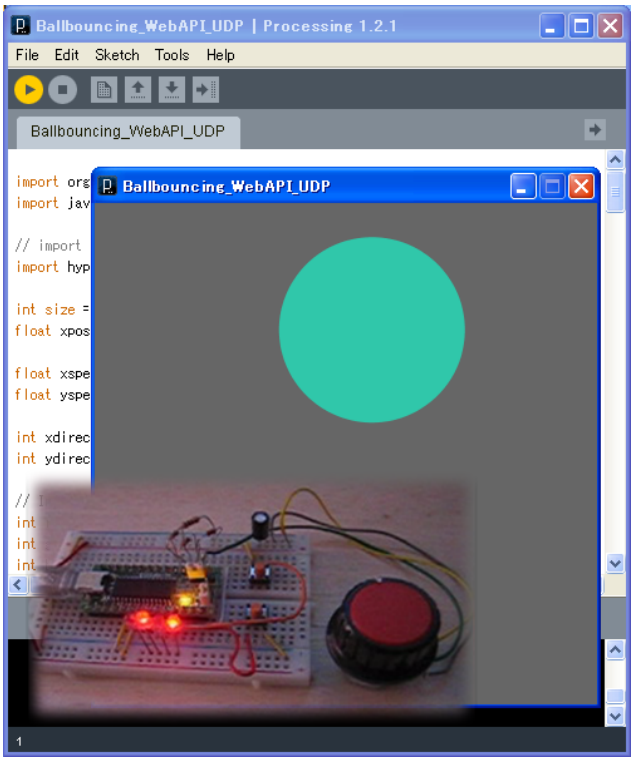
[APNOTE12]

Processing から UIOUSB(USB I/O) デバイスを操作

ABS-9000 DeviceServer

APNOTE12 Rev A.1.0

2011/02/03



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

データのビジュアライズツールとして広く利用されている Processing[1] と DeviceServer を接続してデータのやり取りや、互いのイベントでプログラムの動作をコントロールすることができます。このアプリケーションノートでは、USB I/O デバイス UIOUSB(*1) を Processing からリアルタイムに操作します。

DeviceServer のスクリプトを使用して、UIOUSB 上で発生したイベント(ポート値の変化、A/D 値の変化等)を Processing に通知して、Processing 側でイベントハンドラの動作やデータの受け渡しを行ないます。

Processing 環境に用意されているデモスケッチプログラム (Bounce)に変更を加えて、Processing と UIOUSB 間のデータ受け渡しを行なうためのスケッチ(Ballbouncing_WebAPI_UDP)を作成します。

Ballbouncing_WebAPI_UDP スケッチ(以降 スケッチ) は、ボールがウィンドウの端で跳ね返るアニメーションを表示します。この時、ウィンドウの上下左右でボールが跳ね返った時に USB I/O デバイス(以降 UIOUSB)に接続した LED を反転させます。UIOUSB に接続したタクトスイッチを押すとスケッチで動かしているボールの色をランダムに変更します。同様に、UIOUSB に接続したボリュームを回すとで、ボールの大きさをリアルタイムに変化させます。

UIOUSB では、接続したタクトスイッチを押したりボリュームを回すと、I/O ポート値や A/D 変換値が変化して、イベントを DeviceServer に送信します。DeviceServer では イベントハンドラスクリプト中で、Processing で動作しているスケッチに UDP パケットを送信します。スケッチプログラムでは UDP パケットを受信・解析して、ボールの色と大きさを変更します。

スケッチプログラムのアニメーションでボールがウィンドウの上下左右で跳ね返った時には、スケッチから WebAPI(HTTP プロトコル GET) で DeviceServer にアクセスしてスクリプトを実行してI/O ポート出力を変更します。またボールの大きさの初期値も、WebAPI 経由でスクリプトを実行して、UIOUSB に接続したボリュームのA/D 変換値を取り込んで設定します。

(*1) UIOUSB デバイスについて

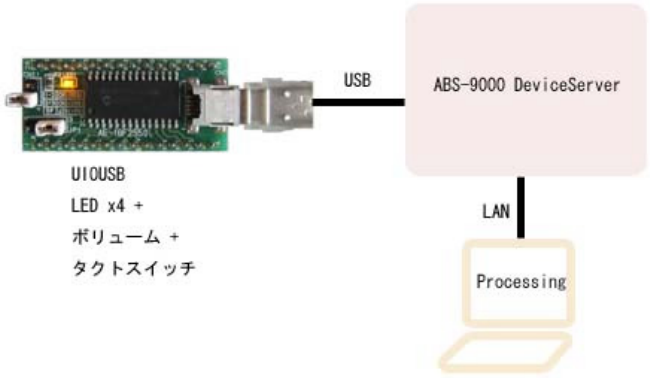
USB 接続する I/O 装置 UIOUSB は、市販の PIC18F2550 CPU ボードに、オールブルーシステムが提供している “UIOUSB ファームウエア” を搭載しています。UIOUSB はフリーで使用することができます。詳しい機能やライセンス、使用方法については下記のマニュアルを参照してください。

