Home Automation Using Flyport

CSM-0E2-0SW10-0-V0, 2012-03-30

Document Information	
Project Name	0E2
Customer Name	CISIUM
Product Name	0E2
Model Year	2012
Authorized Audience	Internal
Abstract	This document is a guideline for the setup of home automation project involving the flyport module, D-link wireless Camera and a Local Webserver to control flyport through Wi-Fi connection.
Theme DN	CSM-0D0-0D6A-0-V11
Template DN	CSM-0D0-0D6A-0-V11

CISIUM

Contents

List of Figures

List of Tables

Revision History

The next table summarizes the changes of the present document based on the associated version number. The table is arranged form the newest version to the oldest.

Ver.	Date (YYYY-MM-DD)	Change Description	Author	Approved
0	2012-03-20	Newly established.	J. Aparecio	R. Bartolomé

Table 1: Document version

1. Introduction

1.1. Purpose of This Type of Document

This is a guideline document.

1.2. Abstract

This document is a guideline for the setup of home automation project involving the flyport module, D-link wireless Camera and a Local Webserver to control flyport through Wi-Fi connection.

1.3. Maintenance

The maintenance and ownership of this document is by the Software Engineering Department. The Manager of the Software Engineering Department can audit and review the document in any time.

2. Scope

The scopes of this document are to discuss the processes involved in making and running a home automation program using the flyport; to discuss the connection of two different web servers and how they communicate; and embedding the IP camera to the project to have a real time monitoring of the devices.

3. Hardware Pre-Requisites

The following hardware specifications are used in developing this project.

- Desktop Computer
  - Operating System: Windows 7 x64 Professional
  - Processor: Pentium(R) Dual-Core CPU E5300@2.6GHz
  - Memory: 4GB
- D-Link Camera
  - Model DCS-932L
  - Body No: 30228025
- Flyport
  - HW 1.02
  - FW BTS 1.0.0
  - Module Serial Number: 895
- Flyport nest

3.1. Set up

Additional material:

- 3 green LED
- 2 tactile switches
- 1 potentiometer
- 4 resistors

In the next image we can see the setup of the project:

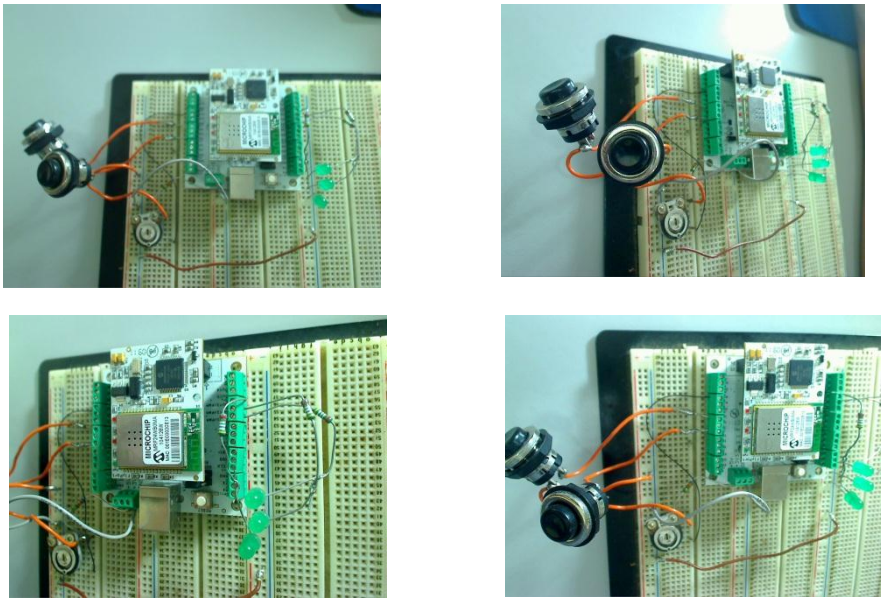

Figure 1: Set up

3.2. Schematic

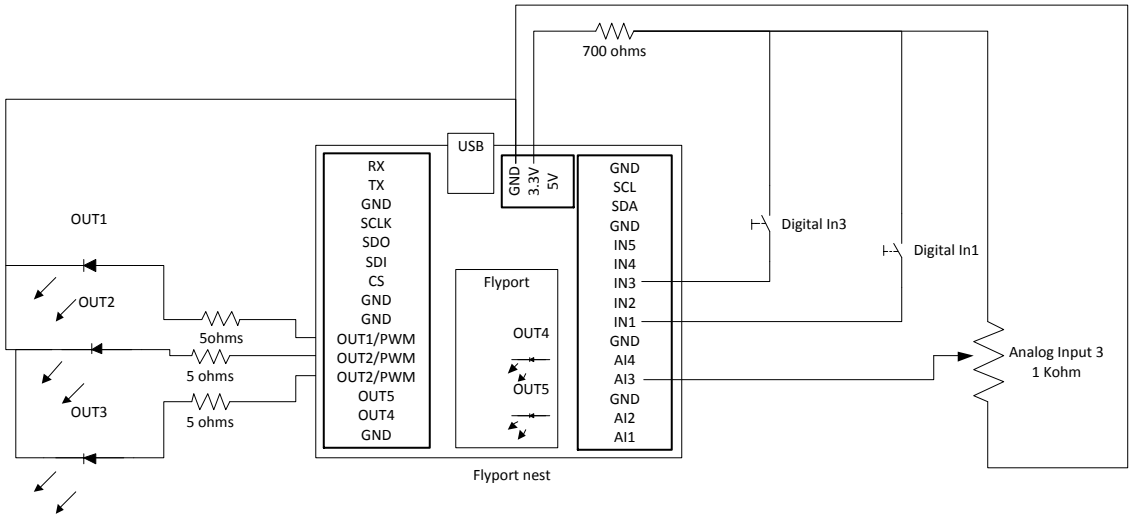

Figure 2: schematic

4. Software Pre-Requisites

The project requires the following software to be installed in the host machine.

