



Flyport IDE 2.2 – User guide

revision 1.0

[this page has intentionally been left blank]

[this page has intentionally been left blank]

Contents

| | |
|--|----|
| IDE 2.2 overview..... | 5 |
| Introduction..... | 5 |
| The working area..... | 6 |
| The download tool..... | 7 |
| Using the IDE..... | 8 |
| Project structure..... | 8 |
| Creating a new project..... | 8 |
| Opening an existing project – Importing from v2.0..... | 9 |
| Working with templates..... | 10 |
| Introducing the templates..... | 10 |
| Saving templates..... | 10 |
| Importing webpages..... | 11 |
| Importing external libraries..... | 12 |
| The TCP/IP wizard..... | 13 |
| Introduction..... | 13 |
| First page – services selection..... | 13 |
| Second page – Network configuration..... | 15 |
| Third page – Wireless configuration..... | 16 |
| Fourth page – Wireless security..... | 17 |
| Fifth page – Socket configuration..... | 18 |
| Sixth page – UDP Socket configuration..... | 19 |
| Seventh page – Hardware configuration..... | 20 |
| Eighth page – Finished..... | 21 |

IDE2.2 overview

Introduction

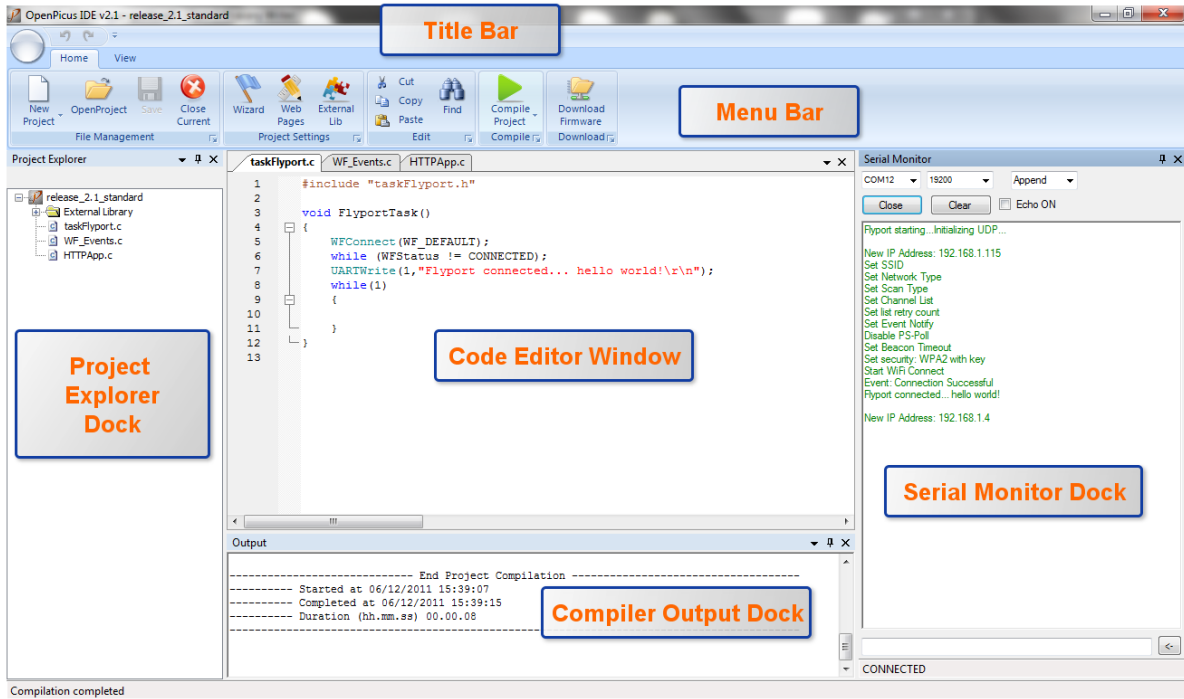
OpenPicus IDE 2.2 is a Windows application that allows the user to program and manage all the functionalities of the OpenPicus Flyport. It extends to the functions of the IDE 2.1 program, integrating some powerful tool to ease the developing of an entire Flyport project, and adds support for the new Flyport Ethernet Module. Here the features:

- **project explorer** to manage and modify all the project files (application files and external libraries);
- **powerful editor**, with syntax highlight, autocompleate function and tooltip helpers;
- **wizard tool**, to properly configure the device, with dynamic library compilation;
- compiling through direct control of **C30 Microchip compiler** (*user selectable versions from v3.24*) , to catch all compiling messages;
- **web pages importing**, to include html pages, javascript, images and all media of the webservice pages inside the project;
- **external libraries importing**, to add new functions to the program, for example to interface with external devices;
- **serial firmware download tool**, to deploy the firmware directly on the Flyport;
- **serial monitor**, to debug application and communicate with the device.
- **HTML project informations**, to embed documentation inside the project folder
- support for both **Flyport Ethernet and WiFi** modules

In the next paragraphs we'll see in detail any section of the IDE, and we'll provide some informations about its usage and how to create a complete project from the beginning to the download and testing.