(UIOUSB マニュアルとファームウエアダウンロード)

http://www.allbluesystem.com/dl.html

[1] Processing was initiated by Ben Fry and Casey Reas. It is developed by a small team of volunteers. (http://processing.org/)

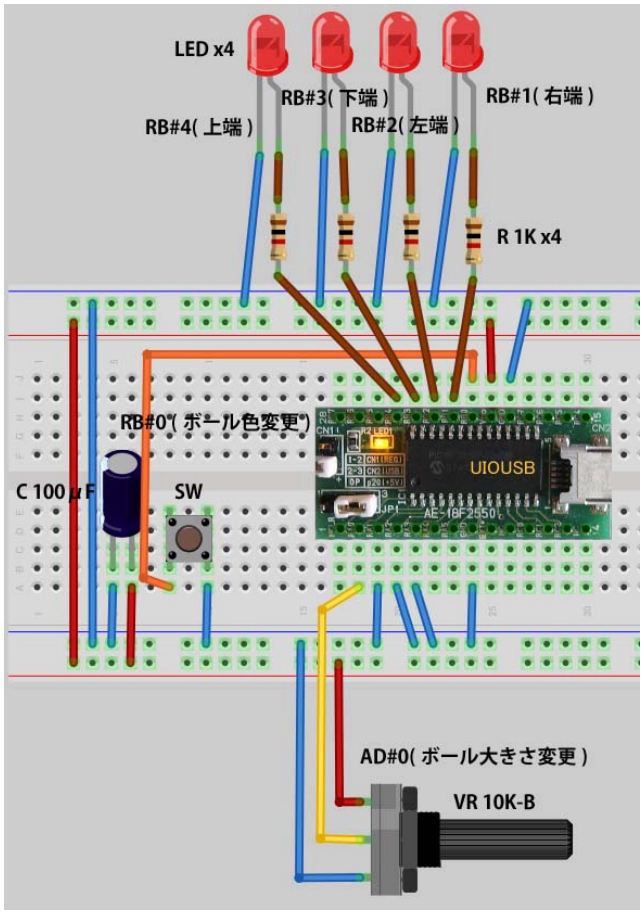
1.1 システム全体構成図



このアプリケーションノートで実行するシステムは、タクトスイッチやLED,ボリュームが接続された UIOUSB デバイスと、DeviceServer が動作するサーバーPC、Processing プログラムが動作するクライアント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 で Processing プログラムを起動して動作させることもできます。

1.2 UIOUSB(USB I/F)デバイス配線図

UIOUSB デバイスに接続する各パーツの配線図です。



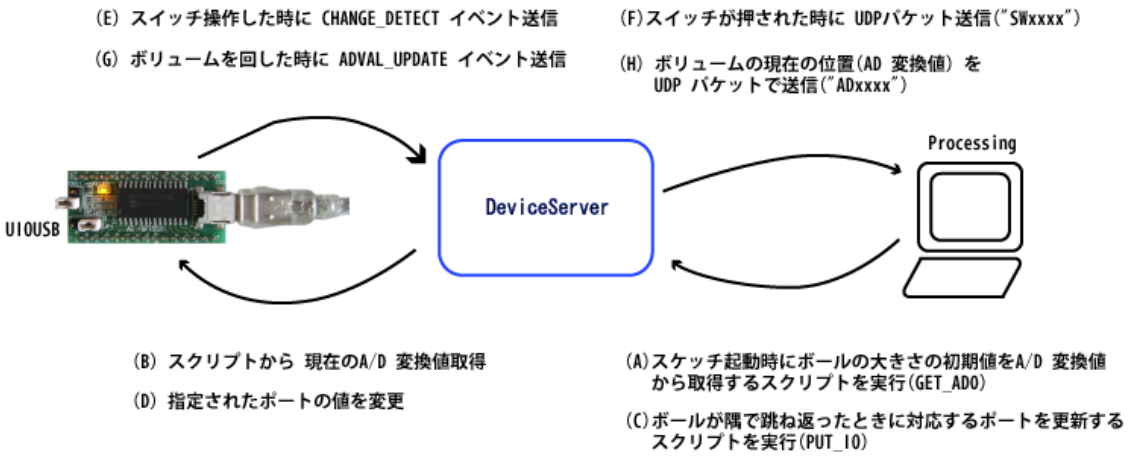
I/O ポートの ビット0 が入力モードで、その他の I/O ポートのビットは出力モードに設定します。ビット0 にはタクトスイッチ(SW)を接続します。CPU 内蔵のI/O ポート入力プルアップを使用しますので、タクトスイッチを押すことでポート値が High から Low に切り替わります。