- **Microsoft .NET framework 4.0**
  Software installer can be found in software repository. But if you found the latest version available in the website below, please approach the Software Manager directly.
  "path1"
- **C30 Microchip Compiler (3.24 Lite Version)**
  Software installer can be found in software repository. But if you found the latest version available in the website below, please approach the Software Manager directly.
  "**path2**"
- **OpenPicus IDE 2.1**
  Software installer can be found in software repository. But if you found the latest version available in the website below, please approach the Software Manager directly.
  "**path2**"
- **WAMP SERVER 64 bits (recommended)**
  Software installer can be found in software repository. But if you found the latest version available in the website below, please approach the Software Manager directly.
  "**path3**"

4.1. Microsoft .NET Framework 4.0

The .NET Framework is Microsoft's comprehensive and consistent programming model for building applications that have visually stunning user experiences, seamless and secure communication, and the ability to model a range of business processes.

To install the Microsoft .NET Framework 4.0, you can have the following options:

- Go to the website "**path1**" and click the "Install it now" button. This button will download the file and install directly to the host machine and follow installation instruction.
- Download the executable file in the link below and install it into the host machine.
  "**path4**"

4.2. C30 Microchip Compiler (3.24 Lite Version)

The MPLAB® C Compiler for PIC24 MCUs and dsPIC DSCs (also known as MPLAB C30) is a full-featured ANSI compliant C compiler for the Microchip 16-bit devices: PIC24, dsPIC30F and dsPIC33F. MPLAB C is a 32-bit Windows® console application as well as a fully integrated component of Microchip's MPLAB Integrated Development Environment (IDE), allowing source level debugging with the MPLAB REAL ICE™ Emulator, MPLAB ICD 2 In-Circuit Debugger and MPLAB SIM Simulator.

To install the C30 Microchip Compiler (3.24 Lite Version), download the executable file in this link "**path2**", click "C30 Microchip compiler (3.24 Lite version)" and follow installation instruction.

4.3. OpenPicus IDE 2.1

OpenPicus IDE 2.1 is a Windows application that allows the user to program and manage all the functionalities of the OpenPicus Flyport. It extends to basic functions of the IDE 2.0 program, integrating some powerful tool to ease the developing of an entire Flyport project.

For more details with regards to OpenPicus IDE 2.1, please refer to the PDF file located in "**path5**/ DOC Flyport Programmer's Guide Rel 1.2.pdf"

To install OpenPicus IDE 2.1, download the executable file in this link "**path2**" and follow installation instructions.

4.4. WAMP SERVER (recommended)

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database.

To install Wamp Server, download file from this link "**path2**", choose 64bit installer (for 64 bits operating systems) and follow installation instructions.

Note: Installation of Wamp server uses the port 80 as the default; you might encounter some errors with regards to port especially if you are using a Skype. Skype also uses port 80 and 443 as the default port for incoming connection. To solve the problem, change the port default settings of the Skype. Go to "**Tools** -> **Options** -> **Advanced** -> **Connection**" then uncheck the "*use of port 80 and 443 as alternative for incoming connections*".

5. Projects References

The current manual is based to two software projects used as reference:

- Web site controller: CSM-0E2-0SW1B-0-V1.0
- Flyport Firmware: CSM-0E2-0SW1A-0-V1.0.0

Note: Website Controller is best performed in Google Chrome, some JavaScript issues are encountered in running the project in IE and Firefox. Basically, project demo is not created in cross-browser compatible.

6. Getting Started

6.1. Overview

The Home Automation Project using the flyport development board aimed to have an automated controller of home devices using the cloud. This can be achieved with the use of Wi-Fi technology and the WWW.

The project involves three development processes;

- Develop and create a program using the openPICUS IDE. This program is responsible for controlling the devices as well as handle requests from the website (from the other webserver).
- Create a website on local server or any host. This serves as the control interface of the user. This must communicate to the flyport webserver.
- Install a wireless camera. Camera serves as a look out of the actions made by both website and the flyport (or real time monitoring).

6.2. Creating Flyport Firmware

This chapter assumes that you have already installed the openPicus IDE 2.1.

6.2.1. OpenPicus Integrated Development Environment.

OpenPicus IDE is a Windows application that allows the user to program and manage all the functionalities of the OpenPicus Flyport. To have a better understanding of the OpenPicus Environment, please see the PDF file,"**path5/** DOC- IDE 2.1 Flyport user guide - rev1.0.pdf".

6.2.2. OpenPicus Three Major Files

These files are default files in creating new project in OpenPicus.

6.2.2.1. TaskFlyport.c

Basically, this file handles the functions that manipulate the flyport in/out pins. Pin initialization, declaration and execution are some of the methods found here.

Based on the set-up, there are two push buttons attached to the flyport. The push buttons main goal is just to toggle the current state of the pins where it is connected. Buttons are connected to IN1 and IN3, both are input pins. To toggle the current state of the pins, code below shows how it is done.

"`void FlyportTask()`" is the function that is executed upon loading the firmware. It is like the main() function of other languages. Inside this function, it checks the connectivity settings of the WIFI and write the findings in the UART through UARTWrite() function. You can add anything in this function depending on your needs. IOInit() is a built-in function that assigned the pins initial value. Iobuttonstate() check the current button status.

```
#include "taskFlyport.h"

//this is the function that is read upon //executing the firmware
void FlyportTask()
{
	//check wifi status
	WFConnect(WF_DEFAULT);
	while (WFStatus != CONNECTED);
	UARTWrite(1,"Flyport connected... hello world!\r\n");

	//Initialize initial value of the PINS
	IOInit(D1In,INUp);		//IN1 initialized as input with pull-up res
	IOInit(D2In,INdown);		//IN2 initialized as input with pull-dwn res
	IOInit(D3In,INdown);		//IN3 initialized as input with pull-dwn res


while(1)
{
	//while connected, it check if a button is pressed and toggle the pins
	if (iobuttonstate(D1In) == PRESSED)
	{
		IOPut(D1Out , TOGGLE); //if D1In is pressed, D1out will toggle
	}
	if (iobuttonstate(D2In) == PRESSED)
	{
		IOPut(D2Out , TOGGLE); //if D2In is pressed, D2out will toggle
	}
	if (iobuttonstate(D3In) == PRESSED)
	{
		IOPut(D3Out , TOGGLE); //if D3In is pressed, D3out will toggle
	}

}
}
```

6.2.2.2. WF_Events.c

It is responsible in setting the Wi-Fi connectivity and settings.

