクリプト中で
-- 設定されている。
-----------------------------------------------------------------------
local now = os.date "*t"
local stat,flag = get_shared_data(flag_name)
if not stat then error() end
-----------------------------------------------------------------------
-- 検出対象となる時間帯を 07:00 - 22:00 の日中に限定している
-----------------------------------------------------------------------
if (flag == "") and (now["hour"] >= 7) and (now["hour"] <= 22) then
          local body = {}
          table.insert(body,"一定期間IRセンサーからの検出がありませんでした")
          if not mail_send(mail_addr,"","不在警報",unpack(body)) then error() end
          log_msg("*WARNING* 一定期間IRセンサーの反応がありません",file_id)
end

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

```
local mail_addr = "不在警報メール宛先 <your_mail_address@your.mail.domain>"
```

この部分は、メールの送信アドレスを設定しています。設置する環境に合わせてメールの宛先を正しく設定してください。

赤外線データを検出したかどうかのフラグ(共有データ) は RESIDENTS_ACTIVITIES_FLAG をチェックしています。このフラグは REGISTER_ACTIVITY スクリプトからセットされます。

```
if (flag == "") and (now["hour"] >= 7) and (now["hour"] <= 22) then
```

このスクリプトでは夜間寝ている間など、在宅者が活動していない期間は間違ってアラートメールを送信しない用に時間帯を判断しています。検出対象となる時間帯を変更したり、曜日によって時間帯を変える場合には、この判断文を変更して下さい。

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

6 Web クライアントプログラム設定

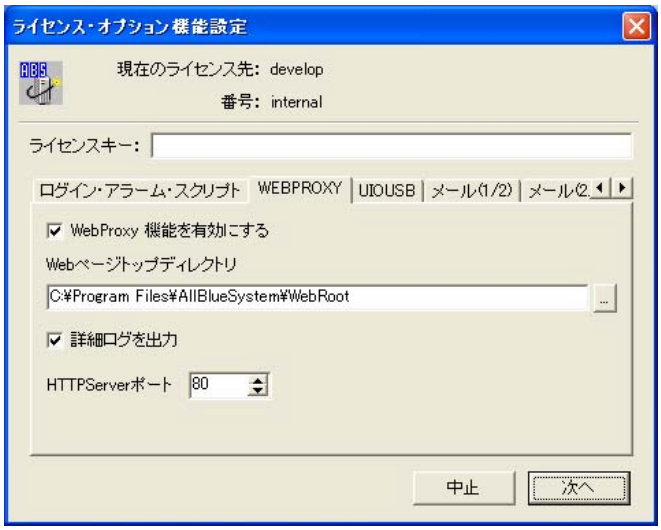
インターネットまたはLAN 上の Webブラウザから、現在のセンサーデータのサンプリング値と指定した日付のセンサーデータグラフを参照するための RemoteSensorアプリケーションの設定方法について説明します。

6.1 DeviceServer の WebProxy セットアップ

DeviceServer には Webブラウザの Flash アプリケーションからコマンド操作をするためのインターフェイス WebProxy機能があります。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、同一 PC 上でDeviceServer の WebProxy 機能を動作させることはできません。これらの既存の HTTP サーバーとDeviceServer のWebProxy 機能を両方共使用したいときには、それぞれを別のPC に分離して設置します。設置の方法については、"DeviceServer ユーザーマニュアル" 中の "DeviceServerとHTTPサーバーを分離して設定する" の項を参照してください。

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 プログラムインストール

DeviceServer インストール時に、”スクリプト操作” など Webブラウザから実行する幾つかの Flash プログラムがインストールされています。ここでは、センサーデバイス操作を行うための Flash プログラムを DeviceServer に追加設定します。

添付ファイルの “WebRoot” フォルダ中の下記の２つのファイルを DeviceServer をインストールしたフォルダにコピーしてください。(デフォルトフォルダ “C:¥Program Files¥AllBlueSystem¥WebRoot¥remote”)