I/O ポートの ビット1 から4 には電流制限抵抗(R) を通して LED が接続されています。対応するI/O ポートのビット値が “1” になると LED が点灯します。

A/D 変換入力のチャンネル 0 には可変抵抗器(VR)によって得られた電圧を入力します。これによってVR を回すことで A/D チャンネル0 の変換値が変化します。

電源には動作安定の為に、バイパスコンデンサ(C) を接続しています。この回路は UIOUSB のUSB ポートから電源が供給されますので外部電源は必要ありません。

1.3 イベントデータの処理フロー



Processing で動作するスケッチと、UIOUSB と接続された DeviceServer 間でやりとりされるイベントとその動作フローを説明します。

Ballbouncing_WebAPI_UDP スケッチは、ボールがウィンドウの端で跳ね返るアニメーションを表示します。
スケッチ起動時に、ボールのサイズの初期値を決めるために現在のボリューム(UIOUSB デバイスに接続された可変抵抗器) の位置を A/D 変換値を取得するためのスクリプト(GET_ADO) を WebAPI 経由で DeviceServer で実行して取得します。(A)

DeviceServer では GET_ADO スクリプト中から、UIOUSB デバイスの A/D 変換値を取得するためのコマンドを実行して A/D 変換値を取得します。(B)

スケッチのアニメーションで表示しているボールがウィンドウの上下左右で跳ね返った時に、UIOUSB に接続した

LED を反転させるために、対応するポートのビット番号と点灯または消灯を意味するフラグ値をパラメータに指定してスクリプト(PUT_IO)を起動します。(C)

DeviceServer では PUT_IO スクリプト中から、UIOUSB デバイスのポート値を変更するためのコマンドを実行します。(D)

UIOUSB に接続したタクトスイッチを押すと、DeviceServer の UIOUSB サービスモジュールで CHANGE_DETECT イベントが発生して、イベントハンドラスクリプト(UIOUSB_EVENT_DATA)が実行されます。(E)

DeviceServer の UIOUSB_EVENT_DATA イベントハンドラでは、イベントの種別が CHANGE_DETECT で、かつボタンが押されたとき(ボタンリリース時にも同様にイベントハンドラが実行されますが今回は無視します) に UDP パケットを スケッチに送信します。スケッチプログラムでこの UDP パケットを受信すると、ボールの色をランダムに変更します。(F)

UIOUSB に接続したボリューム を回すと、DeviceServer の UIOUSB サービスモジュールで ADVAL_UPDATE イベントが発生して、イベントハンドラスクリプト(UIOUSB_EVENT_DATA)が実行されます。(G)

DeviceServer の UIOUSB_EVENT_DATA イベントハンドラでは、イベントの種別が ADVAL_UPDATEの時に、現在の A/D 変換値を含めた UDP パケットをスケッチに送信します。スケッチプログラムでこの UDP パケットを受信すると、ボールの大きさを A/D 変換値に合わせて変更します。(H)

2 必要な機材・リソース

必要なシステムやデバイス	説明
ABS-9000 DeviceServerの動作しているPC	DeviceServer の動作する PCが１台必要です。
UIOUSB デバイス 下記は、デバイスに接続するパーツ一覧 *LED x4 *抵抗 1K x4 *ボリューム 10K(Bカーブ) x1 *コンデンサ 100μF x1 *タクトスイッチ x1 *ブレッドボードとジャンパワイヤ *USB 接続ケーブル	今回のアプリケーションノートではブレッドボード上で試験を行っています。 配線図を元にパーツを配置して下さい。
Processing (ver1.2.1 以降)	下記のページから Processing 開発環境をダウンロードして下さい。 http://processing.org/

3 セットアップ

3.1 UIOUSB 設定

LED 出力や タクトスイッチ入力、A/D 変換入力を行うための UIOUSB の設定を行います。

UIOUSB設定項目

項目名称	設定内容	設定用 UIOUSBコマンド
I/O ポート入出力設定	ビット0 を入力、それ以外を全て出力に設定	“dcfg 0x01”
入力ポートのプルアップ	プルアップを有効に設定する	“pullup 1”
A/D 変換入力の変化イベント設定	チャンネル #0 のみ A/D 変換値が 9 以上変化した場合にイベントを送信する。 それ以外の A/D チャンネルの変化イベントは使用しない。	“ad_margin0 9” “ad_margin1 0” “ad_margin2 0” “ad_margin3 0”
入力ポートの変化イベント設定	ビット0の入力値が変化した場合にイベントを送信する	“change_detect 0x01”
コンフィギュレーション保存	設定内容を CPU 内蔵の EEPROM に保存	“save”