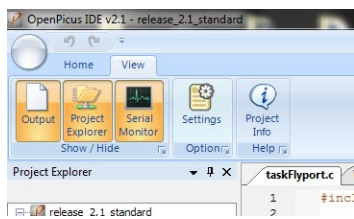
The working area

The IDE working area contains all the elements to create and manage the Flyport projects.



Let's take a look at each element of the working area:

- **Title bar:** displays the name and the version of the application, and the project name.
- **Menu bar:** is composed by the **Home tab** and the **View tab**. The *home tab* contains the icons to perform any action inside the IDE, like open/save/close a project, compile and import elements, and modify the text. In the *view tab* it's possible to switch on/off the output window, the project explorer and the serial monitor window, open the C30 compiler settings and project info documentation window



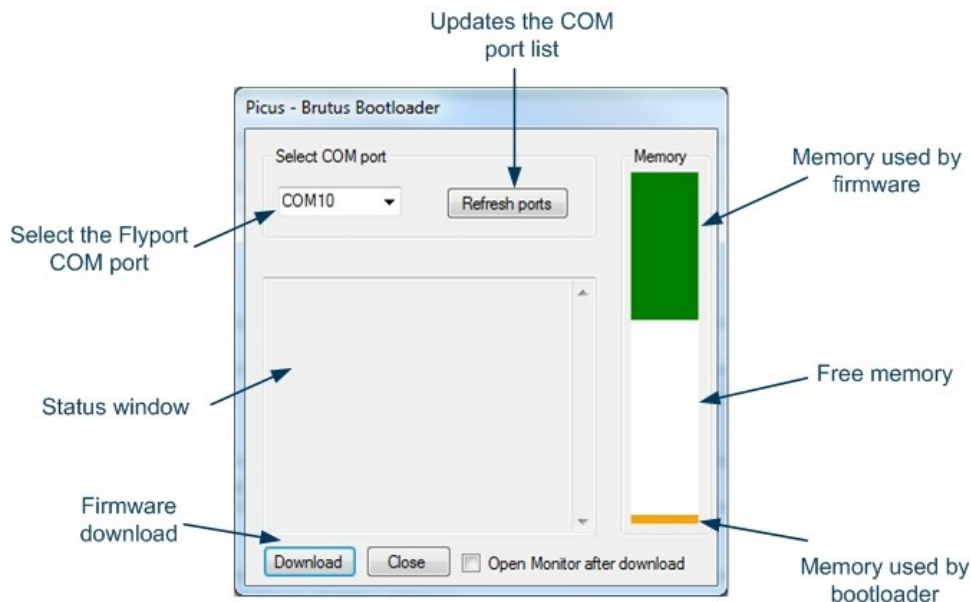
- **Project explorer:** here is where are shown all the project files that can be modified by the user. The application files are:
 - **taskFlyport.c:** contains the user's custom firmware.
 - **WF_events.c:** to manage all the wifi events, like connection established or lost (present only in flyport wi-fi projects).
 - **HTTPApp.c:** to manage the interaction with the web pages of the webserver (present only if webserver feature is activated inside the wizard).
 - The other files that can be modified are the external libraries files. The user can add libraries to communicate with displays, sensors or other devices. All the TCP stack files, operating system files

and Flyport Framework libraries are modified “behind the scene” by the system, according to the settings of the TCP wizard.

- **Code editor window:** is where can be edited the code for any project file. It supports command highlight and autocompletion.
- **Output window:** here is reported any message of the Compiler, and any other report about IDE operations.
- **Serial monitor:** an embedded serial monitor to communicate directly with the Flyport UART. Just set baud rate and COM port to receive debug messages and send strings to the device.

The download tool

The Flyport embeds a **serial bootloader** (a customized version of the ds30 loader) that enables the user to download the firmware simply using the serial port, instead a programmer. This makes everything easier and cheaper. The download tool integrated in the IDE communicates with the device and downloads the firmware directly on the Flyport, simply selecting the serial COM port and clicking “Download”. On the right is indicated the firmware size (green bar), the free memory (white bar) and the bootloader size (orange bar, at the end of the program memory).



Using the IDE

Project structure

When a new project is set up, a new folder is created. The folder contains the following subfolders and files.

Project subfolders:

- **Libs** – contains the Flyport Framework libraries, the FreeRTOS files and the (optional) external libraries.
- **Obj** – contains the compiled files.
- **Web pages** – contains the webpages of the webserver (optional). The webpages and all the related files can be placed anywhere on the PC, but placing all the files inside this folder keep the complete project easier to port, store and update.
- **Doc** – contains the HTML documentation of the project. The first file that IDE will open should always be named "index.html".