6.2.2.3. HTTPApp.c

This file handles the receive request and response. This is also serves as the webserver of flyport. Two major methods that are common used are the POST and GET method this are responsible of handling request and process response. There are global variables also that are responsible of holding the return value. See the PDF in “**path5**/ DOC Flyport Programmer's Guide Rel 1.2.pdf”.

GET Method: The get method is intended to receive small data. This code takes care of activating the LED. The file "flyport.cgi" in the flyport defines the variables (out) that will be read by the flyport and write by the website controller.

It is simple and easy to use but can handle only limited bytes of data. Code below shows how GET method handles the request from the website controller. Refer to this PDF "**path5**/Microchip TCPIP Stack Programming Guide - Microchip.pdf" for better understanding for the GET method.

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
	BYTE *ptr;
	BYTE filename[20];

	// Load the file name
	// Make sure BYTE filename[] above is large enough for your longest name
	MPFSGetFilename(curHTTP.file, filename, 20);

	// If it's the LED updater file
	if(!memcmppgm2ram(filename, "flyport.cgi", 8))
	{
		// Determine which LED to toggle
		// out is defined inside of flyport.cgi
		ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"out");
		// Toggle the specified LED
		switch(*ptr)
		{
			case '0':
				IOPut(o1,toggle);
				break;
			case '1':
				IOPut(o2,toggle);
				break;
			case '2':
				IOPut(o3,toggle);
				break;
			case '3':
				IOPut(o4,toggle);
				break;
			case '4':
				IOPut(o5,toggle);
				break;
		}

	}
	return HTTP_IO_DONE;
}
```

POST Method: the post method is intended to receive a big mount of data. This code takes care of receiving data form the text box located in the website controller. The file "index.html" received in data from the website controller, the website controller will send the data to “index.html” in the flyport using a post Java script method.

Much complicated but can handle unlimited number of bytes, usually useful in handling string data or request containing a large data. Refer to this PDF “**path5**/Microchip TCPIP Stack Programming Guide - Microchip2.pdf” for better understanding of the POST method.

```
HTTP_IO_RESULT HTTPExecutePost(void)
{
	BYTE name[20], item;
	WORD len;

	// Load the file name
	MPFSGetFilename(curHTTP.file, name, 20);
	// Make sure it's the index.htm page
	if(strcmppgm2ram((char*)name,(ROM char*)"index.htm") != 0)
	return HTTP_IO_DONE;

	// Loop while data remains
	while(curHTTP.byteCount)
	{

		// Check for a complete variable
		len = TCPFind(sktHTTP, '&', 0, FALSE);
		if(len == 0xffff)
		{
			// Check if this is the last one
			if(TCPIsGetReady(sktHTTP) == curHTTP.byteCount)
			len = curHTTP.byteCount - 1;
			else // Wait for more data
			return HTTP_IO_NEED_DATA;
		}

		if(len > HTTP_MAX_DATA_LEN - 2)
		{// Make sure we don't overflow
			curHTTP.byteCount -= TCPGetArray(sktHTTP, NULL, len+1);
			continue;
		}
		len = TCPGetArray(sktHTTP,curHTTP.data, len+1);
		curHTTP.byteCount -= len;
		curHTTP.data[len] = '\0';
		HTTPURLDecode(curHTTP.data);

		// Figure out which variable it is
		// variable is declared in HTTPApp.c
		if(memcmppgm2ram(curHTTP.data,(ROM void*)"firstname", 9) == 0)
		{	// A name was found
			item = curHTTP.data[10] - '0';
			if(item > MAX_PRODUCTS)
			continue;
		memcpy((void*)profile[0].firstname,(void*)&curHTTP.data[10], 100);
		}

	}
	return HTTP_IO_DONE;
}
```

Global Variable: this part of code is the mechanism to implement sending data from the flyport to the webserver controller. The flyport updates the global variables, and the website controller reads those variables.

Each global variable is defined in an XML file located in the Web pages of the flyport project. Because the webserver controller java script cannot read directly the XML we use a PHP file which take care of the reading of the XML file “outputxml_reader.php”

Global Variables are the one that holds the value return by the flyport webserver that later will be retrieve with the website controller. Refer to this PDF “**path5/**Microchip TCPIP Stack Programming Guide - Microchip3.pdf” for better understanding of the global variables.

In the example code below, the global variable is “out”. HTTPPrint_{global_variable}() is the syntax for giving value to the global variable “out”.

```
void HTTPPrint_out(WORD num)
{
	switch(num)
	{
		case 0:
			num = IOGet(o1);
			break;
		case 1:
			num = IOGet(o2);
			break;
		case 2:
			num = IOGet(o3);
			break;
		case 3:
			num = IOGet(o4);
			break;
		case 4:
			num = IOGet(o5);
			break;
		default:
			num = 0;
	}