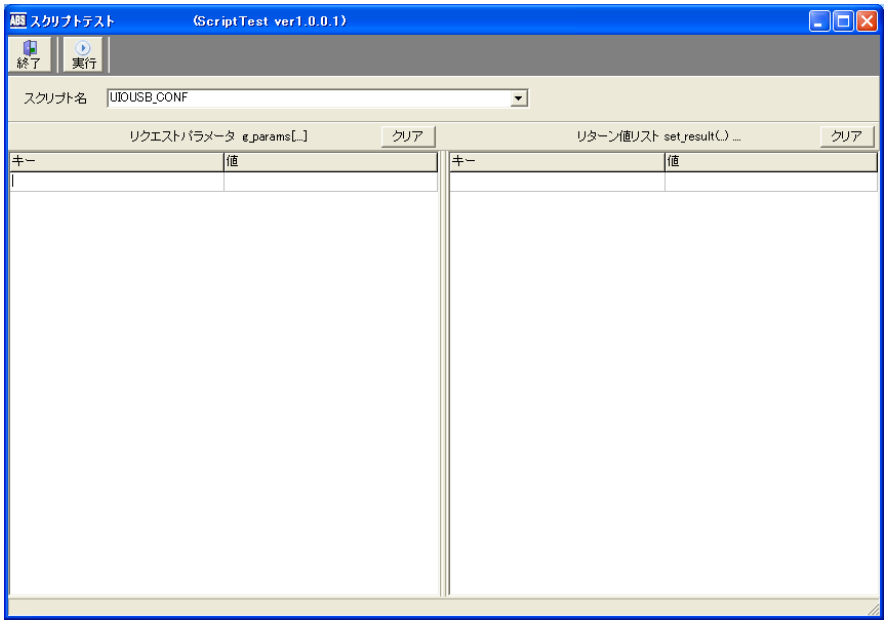
3.1.1 UIOUSB_CONF スクリプト作成

下記のスクリプトを作成して、UIOUSB の設定を行います。

```
file_id = "UIOUSB_CONF"
--[[
*********************************************************************************
  UIOUSB デバイスに下記の初期設定を行う
    I/O ポート  bit#0     入力 & プルアップ
                bit#1..#7 出力
    I/O ポート bit#0 の値が変化した場合にイベント送信
    A/D #0 変換値が 9 以上変化した場合にイベント送信
*********************************************************************************
]]
------------------
-- BEGIN SCRIPT --
------------------
log_msg("start..",file_id)
if not uio_command("dcfg 0x01") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("ad_margin0 9") then error() end
if not uio_command("ad_margin1 0") then error() end
if not uio_command("ad_margin2 0") then error() end
if not uio_command("ad_margin3 0") then error() end
if not uio_command("change_detect 0x01") then error() end
```

```
-------------------------------------------------------------------------------
-- コンフィギュレーションを EEPROM に書き込む コマンド実行時間が100ms 以上かかるので
-- タイムアウト検出をデフォルトの 100ms から 1000ms に変更してコマンド実行する
-------------------------------------------------------------------------------
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
------------------
-- END SCRIPT   --
------------------
```

ファイル名(UIOUSB_CONF.lua) で DeviceServer のスクリプトフォルダ
“C:¥Program Files¥AllBlueSystem¥Scripts” 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、UIOUSB の設定が行われます。

```
2011/02/01 21:59:52 falcon      UIOUSB_CONF          0 start..
2011/02/01 21:59:52 falcon      UIOUSB_CONF          0 end.
```

UIOUSB 設定や、UIOUSB イベントの詳細については ”UIOUSB ユーザーマニュアル”を参照して下さい。

http://www.allbluesystem.com/UIOUSB/UIOUSB_users.pdf

4　　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。

4.1　　UIOUSB_EVENT_DATA スクリプト作成

DeviveServer で UIOUSB デバイスからイベントデータ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イベント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。

CHANGE_DETECT イベントを受信した場合には、UDP パケットを Processing が動作している PC にポート番号 8091 で送信します。デフォルトでは “localhost” が送信対象になっていますので Processing が動作するPC のホスト名に変更してください。UDP のポート番号は Processing 側のスケッチで記述する UDP サーバーのポート番号と一致している必要があります。Processing が動作する PC にファイヤーウォールを設置している場合には、該当するポート番号とプロトコル(UDP) を通すように設定してください。