Project files (in blue the files that can be edited by the user in the IDE):

- **taskFlyport.c**: the custom application source code (with related taskFlyport.h).
- **HTTPApp.c**: the code to manage the dynamic components of the webpages for the webserver.
- **WF_events.c**: the file that manages all the WiFi events (and related WF_events.h). Those files are present only in Flyport Wi-Fi projects.
- **Project name.conf**: it's the file used by the IDE to load the project and retrieve its informations.
- **HTTPPrint.h, HTTPPrint.idx, MPFSImg2.s (pic memory usage) or MPFSImg2.bin (external flash usage)**: these files contain the webpages, the images (and all media) scripts, and all the information about dynamic variables of the webserver. They are automatically generated during webpages import, user **MUST NOT** modify them manually.
- **TCPIPConfig.h** and **WF_Config.h**: the configuration files. These files are automatically modified by TCP wizard. User should not modify them manually unless he doesn't know what he's doing. However, after manual changes (made outside of the IDE), to take effect, a **Recompile All** command must be issued.

Creating a new project

To create a new project, click on the button **New project**, you'll get a drop down menu that will allow you to choose the right *template* for your purpose. Template management will be focused later, for now let's see the starting templates:

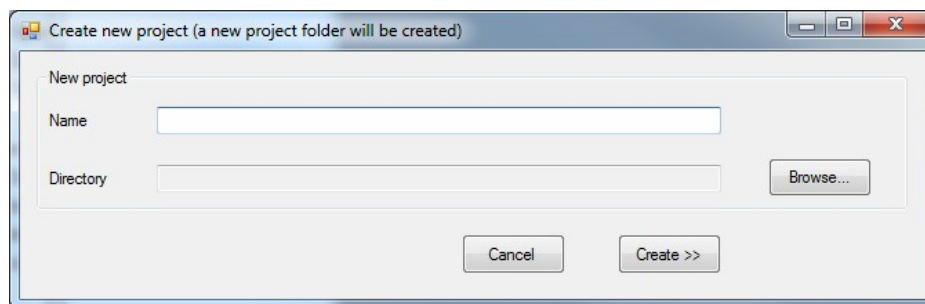
- **Wifi 2.2 – basic no webserver**: an empty project, without the webserver, so the HTTPApp.c file is not present inside the project.
- **Wifi 2.2 – blank webserver**: an empty project, including a blank test webpage for the webserver, with only a welcome message, but no dynamic components. So the HTTPApp.c file just contains empty definitions for all the functions to manage the webserver.
- **Wifi 2.2 – webserver example**: this template includes the standard Flyport webpage, with IO monitor and control and two analog inputs monitor. The HTTPApp.c files contains the needed code to

manage the dynamic components of the webpage.

- **Ethernet 2.2 – no webserver:** an empty project, without the webserver, so the HTTPApp.c file is not present inside the project.
- **Ethernet 2.2 – webpages in PIC memory:** this template includes the standard Flyport webpage, with IO monitor and control and two analog inputs monitor. This template is for use of Flyport Ethernet module, and the file WFEvents.c is not present. In this configuration, the MPFSIm2.s file is stored inside the microcontroller flash memory.
- **Ethernet 2.2 – webpages in Flash memory:** this template includes the standard Flyport webpage, with IO monitor and control and two analog inputs monitor. This template is for use of Flyport Ethernet module, and the file WFEvents.c is not present. In this configuration, the MPFSIm2.bin file must be uploaded inside the external flash memory, using the mpfsupload page of the webserver.
- **Ethernet 2.2 – webpages in Flash memory with protection select:** this template includes the same code of the above template, with the add-on of some UART commands to enable or disable the protection of the mpfsupload page. Use this code if the mpfsupload page should be reachable using *user* and *password* combination

All of the Wi-Fi templates (by default values) automatically creates an adhoc network (named FlyportNet) with no encryption and checks for the connection. When the network is created, the led on output 5 is turned on (the related code is inside the *WF_events.c* file).

After choosing the template, you have to select the name for the project and related directory in which the project folder will be created.

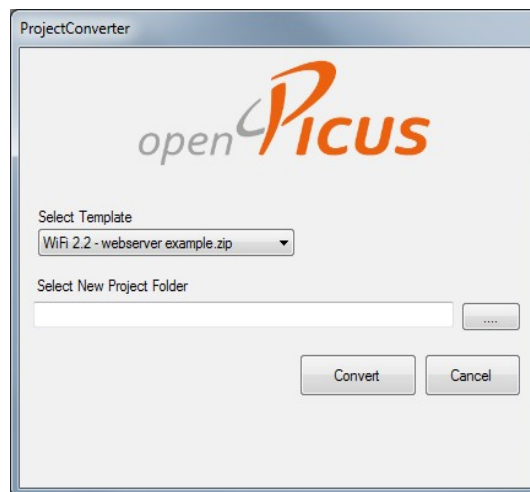


When the **Create** button is clicked, the project is created with his configuration parameters.

DIFFERENCE WITH IDE 1.0 VERSION: in the IDE v1.0, after creating a project, a TCP wizard was always raised. In the new version it's not needed, unless you don't want to change some settings. **From IDE v2.0 the project created is ready to be compiled.**

Opening an existing project – Importing from v2.0 or v2.1

To open an existing Flyport Wi-Fi project, click on Open Project button, and choose the proper .conf file. The IDE will automatically recognise an old 2.0 or 2.1 project and will ask you if you want to start a conversion process to import the old project.