追加するファイル

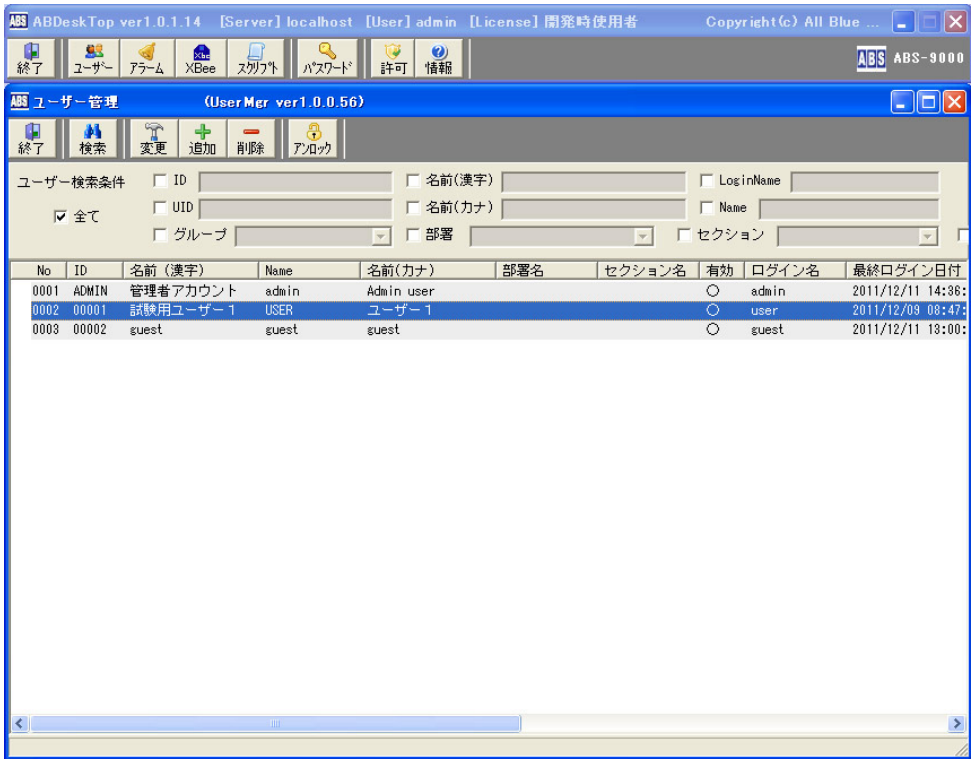
- RemoteSensor.swf
- RemoteSensor.html

DeviceServer 自身の持つ http サーバー機能を使用しないで、別の http サーバープログラムを使用する場合や、http サーバーの動作する PC とDeviceServer の動作する PC が別ドメインにある場合には crossdomain.xml ファイルと servercfg.xml ファイルの設定が必要になります。詳しい設定方法は、”DeviceServer ユーザーマニュアル” 中の “DeviceServerとHTTPサーバーを分離して設定する” の項を参照してください。

6.3 ユーザーアカウントの設定

Webブラウザから実行する RemoteSensor (Flash)アプリケーションは、ログイン認証を行ってから操作できるようになります。WebProxy経由でログイン可能にするために、クライアントプログラムを起動してユーザー管理プログラムを実行します。操作を行うユーザー情報のアプリケーション許可フラグ中の “WebLogin” と “RemoteSensor” にチェックを付けて下さい。

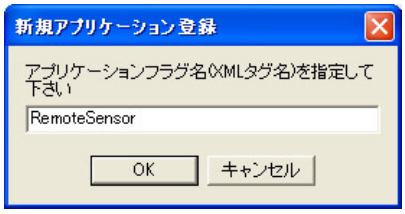
下記は、DeviceServer クライアントプログラムから、ユーザー管理プログラムを起動した状態です。