	// Print the output
	TCPPut(sktHTTP, (num?'1':'0'));
	return;
}
```

6.2.3. Setting up TCP/IP Wizard

The TCP/IP wizard is the tool to properly configure the Flyport, the service available on the device, and the network configuration (like IP address, SSID, security). The network configuration is needed to make the Flyport run in the right way, and connect to the desired network; the selection of the services is needed to minimize the resources used by the TCP/IP stack. The TCP/IP wizard has many services offered to give users a lot of options to choose. The next items follow are the setting in running the project.

6.2.3.1. Services Selection

The project enables the following services;

- Webserver
- DHCP Client
- ICMP Client
- ICMP Server
- Post Method
- DNS Client
- Announce Service
- NetBIOS Name Service
- TCP Debug on UART1
- SNTP Client
- Remote Reboot Service

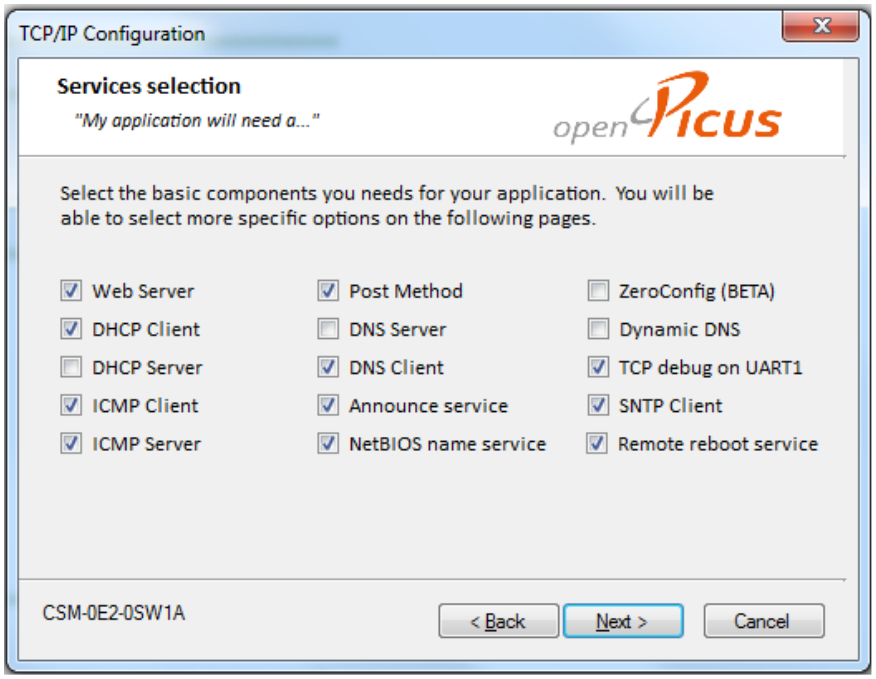

Figure 3: TCP/IP Wizard

6.2.3.2. Network Configuration

This contains the network parameters to select IP address of the device, subnet mask and etc.

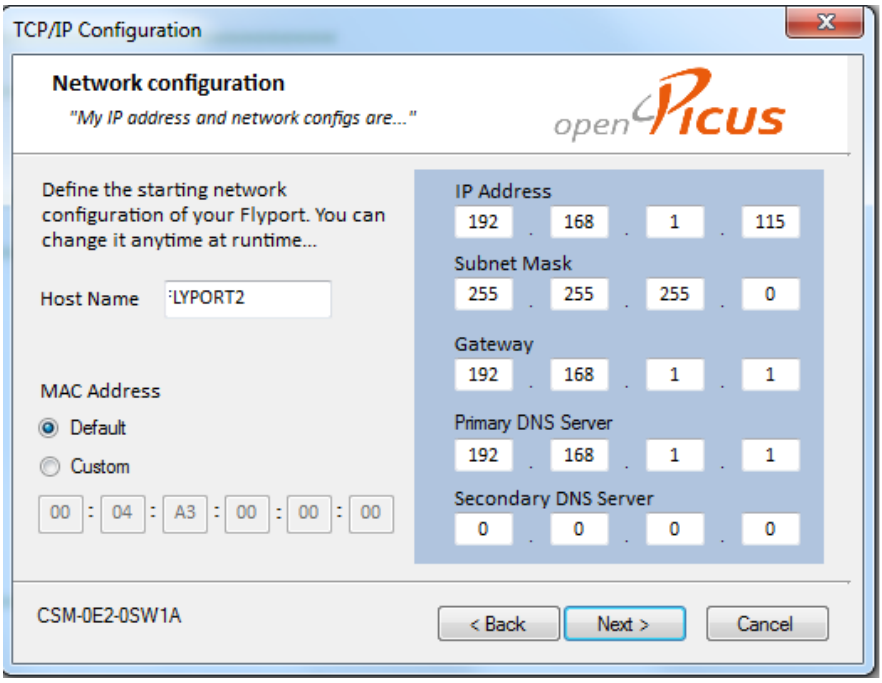

Figure 4: Network Configuration

- The HOST NAME will be the name of your flyport device.
  - This is very important in this project because the HOST NAME will be used to call the flyport webserver from the website controller. If you change the HOST NAME, you must change the HOSTNAME in the website controller which is “flyport2” as default.
- The IP address, Subnet mask, gateway and DNS server depends on the internet connection set-up.

6.2.3.3. Wireless Configuration

This page is the wireless connection set-up (Figure 3). This is used to configure wireless parameters (SSID, network type).

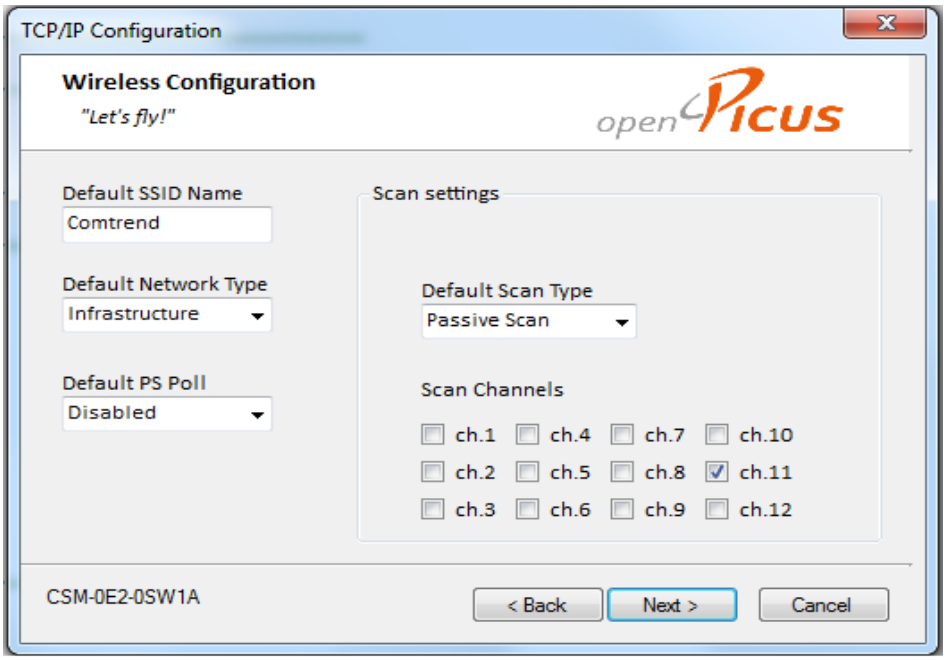

Figure 5: Wireless Configuration

- SSID is the name of the network to create or join to.
- Network type, if you want to make the connection ad hoc or infrastructure.
- PS Poll is the power save mode