You have to choose the new folder for the project and the template to apply to it. From the old project version will be kept the following files:

- All the application files (**taskFlyport.c**, **HTTPApp.c**, **WF_events.c**);
- the compiled webpages and dynamic data (**HTTPPrint.h**, **HTTPPrint.idx**, **MPFSImg2.s**);
- the configuration files (**TCIPConfig.h** and **WF_Config.h**);
- the external libraries files.

The Flyport Framework Libraries, and the FreeRTOS files will be kept from the template. In the case of the blank template and the standard template, there are not differences.

Note: the old projects can be imported as wifi project, but not as ethernet project due to differences in libraries configuration. The IDE will automatically show a message if user tries to convert a wi-fi project using a ethernet template.

Working with templates

Introducing the templates

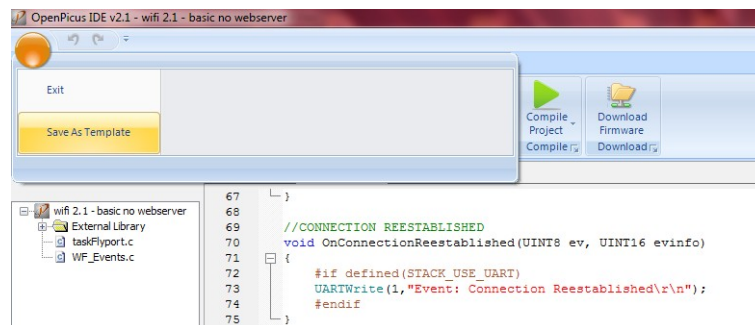
The templates are models that can be used as starting point for the new projects. A template contains the application files, the configuration files and all the related compiled files for the project, so it's ready to be compiled and downloaded on the Flyport, without needing any further operation.

Starting from one of the provided templates, you can change all the network settings, add your own code and modify/add/remove webpages. Then you can choose to save your own template to share it, or to reuse in future.

For example you could set Wizard parameters, then make the task Flyport do some standard operations you always want to include inside your firmware (for example a network scan, or a connection) and save them in a new template "**my standard app**", so, every new project will have the correct network and device settings and the right initialization code.

Saving templates

To save your own template, click the main “orb” and select “**Save as template**”. You can choose any folder



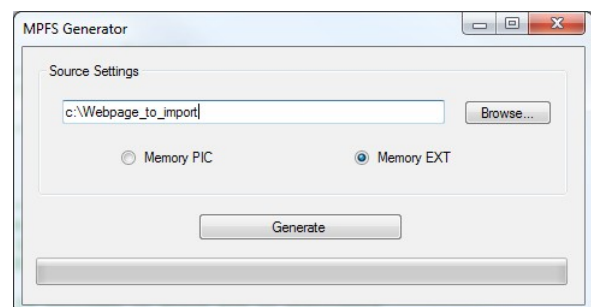
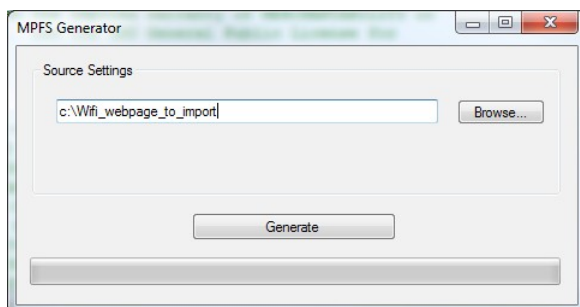
you want, but if you want to use the new template in the IDE, you have to put it inside the subfolder “..\Template\...”. At the startup the IDE will look in that folder for any .zip template file and will make it available for a new project.

WARNING: To properly save templates in IDE folder, it is needed to run the software with administrator privileges, or it could be impossible to write inside the “...\OpenPicus FlyPort IDE\Template” folder

Importing webpages

The IDE 2.2 includes a tool to automatically import new webpages inside the project. Importing webpages doesn't mean only importing the HTML code, but also images, sounds and dynamic code (like javascript). In this new release of the IDE, with a Flyport Ethernet project, there is the possibility to choose between internal flash memory, or external one.

To start the process, just click on the button **Web Pages**, and the import window will appear.



Webpage importing tools for Wi-Fi and Ethernet Projects

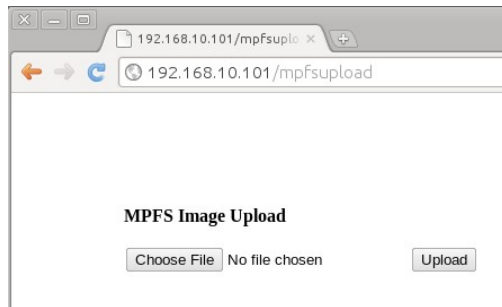
In the new form, you must select the folder in which are located all the webpages files (HTML, images, media, javascript...). Inside the folder you can order the data as you like, also with other subfolders. **All the files located in the selected folder and subfolders will be included inside the project.**

IMPORTANT:

the webpages of the webserver not only contain static data, but also dynamic contents that interacts with the firmware. The integration with the firmware is managed by the file HTTPApp.c. So when the webpages are changed, also the code in HTTPApp.c must be changed (if there are differences with the previous dynamic contents), otherwise, compiling errors can arise.