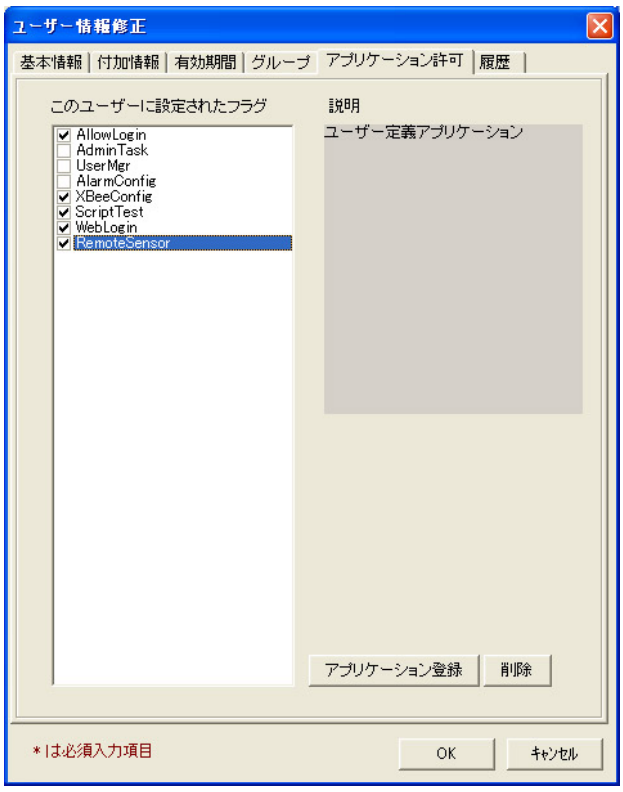


ここで、センサーデバイス操作アプリケーションにログインするユーザーを新規に作成するか、既存のユーザーを選択して”ユーザー情報”タブを表示します。

インストールする Flash プログラム RemoteSensorMgr.swf（センサー管理プログラム）は、プログラムの実行権限をユーザーが持っているかどうかをチェックしています。このため、ログインユーザー情報のアプリケーション許可フ

ラグに“RemoteSensor” フラグを追加指定する必要があります。ユーザー情報の “アプリケーション許可” タブを選択します。ユーザーに現在許可されているフラグの一覧が表示されますので、”アプリケーション登録” ボタンを押して “RemoteSensor” フラグを追加してください。



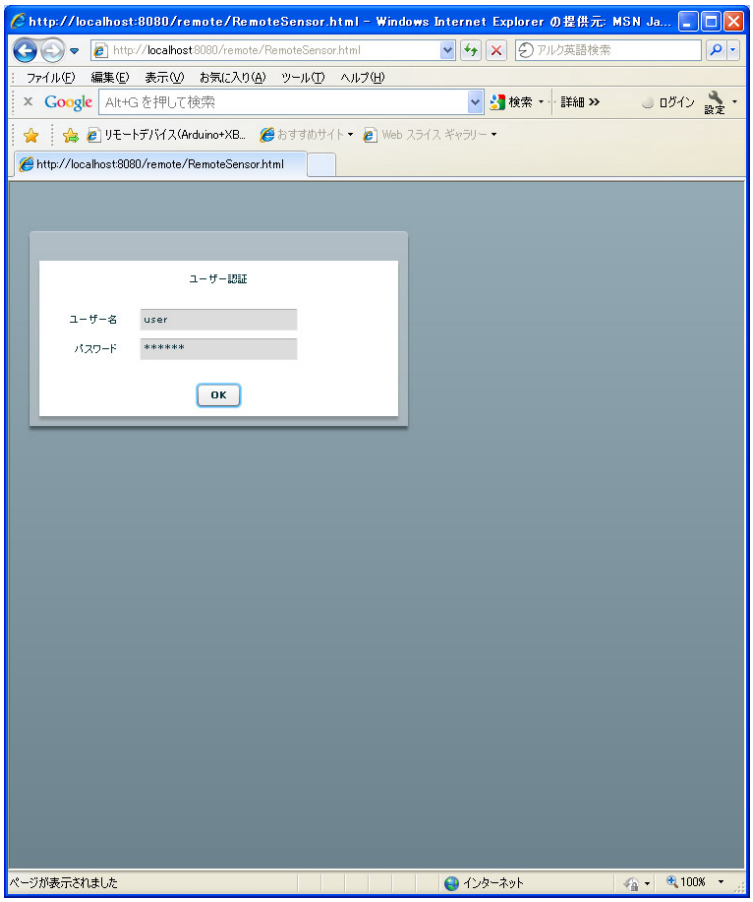


いま追加した “RemoteSensor” アプリケーションフラグにチェックを付けます。また、“AllowLogin”,”WebLogin” の両方のフラグにもチェックを付けて下さい。“OK” を押してユーザー情報を修正します。