- Scan type, if you don't know the channels you are to connect you can choose active otherwise, passive and select the exact channels.

6.2.3.4. Wireless Security

In this page, you have to select what type of security is set to connect to the Wi-Fi.

- PSK Phrase is the password of the Wi-Fi you are going to connect.

Note: Be sure to check the checkbox “Flyport will calculate Key from Passphrase (see figure 4)”.

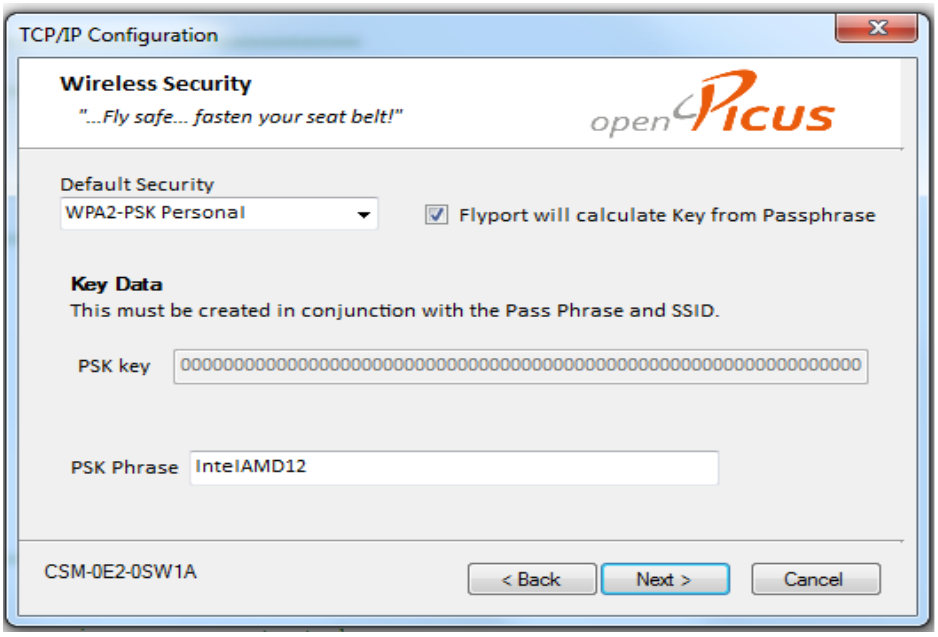

Figure 6: Wireless Security

The following steps are not touched. It is set as per default, just click next until “Finish”.

6.2.4. Importing Webpages

To run the built-in webserver of the Flyport, you need to import its html pages. Html pages usually located inside the folder from which the project is created. Normally, folder containing the html pages of the flyport is named “Web Pages”. It also includes other web scripting languages like javascript, CSS and xml files.

6.2.5. Compiling Project

Before loading to the device, you need to compile all the changes. You can compile the project by clicking the “Compile Project” menu (Figure 5).

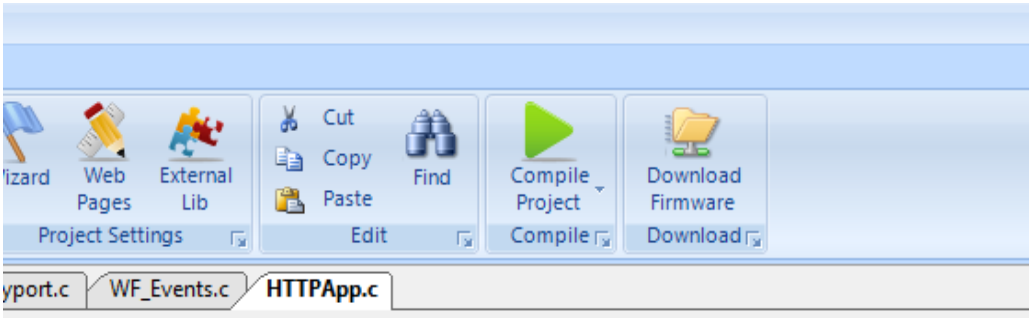

Figure 7: Compiling Project

6.2.6. Downloading Firmware to the Device

This assumes that the web pages are already imported and changes have already compiled.

To download firmware to the device, click “Download Firmware” menu (figure 6).

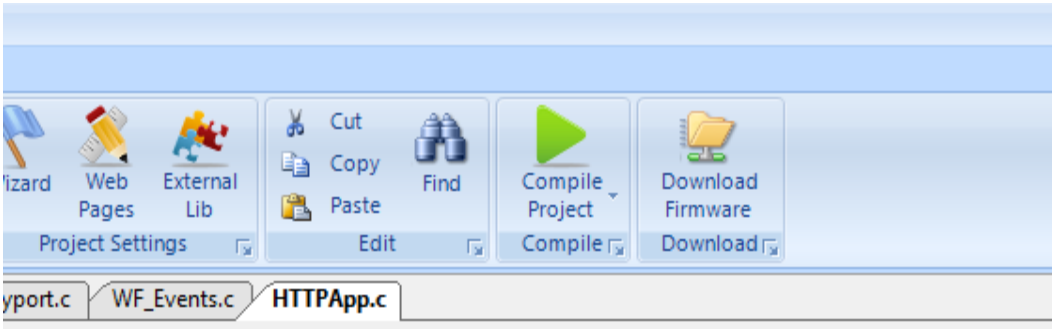

Figure 8: Downloading a Firmware

Upon clicking the menu, another window (Figure 7) appears wherein you need to set the port where the device is connected. Choose the right port or else it won’t download the firmware. Please ask the administrator if you don’t know the port number.

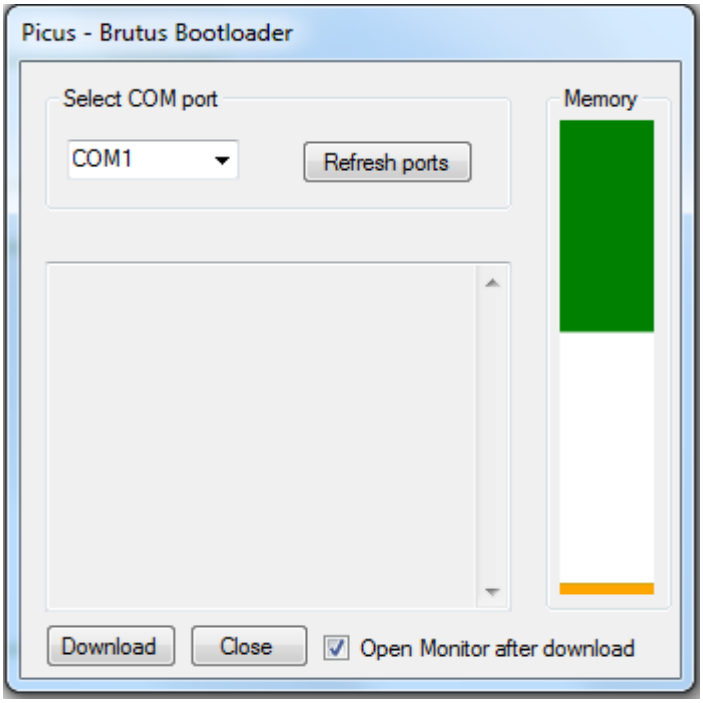

Figure 9: Bootloader Window