Ethernet only:

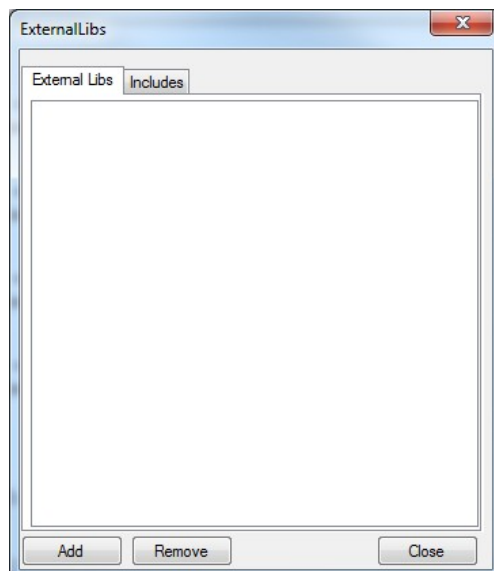
the compiled webpages can be located inside the PIC Flash Memory (embedded in firmware) like in Flyport Wi-Fi, or inside the expansion Flash memory mounted onboard of the Flyport Ethernet modules. Using the “Memory EXT” option, the IDE will generate a **MPFSImg2.bin** file inside the project folder. This file can be loaded with the using of a browser, with the *mpsfupload* webpage embedded inside the Flyport Ethernet.



This page permits the upload of the MPFSImg2.bin file directly into the External Flash memory. In this way it is not needed to reboot the flyport to have a upgraded webserver.

Importing external libraries

External libraries can be useful to extend Flyport functionalities (for example to connect to an external device, sensor, or display...), contains a code that can be easily reused and shared with others. To include the files, click on the **External lib** button, will appear a window to select the files to import in the project (.c, .s or .h), the source files and the header files will be automatically copied in the project subfolders `..\Libs\ExternalLib\` and `..\Libs\ExternalLib\Include\`.



The libraries must then be included in the task Flyport, in the WiFi events or in the webserver code using the directive `#include "header_name.h"`. The library notes will specify the header file to include. The files of the libraries can be added/removed using the same window.

The TCP/IP wizard

Introduction

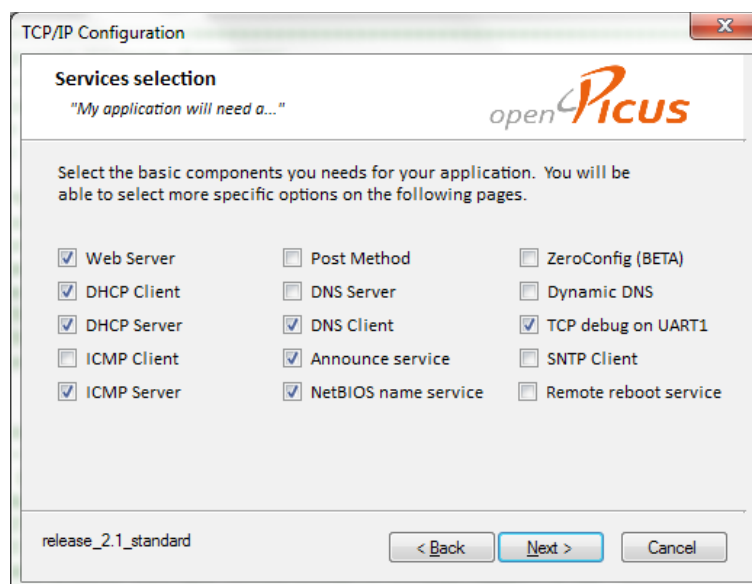
The TCP/IP wizard is the tool to properly configure the Flyport, the service available on the device, and the network configuration (like IP address, SSID, security...). The network configuration is needed to make the Flyport run in the right way, and connect to the desired network, the selection of the services is needed to minimize the resources used by the TCP/IP stack.

The Flyport is an embedded device, running a bootloader, a real time operating system, the TCP/IP stack and the user firmware. A lot of thing for an embedded microcontroller with a limited amount of resources, especially program and data memory. So, optimizing these resources it's a primary issue. For example, if your project doesn't use the webserver, maybe it's better if you not include it, so you'll have more memory available for your application, and the task managing the TCP stack will be lighter and faster.

In the next paragraphs we'll see in detail any property can be configured in the wizard.

First page – services selection

In the first page of the wizard is possible to select the service to include inside the Flyport. In this way it is possible to reduce at the minimum the size of the code for the TCP stack according to your needing.