ADVAL_UPDATE イベントを受信した場合も同様に UDP パケットを送信します。UDP パケットには現在の A/D 変換値を文字列形式でつなげてあります。Processing 側のスケッチでこの A/D 変換値を読み込んでボールの大きさをリアルタイムに変更します。

```
file_id = "UIOUSB_EVENT_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。
```

```
********************************************************************************
UIOUSB_EVENT_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
--------------------------------------------------------------------------------

COMPort                 イベントを送信した UIOUSB デバイスのポート名          "COM3"
EVENT_DATA_COUNT        UIOUSB EVENT データカラム数                          2
EVENT_DATA_<Column#> UIOUSB EVENT データ値(ASCII 文字列)
EVENT_DATA_1 は常にイベントプリフィックス文字列を表す                        "$$$"
EVENT_DATA_2 はイベント名が入る                                             "SAMPLING"
EVENT_DATA _3以降のデータはイベント毎に決められた、オプション文字列が入る
<Column#> には 最大、EVENT_DATA_COUNT まで 1から順番に
インクリメントされた値が入る。
]]

log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id)

------------------------------------------------
-- Ballbouncing_xxxxx_xxxxx.pde Processing デモ用
------------------------------------------------
if g_params["EVENT_DATA_2"] == "ADVAL_UPDATE" then
    if bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 then
        ------------------------------------------------
        -- Ballbouncing_xxxxx_UDP.pde Processing デモ用
        ------------------------------------------------
        if not udp_send_data("localhost",8091,"AD" .. g_params["EVENT_DATA_4"]) then error() end
    end;
end;

if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
    if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) and
        (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then
        ------------------------------------------------
        -- Ballbouncing_xxxxx_UDP.pde Processing デモ用
        ------------------------------------------------
        if not udp_send_data("localhost",8091,"SW" .. g_params["EVENT_DATA_4"]) then error() end
    end;
end;
```

ファイル名(UIOUSB_EVENT_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。

4.2 GET_AD0 スクリプト作成

現在の A/D #0 の値を取得してリターンパラメータに設定するスクリプトを作成します。

スケッチからボールの大きさの初期値を決めるために、このスクリプトが呼び出されます。

```
file_id = "GET_AD0"
--------------------------------
-- UIOUSB AD#0 の値を取得する
--------------------------------
log_msg("start..",file_id)
stat,ad = uio_ad()
if not stat then error() end
script_result(g_taskid,"AD0",ad[1])
```

ファイル名(GET_AD0.lua) で DeviceServer のスクリプトフォルダ

“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

4.3 PUT_IO スクリプト作成

I/O ポートの指定されたビット値を 0 または 1 に更新します。

ビット値とポートの更新値は、スクリプトパラメータ g_params["BIT"]と g_params["VAL"] で、Processing のスケッチから渡されます。

```
file_id = "PUT_IO"
---------------------------------------
-- UIOUSB DIO の値を更新する
-- g_params["BIT"] bit#
-- g_params["VAL"] ポート値 '1' or '0'
---------------------------------------
log_msg("start..",file_id)
if g_params["BIT"] and g_params["VAL"] then
    if not uio_don(tonumber(g_params["BIT"]),(g_params["VAL"] == "1")) then error() end
end
```

ファイル名(PUT_IO.lua) で DeviceServer のスクリプトフォルダ

“C:¥Program Files¥AllBlueSystem¥Scripts” に保管します。

4.4 PERIODIC_TIMER スクリプト作成

今回のアプリケーションノートでは、Processing のスケッチから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。スケッチからは、予め DeviceServer に作成されたセッション情報を利用することで、ログイン操作をなくした簡単な仕組みになっています。

DeviceServer 側で認証用に予め用意しておくセッション情報を、DeviceServer 起動時に自動的に作成するための記述を PERIODIC_TIMER スクリプトに作成します。
PERIODIC_TIMER スクリプトは、DeviceServer で１分ごとに起動されますので、一回だけセッションを作成するための動作を記述します。 Processing のスケッチ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トークン文字列と同一のものを指定してください。

共有変数 “STARTUP_SCRIPT” は、一回だけの実行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"
------------------
-- BEGIN SCRIPT --
------------------
local stat,val
------------------------------------------------------------
-- DeviceServer 起動時に一回だけスクリプトを実行する
------------------------------------------------------------
stat,val = get_shared_data("STARTUP_SCRIPT")
if not stat then error() end
if val == "" then
    if not inc_shared_data("STARTUP_SCRIPT") then error() end
    ----------------------------
    -- 起動時に一回だけ実行される
    ----------------------------