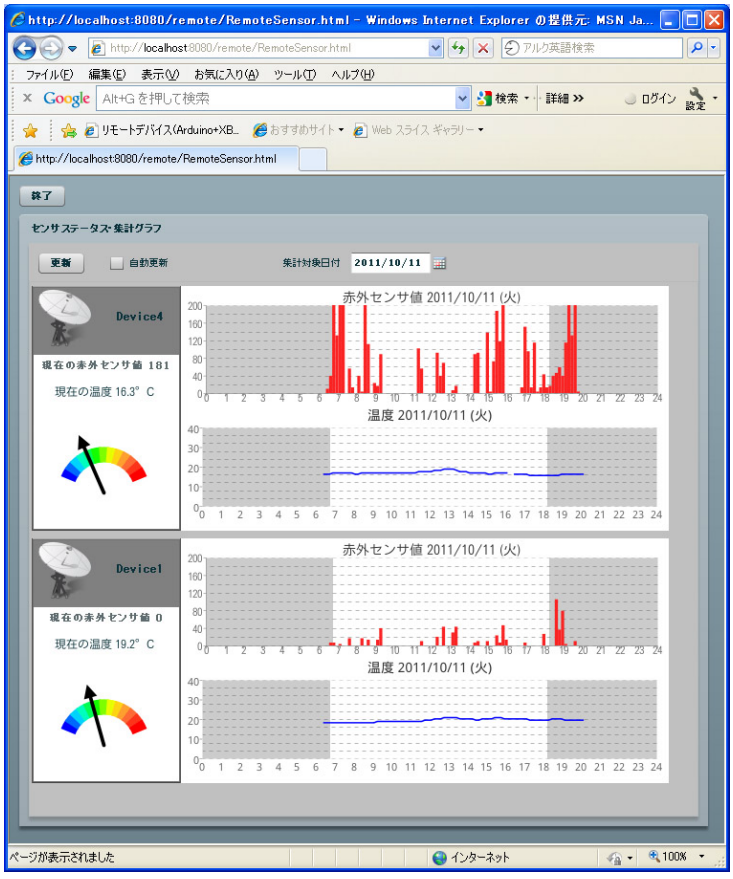
7 アプリケーションを起動する

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。”http://localhost/remote/RemoteSensor.html”

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



ログイン画面が表示されますので、"RemoteSensor" アプリケーションフラグを有効に設定したユーザーでログインします。



設定されている全てのセンサーデバイス一覧が表示されます。センサーデバイスが３つ以上ある場合にはスクロール表示されます。各デバイスの左部分には、現在のセンサーデバイスの計測値が表示されています。これはアプリケーション起動時にマニュアルサンプリングを実行して、リモートデバイスからサンプリング値を直接取得しています。

右部分は選択した集計対象日付の、温度と赤外センサーグラフが表示されます。グラフは10 分毎に集計された値を表示しています。リモートからのサンプリングデータが取得できなかった部分のグラフは描画されません。

赤外センサーの棒グラフの濃い赤色は、その時間帯の平均値を示しています。センサーデバイスから送信するサンプリング間隔を 10 分より短くすると、その時間帯に取得したサンプリング値の最大値が、薄い赤色でグラフに重なって表示されるようになります。

8 このドキュメントについて

8.1 著作権および登録商標

8.2 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

8.3 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。

9 更新履歴

REV A.1.1 2012/03/16

サーバーPC に接続する XBee デバイス名が XBee#2 になっていたのを XBee#3 に修正した

REV A.1.0 2011/12/14

初版作成