Let's see all the services that can be enabled/disabled:

- **Webserver:** enabling this service, the Flyport can be accessed through a classic HTML browser, and will display dynamic HTML pages.
 - **POST method:** enabling/disabling the support for the POST method in the Flyport webserver.
- **DHCP client:** if the Flyport is connected to an infrastructure network where is present a DHCP server, this service will allow the Flyport to acquire a dynamic IP address assigned by the server. **NOTE:** in the user application is possible to enable/disable at runtime the DHCP client, but if it has not been selected in the wizard, the DHCP client will not be present in the TCP stack, so the instruction will be useless. If you think you may need the client in your app, select it, you'll always be able to turn off it at runtime.
- **DHCP server:** useful in Wi-Fi ad hoc networks, or for a point-to-point ethernet link. It's a DHCP server capable of assign up to one IP address.
- **ICMP client:** client to send ping requests to a remote server (demo).
- **ICMP server:** a server to respond to remote ping requests. So if you want to ping the Flyport from your pc, you have to enable this service.
- **DNS server:** simulates a DNS server, answering with a dummy reply (with Flyport address) to DNS requests. Can be useful to simulate the presence of a DNS inside a network.
- **DNS client:** allows the Flyport to access to a remote DNS server to resolve host names (for example www.openpicus.com). It will be possible to make requests using host names instead of IP address.
- **Announce service:** sends an UDP announcement packet at boot and upon DHCP events. The Microchip Ethernet Discoverer can be used to reveal this announcement.
- **NetBIOS name service:** the Flyport will be able to answer to NetBIOS name queries.
- **ZeroConfig (BETA):** this service allows flyport to use the "Zeroconfig" service, but it is a very beta version, to use only for development.
- **Dynamic DNS:** a client for dynamic DNS, like dyndns.org.
- **TCP debug on UART1:** all the TCP events will be communicated on UART1 (19200, 8, N, 1). Disable it if you need to communicate with another device on UART1.
- **SNTP client:** obtain the current time from a pool of SNTP servers. Time is returned like number of seconds from 1st jan 1970, 00:00 (beta).
- **Remote reboot service:** allows the remote reboot of the Flyport, if receive special message on defined port (by default, port 69, char 'R', only allowed device inside the Flyport subnet).

Second page – Network configuration

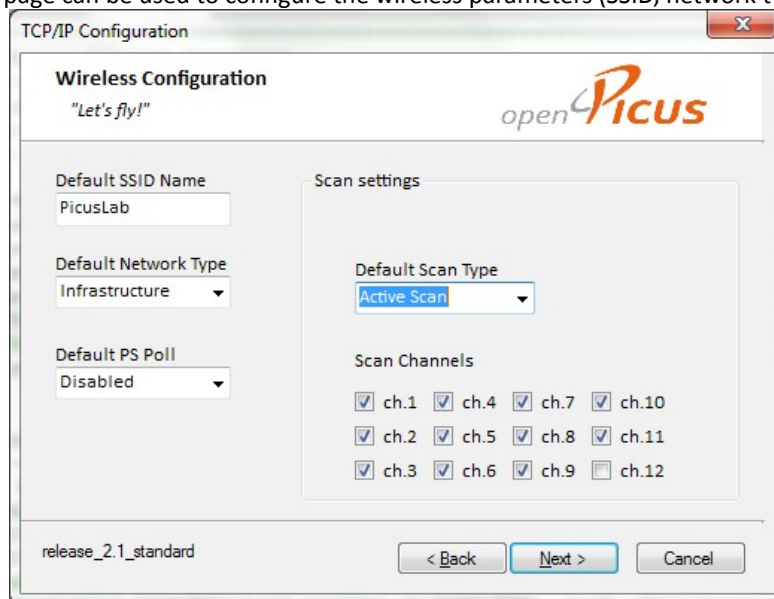
The second page of the wizard contains the network parameters to select IP address of the device, subnet mask, etc...

- **Host name:** the host name the Flyport will respond to (if NetBIOS enabled).
- **MAC Address:** any network device has it's own MAC address (you can find the Flyport Wi-Fi one printed on the wireless module, it should start with 00:1E:C0...). If you need i can experiment changing that MAC address with your own selecting Custom and writing the desired MAC Address.
NOTE: inserting the MAC address 00:04:A3:00:00:00 (the default mode) the Flyport will use it's default MAC address (printed on the wireless module). This is just a convention, because the IDE doesn't know the Flyport MAC, so that address is used like "standard" for all the devices. So if you let that address, even if you have selected "Custom" you'll se the default Flyport address.
- **IP Address:** the starting IP address of the Flyport (will be overwritten if using DHCP client, in a network with a DHCP server different from Flyport itself).
- **Subnet mask:** the subnet mask of the network (overwritten by DHCP).
- **Gateway:** default gateway (overwritten by DHCP).
- **Primary DNS server:** default primary DNS server (overwritten by DHCP).
- **Secondary DNS server:** default secondary DNS server (overwritten by DHCP).

Third page – Wireless configuration

This page appears only using Flyport Wi-Fi projects

The third page can be used to configure the wireless parameters (SSID, network type...).