    ------------------------------------------------------------
    -- セッショントークンを自動的に作成する
    -- WebAPI でログイン認証を省略してアクセスする場合に利用する
    -- セッショントークン文字列を変更するときは、
    -- WebAPI を使用する側の設定も変更すること
    ------------------------------------------------------------
    local token
    stat,token= create_session("1234",true)
    if not stat then error() end
```

```
end
```

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。

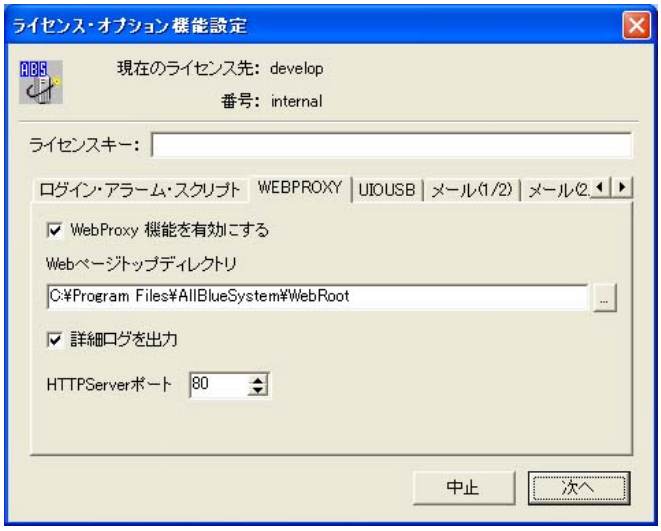
5 DeviceServer の WebProxy (WebAPI)設定

DeviceServer には HTTP サーバー機能と、Web API 機能を持った WebProxy モジュールが内蔵されています。インターネットやイントラネットから HTTP プロトコル経由でDeviceServer の機能を使用する場合に、このWebProxyモジュールを使用します。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。デフォルトでは 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。例えば 8080 等のポート番号に変更します。

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 Processing プログラム設定

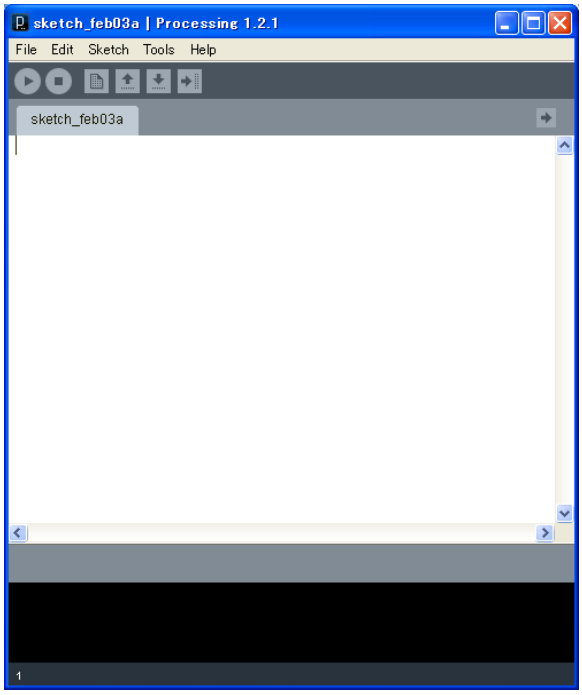
6.1 Processing インストール

始めに Processing 開発環境をインストールします。下記の URL にアクセスして Processing をダウンロードします。

http://processing.org/

ダウンロードしたアーカイブを解凍してProcessing を起動すると下記の様な画面になります。

また、スケッチの保存フォルダがマイドキュメント中に “Processing” フォルダとして作成されるのを確認して下さい。



6.2 スケッチで使用するライブラリファイル設定

ボールアニメーションで使用するスケッチで使用するライブラリを設定します。

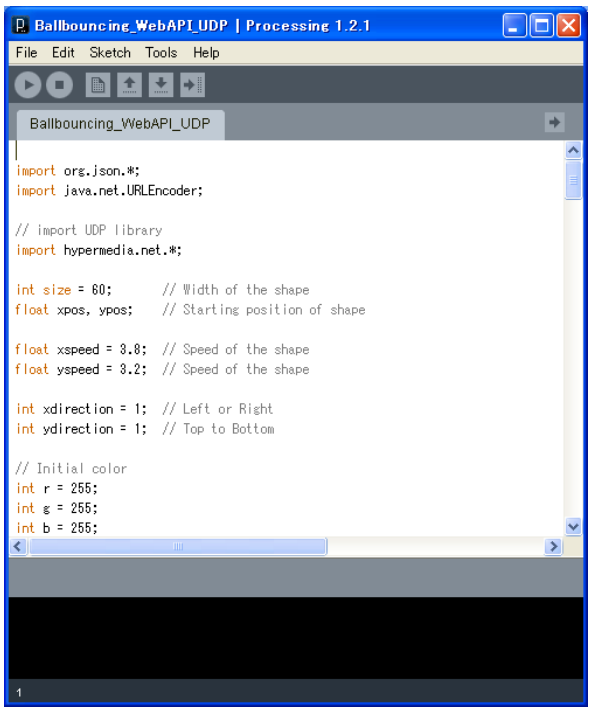
一旦 Processing プログラムを終了してから、APNOTE12の 添付ファイルに含まれる Processing フォルダ中の全てのファイルを、マイドキュメントの中にある Processing フォルダに全てコピーします。

コピーした libraries フォルダ中には、UDP ライブラリと json ライブラリが含まれています。既に該当するライブラリを導入している場合には、添付ファイルからコピーする必要はありません。

6.3 スケッチ Ballbouncing_WebAPI_UDP.pde 設定

Processing フォルダに前項の手順で ANOTE12 の添付ファイルがコピーされていると、スケッチファイル “Ballbouncing_WebAPI_UDP.pde” も同時に Ballbouncing_WebAPI_UDP フォルダにコピーされています。

Processing を起動してメニューから “File” -> “Sketchbook” -> “Ballbouncing_WebAPI_UDP” を選択してスケッチをロードして下さい。



この状態で、スケッチは自由に変更することが出来ますので、DeviceServer のホスト名と WebProxy で設定した HTTP サーバーポート番号を server_host_url 変数に設定してください。デフォルトは下記の用になっています。

String server_host_url = "http://localhost:8080";

また、PERIODIC_TIMER スクリプト中で設定した DevicveServer の認証用 セッショントークンは session_token 変数に設定します。もし PERIODIC_TIMER スクリプトで作成するセッショントークン文字列を変更した場合には、この部分も変更が必要です。

String session_token = "1234";