Once the firmware is already downloaded, see the serial monitor. It will display the flyport activity. It will show if the flyport is connected to the network or not. If it displays the designated IP of the device, the flyport is successfully connected. After that, go to your browser (Firefox,IE, Safari and etc.) and type in the address bar the HOST NAME or the IP address of the flyport. Example “FLYPORT2” since where using “FLYPORT2” as the HOSTNAME. If it displays any page, then you successfully created your flyport program.

Note: When connecting to the flyport using a wireless security, it usually takes several seconds to connect. Be patient.

6.3. Creating Website Controller

The main function of the website controller is to connect to the flyport and control the activity of the pins. Through internet you can control any devices that are connected to the flyport. This chapter assumes that you already have local server installed (WAMP Sever).

6.3.1. Web Server to Web Server (W2W) Communication

This communication refers to the sending requests and receiving responses from one server to another webserver. Website controller is in different webserver, for example, in a bluehost, localhost or any host and this will communicate to each built-in webserver of the flyport.

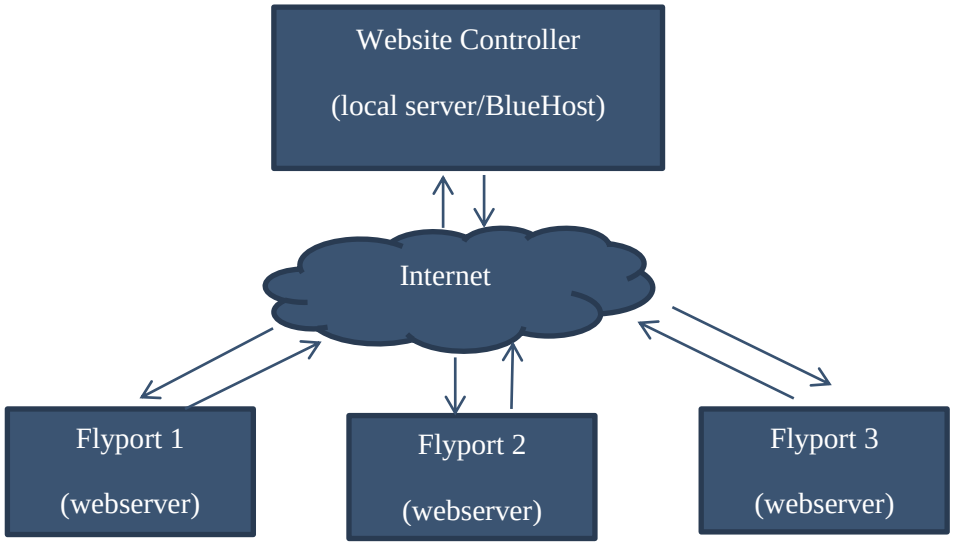

Figure 10 W2W Communication

6.3.1.1. Exchanging of Data from Two Webservers

JavaScript cannot directly access file from other webserver for some security issues. In this case, to communicate from localhost to flyport webserver, PHP is used as the tool. Below shows how data is transferred and receive by two webservers.

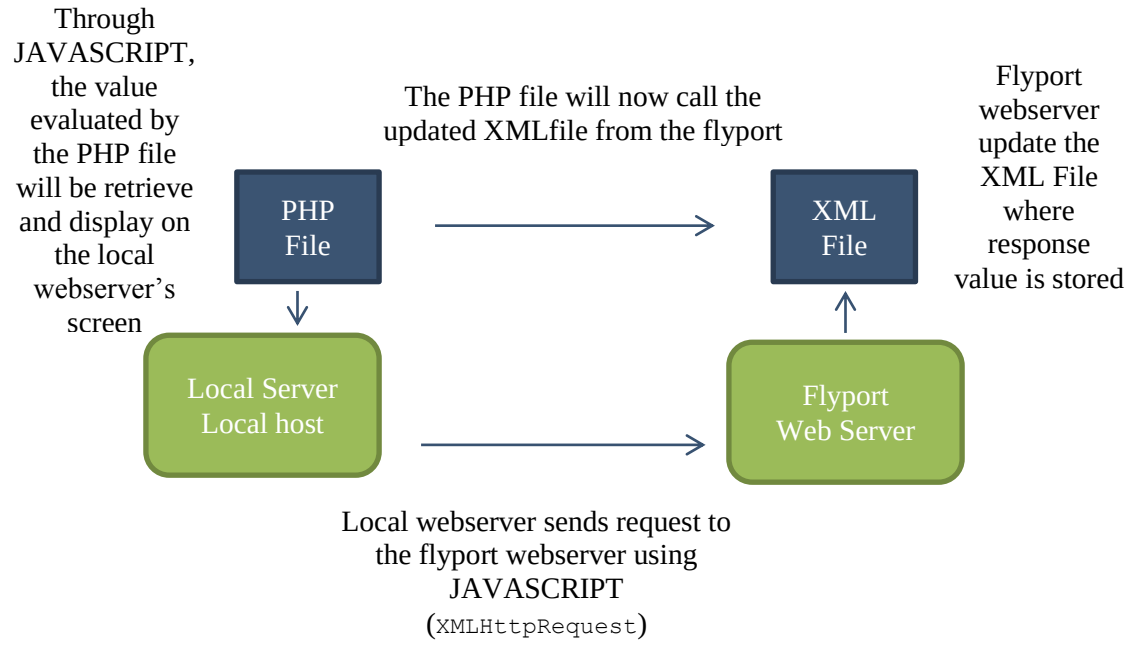

Figure 11: Exchanging of Data

6.3.2. Website Controller

This part will discuss how website controller is created and how it communicates with the flyport web server.

Website controller is no different from other websites. It can be done using any web development tools and languages or can be CMS like JOOMLA, WORDPRESS or Drupal as long as it work and it displays the exact output return by flyport.