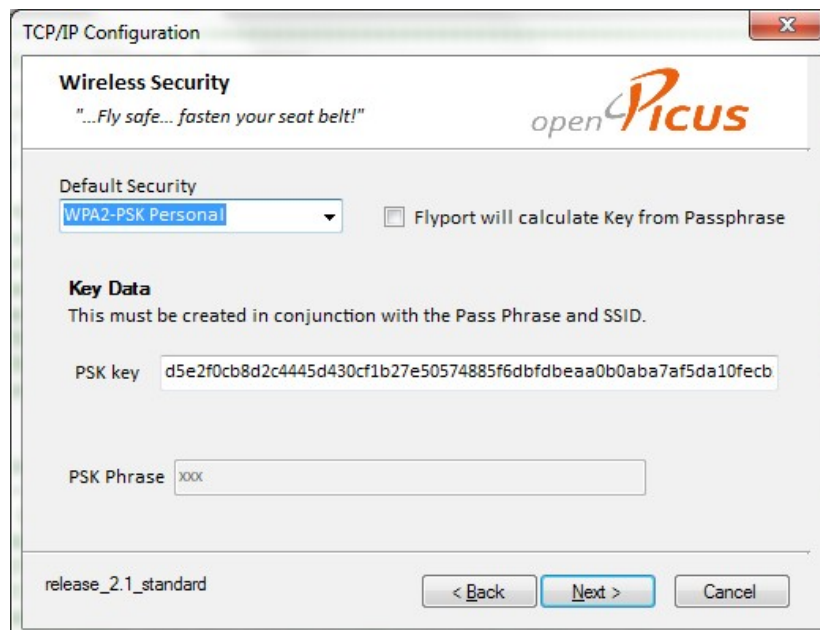
- **SSID:** the name of the network to create or to join to.
- **Default network type:** defines if Flyport must create an adhoc network or must join an existent infrastructure one.
- **Default PS poll:** enables or disables power save mode.
- **Default scan type:** active scan (with probe request sending) or passive scan (just listening for beacons).
- **Scan channels:** channels to scan when searching for networks.

Fourth page – Wireless security

This page appears only using Flyport Wi-Fi projects

In this page is possible to choose between the following security mode:

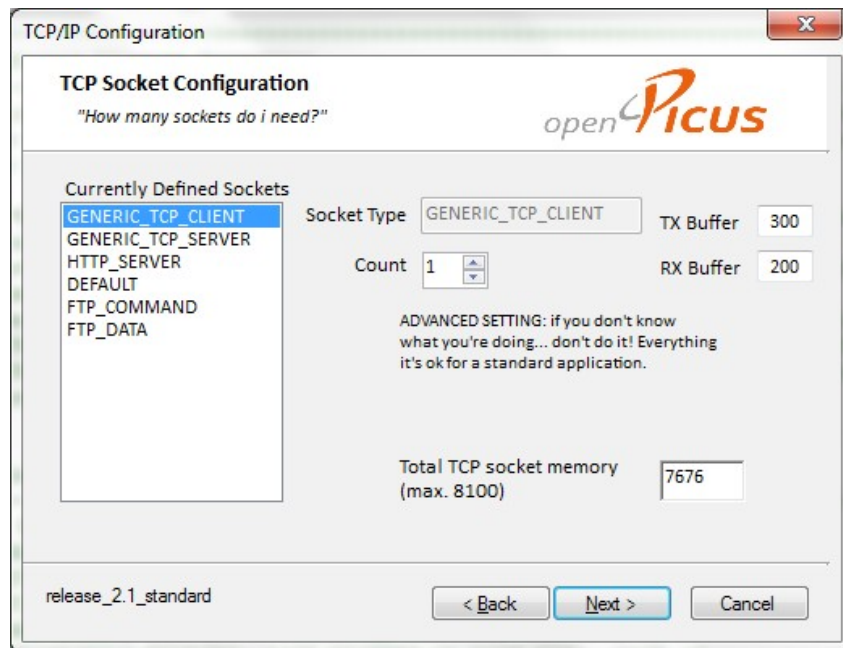
- **Open:** no security.
- **WEP:** WEP security, at 40 or 104 bits.
- **WPA or WPA2:** WPA, WPA2, or automatic detection. **NOT AVAILABLE FOR AD HOC NETWORKS.** When using WPA or WPA2 you can choose to insert the PSK key or the PSK passphrase. Inserting the passphrase, the Flyport will calculate the key. This is a LONG process and needs many operations, so **it will took about 20-30 seconds to complete.** The connection using the PSK key is much faster (few seconds). You can calculate your key from the passphrase and the SSID using one of the many online tools, for example: <http://www.wireshark.org/tools/wpa-psk.html> .