```
import org.json.*;
import java.net.URLEncoder;
// import UDP library
import hypermedia.net.*;

int size = 60;       // Width of the shape
float xpos, ypos;    // Starting position of shape
float xspeed = 3.8;  // Speed of the shape
float yspeed = 3.2;  // Speed of the shape
int xdirection = 1;  // Left or Right
int ydirection = 1;  // Top to Bottom
```

```
// Initial color
int r = 255;
int g = 255;
int b = 255;

boolean bit1_val = false;
boolean bit2_val = false;
boolean bit3_val = false;
boolean bit4_val = false;

UDP udp;  // define the UDP object

// ----------------------------------------------------------------
// DeviceServer設定
//
// * 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
//
//String server_host_url = "http://your_DeviceServer_public_url:80";
String server_host_url = "http://localhost:8080";
//String server_host_url = "";
//
// * SessionToken 認証を省略して既存のセッショントークンを使用して
//   DeviceServer にアクセスする
//
String session_token = "1234";
//

// ----------------------------------------------------------------
// DeviceServer のスクリプトを起動する (WebAPI 経由でリクエストを発行する)
// スクリプトへのリクエストパラメータを指定して、スクリプト中で設定されたリプライデータを
// 受信する
boolean ExecuteScriptWebAPI(String scriptName,HashMap reqParams,HashMap rplParams){
    try {
        String url = server_host_url + "/command/json/script?name=" + scriptName + "&session=" +
session_token;
        // construct the request url with parameters
        Iterator i = reqParams.entrySet().iterator();  // Get an iterator
```

```
        while (i.hasNext()) {
            Map.Entry me = (Map.Entry)i.next();
            url = url + "&" + URLEncoder.encode((String)me.getKey(),"UTF-8") + "=" +
URLEncoder.encode((String)me.getValue(),"UTF-8");
        }
        // execute the script on DeviceServer
        String result = join(loadStrings(url), "");
        // parse reply JSON and construct a reply HashMap
        JSONObject result_all = new JSONObject(result);
        if(result_all.getString("Result").equals("Success")){
            JSONObject result_params = result_all.getJSONObject("ResultParams");
            for (Iterator j = result_params.keys(); j.hasNext();) {
              String key = (String)j.next();
              String val = result_params.getString(key);
              rplParams.put(key, val);
            }
        } else {
            throw (new Exception((String)result_all.getString("ErrorText")));
        }
        return true;
    } catch(Exception e) {
        println("ExecuteScript:*EXCPTION* " + e.getMessage());
        return false;
    }
}