In this project, the website controller is created using PHP, Javascript, Jquery, xml file and HTML. The following chapter shows the functions of each file.

6.3.2.1. Main Page (Request.html)

The main page is the landing page upon opening the website controller. It displays menus, buttons and option that you are able to do with your flyport.

In this page, there are five sections (see figure 9);

- Output section. This section has button, button status and a output response
  - Buttons – able to control the flyport output pins. If click, it toggles the flyport output pins and change its states.

  - Voltage Indicator – this will display the current voltage running in a specific output pins.
  - Output status- these are light buttons that are significant in identifying the result of the output pins. It lights when the output pin is lighted and off if not.
- Input section. This section able to display the responses from the input pins.
  - Input indicator- graphical representation of the return value from the input pins.
  - Input Value – the value itself
  - Input Status – this is an indicator that the data receive by the input pins is beyond the limit. Example, if the temperature is beyond 100C is will turn on.
- Camera section. This section will display the video stream from the camera. Also the button settings of the camera like zoom out, zoom in and etc.
- Additional button section. This is for some additional function to the flyport like sending strings to the flyport.
- Display result section. This will display the result coming from the flyport.

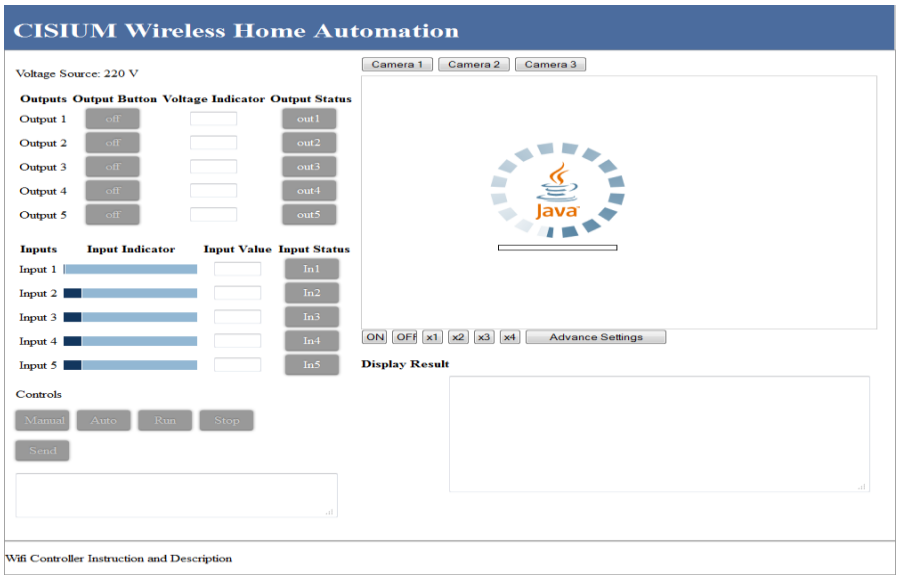

Figure 12 Main Page

6.3.2.2. Sending Request and Displaying Response in Output Section

This explains how requests are send to the flyport web server and how the page manages the response sent by the flyport.

The button “off” in output section is the button that tells the flyport what to do. Clicking it changes the pins state. When it is pressed, it calls the function “`led_onclick(0)`” as shown in the code below.