The screenshot shows a window titled "TCP/IP Configuration" with a close button in the top right corner. The window has a header section with the text "Wireless Security" and the slogan "...Fly safe... fasten your seat belt!" next to the "openPicus" logo. Below the header, there is a "Default Security" section with a dropdown menu set to "WPA2-PSK Personal" and a checkbox labeled "Flyport will calculate Key from Passphrase" which is currently unchecked. The "Key Data" section contains the instruction "This must be created in conjunction with the Pass Phrase and SSID." and two input fields: "PSK key" containing the hexadecimal string "d5e2f0cb8d2c4445d430cf1b27e50574885f6dbfdbbeaa0b0aba7af5da10fecb" and "PSK Phrase" containing "xxx". At the bottom left, the text "release_2.1_standard" is visible. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

Fifth page – Socket configuration

The fifth page contains advanced socket configuration to select the number of available socket for each service and the memory allocated (TX and RX) for each socket.



NOTE: the memory indicated is allocated in the wireless module, not inside the PIC. The maximum quantity available is about 8100, so the wizard will block you from allocating more memory.

The memory for each socket refers to the receive and transmit buffer of each socket, the number (count) refers to the number of socket for each service. So, if you want to create two servers on the Flyport, listening on two different ports, you'll have to select count=2 for **GENERIC_TCP_SERVER**. This will allow you to create two different listening sockets.

The default amount of memory has been decided using standard application, but obviously you can change it, for example if you have to send very long strings over TCP, increase the TX buffer, or you can delete FTP sockets if you don't need them, or TCP client if you're just using the server, and so on... You'd better not to delete the "DEFAULT" socket, since it's used by the TCP stack for standard operations.

Sixth page – UDP Socket configuration

In this page is possible to configure the UDP sockets needed by user application and their related buffers.

| Sockets | Buffer |
|--|--------|
| <input checked="" type="checkbox"/> Socket n.1 | 150 |
| <input checked="" type="checkbox"/> Socket n.2 | 256 |
| <input type="checkbox"/> Socket n.3 | |
| <input type="checkbox"/> Socket n.4 | |

Memory usage

Memory used (by others)
256

Total UDP socket memory
406

Total memory (Max. 6144)
662

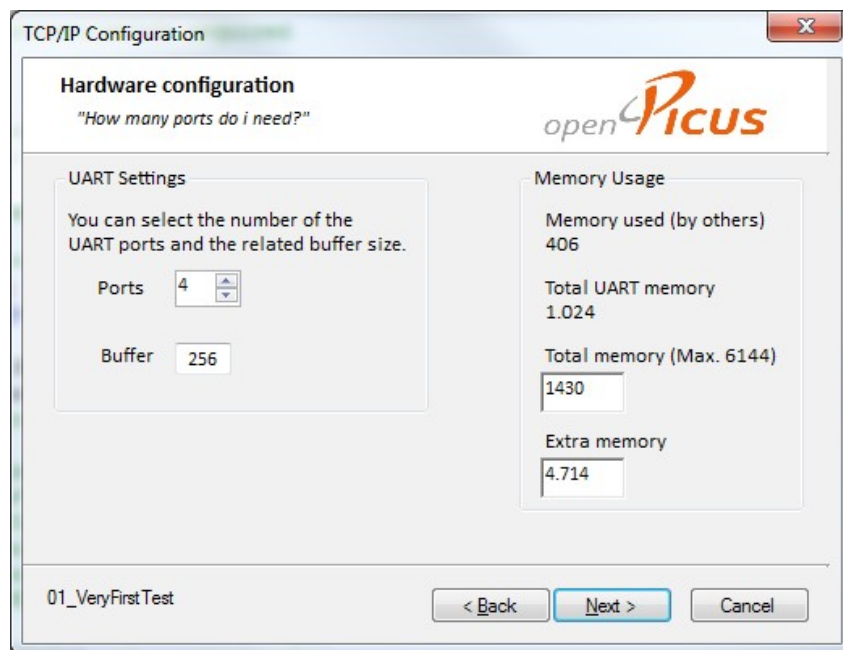
01_VeryFirst Test

< Back Next > Cancel

On the left of the page it is possible to enable up to 4 different sockets configuring also their buffer size. To properly check the memory occupation of UDP sockets there is the sum of all the buffers occupation on the right side of the page. If the user tries to configure buffers size bigger than 6144 Bytes, the wizard will automatically block the process.

Seventh page – Hardware configuration

In this page is possible to configure the UART configuration of Flyport's PIC microcontroller. In difference with UDP Sockets, UART buffers size is equal for every UART Module. Those buffers are used by Receive Interrupt Handler of every UART module enabled to easily access to received datas, so them should be big enough to handle the strings received by other devices connected to Flyport's UART ports, but others buffers can be created in user specific application.



On the top of the page it is possible to enable up to 4 UART modules Buffers, configuring also their size. To properly check the memory occupation of UART Receive Buffers there is the sum of all the buffers occupation on the bottom side of the page, and the extra memory free space. If the user tries to configure buffers size bigger than 6144 Bytes, the wizard will automatically block the process.

Eighth page – Finished

The whole configuration is ready to be compiled on Flyport's project. Click "Finish" and Remember to compile again the application before to download the firmware.