// DeviceServer の "PUT_IO" スクリプトを起動して UIOUSB デバイスの
// I/O ポート値を変更する
void update_io(int bit,boolean val){
    HashMap req = new HashMap();
    HashMap rpl = new HashMap();
    req.put("BIT",  str(bit));
    req.put("VAL", val ? "1" : "0");
    ExecuteScriptWebAPI("PUT_IO",req,rpl);
}

void setup()
{
  size(400, 400);
```

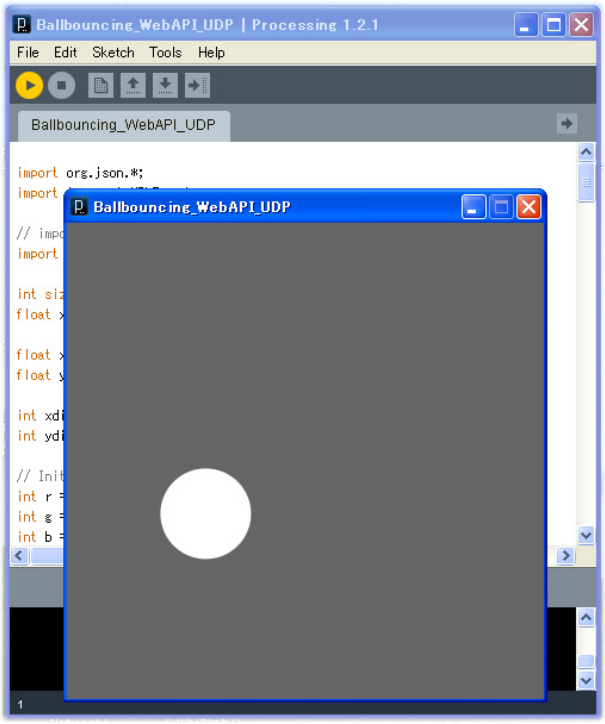
```
  noStroke();
  frameRate(60);
  smooth();
  // Set the starting position of the shape
  xpos = width/2;
  ypos = height/2;

  udp = new UDP( this, 8091 );
  //udp.log( true );                      // <-- printout the connection activity
  udp.listen( true );

  // ADO の値を取得して、shape の初期サイズに設定する
  HashMap req = new HashMap();
  HashMap rpl = new HashMap();
  ExecuteScriptWebAPI("GET_ADO",req,rpl);
  size = int((String)rpl.get("ADO"));
}

void draw()
{
  background(102);

  // Update the position of the shape
  xpos = xpos + ( xspeed * xdirection );
  ypos = ypos + ( yspeed * ydirection );

  // Test to see if the shape exceeds the boundaries of the screen
  // If it does, reverse its direction by multiplying by -1
  if (xpos > width-size) {
    xdirection *= -1;
    bit1_val = !bit1_val;
    update_io(1,bit1_val);
  }

  if (xpos < 0) {
    xdirection *= -1;
    bit2_val = !bit2_val;
    update_io(2,bit2_val);
  }
```

```
  if (ypos > height-size ) {
    ydirection *= -1;
    bit3_val = !bit3_val;
    update_io(3,bit3_val);
  }

  if (ypos < 0) {
    ydirection *= -1;
    bit4_val = !bit4_val;
    update_io(4,bit4_val);
  }

  // Draw the shape
  fill(r,g,b);
  ellipse(xpos+size/2, ypos+size/2, size, size);
}

void receive( byte[] data, String ip, int port ) {  // <-- extended handler
  String message = new String( data );

  // "AD" で始まる AD0 変換値を受信した場合は shape のサイズをAD0 の値に基づいて変更する
  if (message.substring(0,2).equals("AD")) {
    size = int(message.substring(2)) / 2;
  }
  // "SW" で始まるスイッチ入力を受信した場合には shape の色をランダムに変更する
  if (message.substring(0,2).equals("SW")) {
    r = int(random(255));
    g = int(random(255));
    b = int(random(255));
  }
}
```

7 動作確認

DeviceServer に接続した UIOUSB の配線を確認した後、Processing を実行します。UIOUSB デバイスを PC から取り外した場合には DeviceServer の再起動が必要になります。この場合には、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。設定項目は変更しないで、”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 が再起動します。

Ballbouncing_WebAPI_UDP スケッチをロードして RUN ボタンを押してスケッチを実行します。



ボールアニメーションが表示されます。ボールの初期値が大きすぎるとボールが隅で引っかかった状態になっていますので、その場合にはボリュームを回して大きさを調整してください。

ボールが跳ね返る毎に、対応するLED が反転します。タクトスイッチを押すとボールの色がランダムに変化して、ボリュームを回すとリアルタイムに大きさが変化するのが確認できます。

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.1.0 2011/2/3

初版作成