```
<a id="out_button0" class="button" onclick="led_onclick(0);">
```

"led_onclick(0)" is a javascript function that will instruct flyport webserver what to do. It has a parameter inside which determines the pins to be toggle. The parameter inside the function which is "0" is the identifier to which output pins to be light on/off. It could be "0, 1, 2, 3.., and so on" depends on the setup on the flyport.

```
/*found in request.html*/

function led_onclick(val){
       newAJAXCommand('http://flyport2/flyport.cgi?out='+val);
       $.get("outputxml_reader.php", function(data){
                     if(data[val] == 1){
                            $('#out'+val).css("background","#FC0");
                            $('#out_button'+val).html("on");
                     }else{
                            $('#out'+val).css("background","#999");
                            $('#out_button'+val).html("off");
                     }
});
}

/*found in mchp.js*/
function newAJAXCommand(url, container, repeat, data)
{
       // Set up our object
       var newAjax = new Object();
       var theTimer = new Date();
       newAjax.url = url;
       newAjax.container = container;
       newAjax.repeat = repeat;
       newAjax.ajaxReq = null;

       // Create and send the request
       if(window.XMLHttpRequest) {
         newAjax.ajaxReq = new XMLHttpRequest();
         newAjax.ajaxReq.open((data==null)?"GET":"POST", newAjax.url, true);
         newAjax.ajaxReq.send(data);
    // If we're using IE6 style (maybe 5.5 compatible too)
    } else if(window.ActiveXObject) {
         newAjax.ajaxReq = new ActiveXObject("Microsoft.XMLHTTP");
         if(newAjax.ajaxReq) {
             newAjax.ajaxReq.open((data==null)?"GET":"POST", newAjax.url, true);
             newAjax.ajaxReq.send(data);
         }
    }

    newAjax.lastCalled = theTimer.getTime();
    // Store in our array
    ajaxList.push(newAjax);
}
```

When this happens, two methods are executed;

- Sending of request
  "newAJAXCommand('http://flyport2/flyport.cgi?out='+val);"

"newAJAXCommand()" is a function that is responsible of sending the url "'http://flyport2/flyport.cgi?out='+val" to the flyport. This method gets contact with the "flyport.cgi" file inside the flyport and assigned the value from variable "val".

Note: "flyport2" in the url inside the newAJAXCommand() is the HOST NAME of the flyport. This is very useful especially in determining what flyport you are using and from what flyport you get you data.

- Managing response
  `$.get("outputxml_reader.php",function(data){}`

This method follows as soon as the request is sent; this method gets the value from the "`output.xml`" through "`outputxml_reader.php`" file. The values receive by the PHP file will be evaluated and displayed in the screen.

6.3.2.3. Displaying Data in the Input Section

In input section, there will be no sending of requests. This section will just get value from the flyport. The data in the flyport comes from input devices like sensors (pressure, temperature and etc.).

Since we want to display the real time value of the output from sensor, we need to get the value from time to time. And this can be done with Javascript. Javascript is capable of getting values from other pages without refreshing the page.

```
/*found in request.html*/

function update_input(){
	//update input section
	$.get("inputxml_reader.php", function(data){
		var result = data.split(" ");
		var wid;
		if(result){
			for(a=0; a<5; a++){
				$('#in'+a).val(result[a]);
				wid=result[a]/5;
				$('#bar'+a).width(wid);
			}
		}
	});
//update the output section
	$.get("outputxml_reader.php", function(data){
		if(data){
			for(i=0; i<5; i++){
				if(data[i] == 1){
					$('#out'+i).css("background","#FC0");
					$('#out_button'+i).html("on");
				}else{
					$('#out'+i).css("background","#999");
					$('#out_button'+i).html("off");
					}
				}
			}
	});
//update the textarea where message is sent
	$.get("post_reader.php", function(data){
		$('#display').html(data);
	});
}
setInterval( "update_input()", 500 );
```

"`function update_input()`" is the one responsible of getting values from the flyport and displaying it in the page. To make values updated, we use "`setInterval("update_input()", 500 );`" to call the function every 0.5 second. This loads the function and displays the date without refreshing the whole page.

6.3.2.4. Sending String Data to Flyport and Displaying the Response

Same with the idea in the output section, the only thing that differs is that it sends a string.

```
/*found in request.html*/

function confirm_send(){
	var data= document.getElementById('textarea').value;
	var url = "http://flyport2/index.htm";

	$.post(url, {firstname: data});
	$.get("post_reader.php", function(data){
			$('#display').html(data);
	});
	document.getElementById('textarea').value='';
}
```

The string inputs from <textarea> tags are assigned to the variable "data". This variable is sent to the defined URL using POST. Post method is used in this case because as describe in the Flyport manual, that only POST method can handle string data.

6.3.2.5. Getting Value from the XML file of Flyport Webserver

In this section, it will show how the website controller gets value from the XML file of the flyport webserver. Basically, since JavaScript is not capable of getting the data, PHP file is used.

Below is the code from "outputxml_reader.php". This file is responsible of getting the value from the output of flyport, specifically from "post_output.xml" file in the flyport.

First, we have a URL "`http://flyport2/output.xml`" calling the XML file from another webserver, evaluate using "`simplexml_load_file`". This XML file contains all the values from the flyport LED.

```
<?php
$url = 'http://flyport2/output.xml';
$xml = simplexml_load_file($url);
$output = array();
// get first book title
$output[0]=$xml->out0;
$output[1]=$xml->out1;
$output[2]=$xml->out2;
$output[3]=$xml->out3;
$output[4]=$xml->out4;
$output[5]=$xml->fname;
// show title
for($i=0; $i<6; $i++){
echo $output[$i];
}
?>
```

6.4. Installing IP Camera

IP camera is very suitable in real time monitoring especially when you are away from where the camera is installed. Through internet, you can monitor and view what is happening in the area. And this idea applied to the home automation to monitor the flyport's activity and responses.

You can use any IP camera present in the market, but this project used D-Link Camera DCS 932L model. This camera has WI-FI built-in module. To know more features in this camera, see PDF file in "**path5/**DCS-932L_A1_Manual_v1.00(NA)-EN.pdf"

To install the camera, install driver from the CD and follow installation instructions.

6.4.1. Embedding Camera Video Stream in the Website

To embed live stream from DCS932L camera, insert the following codes into your page.

```
<APPLET name="cvcs" CODEBASE="http://192.168.0.51:80" CODE="aplug.class"
WIDTH="440" HEIGHT="380">
   <param name="RemotePort" value=80>
   <param name="Timeout" value=5000>
   <param name="RotateAngle" value=90>
   <param name="PreviewFrameRate" value=2>
   <param name="DeviceSerialNo" value="YWRtaW46Y2lzaXVt">
</APPLET>
```

Make sure that you input the right IP address of the camera to display the video stream. This works well but if you can find better solution on not writing the IP address of the camera much better. This code shows the IP of the camera and might causes some security risks.

7. Appendix

7.1. Conventions in This Document

A convention has been used throughout this document to help the reader better understand the content of this document.

`Constant width text` – is used for showing code snippets.

```
This field shows the codes of some functions used in achieving the solution
```

7.2. Document Paths

The next table summarizes the paths used in the document:

Document Path	Absolute Path
`path1`	S:\Windows\AdminTools\Microsoft\dot NET Framework http://msdn.microsoft.com/en-us/netframework/aa569263
`path2`	S:\Windows\Engineering\Microchip\MPLAB Lite http://www.openpicus.com/site/downloads/downloads
`path3`	S:\Windows\AdminTools\WampServer http://www.wampserver.com/en/
`path4`	http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17718
`path5`	D://svn/CSM-0E2/trunk/0/SW/Manuals/Resources

Table 2: Document paths

7.3. Acronyms

[yyyy-mm-dd]: year-month-day, 4
WWW: World Wide Web, 7

Contact Information	
Email	info@cisium.com sales@cisium.com
Phones	+63 32 416 2927(Philippines) +44 20 7617 7603(UK) +34 93 310 3230(Spain)
Web	http://www.cisium.com