

**2019 MASTERs Conference**

**23075 IoT6**

**Simplifying TCP/IP Applications with MPLAB® Harmony**

**Hands-On**

**Lab Manual**

***Instructors:***

***Martin Ruppert***

***Raji Shanmugasundaram***

***Niklas Larsson***

*Microchip Technology Inc.*

**MPLAB® Harmony TCP/IP Stack**

# UDP Module API Function List

## Socket Management Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_ServerOpen** | Opens a UDP socket as a server. |
| **TCPIP\_UDP\_ClientOpen** | Opens a UDP socket as a client. |
| **TCPIP\_UDP\_IsOpened** | Determines if a socket was opened. |
| **TCPIP\_UDP\_IsConnected** | Determines if a socket has an established connection. |
| **TCPIP\_UDP\_Bind** | Bind a socket to a local address and port. This function is meant for client sockets. It assigns a specific source address and port for a socket. |
| **TCPIP\_UDP\_RemoteBind** | Bind a socket to a remote address This function is meant for server sockets. |
| **TCPIP\_UDP\_Close** | Closes a UDP socket and frees the handle. |
| **TCPIP\_UDP\_OptionsGet** | Allows getting the options for a socket such as current RX/TX buffer size, etc. |
| **TCPIP\_UDP\_OptionsSet** | Allows setting options to a socket like adjust RX/TX buffer size, etc |
| **TCPIP\_UDP\_SocketInfoGet** | Returns information about a selected UDP socket. |
| **TCPIP\_UDP\_SocketNetGet** | Gets the network interface of an UDP socket |
| **TCPIP\_UDP\_SocketNetSet** | Sets the network interface for an UDP socket |
| **TCPIP\_UDP\_TxOffsetSet** | Moves the pointer within the TX buffer. |
| **TCPIP\_UDP\_SourceIPAddressSet** | Sets the source IP address of a socket |
| **TCPIP\_UDP\_BcastIPV4AddressSet** | Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts. |
| **TCPIP\_UDP\_DestinationIPAddressSet** | Sets the destination IP address of a socket |
| **TCPIP\_UDP\_DestinationPortSet** | Sets the destination port of a socket |
| **TCPIP\_UDP\_Disconnect** | Disconnects a UDP socket and re-initializes it. |
| **TCPIP\_UDP\_SignalHandlerDeregister** | Deregisters a previously registered UDP socket signal handler. |
| **TCPIP\_UDP\_SignalHandlerRegister** | Registers a UDP socket signal handler. |
| **TCPIP\_UDP\_Task Standard** | TCP/IP stack module task function. |

## Transmit Data Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_PutIsReady** | Determines how many bytes can be written to the UDP socket. |
| **TCPIP\_UDP\_TxPutIsReady** | Determines how many bytes can be written to the UDP socket. |
| **TCPIP\_UDP\_ArrayPut** | Writes an array of bytes to the UDP socket. |
| **TCPIP\_UDP\_StringPut** | Writes a null-terminated string to the UDP socket. |
| **TCPIP\_UDP\_Put** | Writes a byte to the UDP socket. |
| **TCPIP\_UDP\_TxCountGet** | Returns the amount of bytes written into the UDP socket. |
| **TCPIP\_UDP\_Flush** | Transmits all pending data in a UDP socket. |

## Receive Data Transfer Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_GetIsReady** | Determines how many bytes can be read from the UDP socket. |
| **TCPIP\_UDP\_ArrayGet** | Reads an array of bytes from the UDP socket. |
| **TCPIP\_UDP\_Get** | Reads a byte from the UDP socket. |
| **TCPIP\_UDP\_RxOffsetSet** | Moves the read pointer within the socket RX buffer. |
| **TCPIP\_UDP\_Discard** | Discards any remaining RX data from a UDP socket. |

Table of Contents

UDP Module API Function List 2

Socket Management Functions 2

Transmit Data Functions 2

Receive Data Transfer Functions 2

Introduction 5

Hardware Requirements 5

Software Requirements 5

Lab 1 6

Introduction 6

Lab Procedure 7

Starting MPLAB X IDE 7

Project Setup 8

MHC: BSP Selection 9

MHC: Ethernet I/O Pin Configuration 11

MHC: TCP/IP Stack Configuration 13

MHC: Network Interface Configuration 14

MHC: ICMP Configuration 15

MHC: Network Interface Driver Selection 15

MHC: Console Configuration 17

MHC: Application Configuration 19

MHC: Project Generation 19

LED Flasher Implementation 21

LED Flasher Code 22

Header File 22

Source File Setup 23

Project Build 26

Programming 26

Application Validation 29

Network Interfacing 29

Cable Connections 30

Network Connectivity with TCP/IP Discovery Tool 31

Checking Network Connectivity with Windows Ping Client 32

Harmony TCPIP Command Console 33

Lab 2 36

Introduction 36

Data Protocol 37

Application Implementation 38

Objectives 39

Lab Procedure 39

Project Setup 39

MHC Application Configuration 41

MHC: DHCP Server Configuration 43

MHC: Setting the Host Name 44

MHC: Project Generation 45

Application Source and Header File Setup 47

Network Communications Controller Modification 50

Project Build 54

Programming 54

Application Testing 55

Cable Connections 56

UDP Server Testing 59

TCP Client Testing 63

ECS Testing 73

Harmony TCP/IP API Subset For Lab 2 78

TCP Socket Management Functions 78

TCPIP\_TCP\_ArrayGet Function 78

TCPIP\_TCP\_ClientOpen Function 78

TCPIP\_TCP\_Close Function 79

TCPIP\_TCP\_GetIsReady Function 79

TCPIP\_TCP\_IsConnected Function 80

TCPIP\_TCP\_PutIsReady Function 80

TCPIP\_TCP\_StringPut Function 81

TCPIP\_TCP\_WasReset Function 81

UDP Socket Management Functions 82

TCPIP\_UDP\_ArrayGet Function 82

TCPIP\_UDP\_Close Function 83

TCPIP\_UDP\_GetIsReady Function 83

TCPIP\_UDP\_ServerOpen Function 84

TCPIP\_UDP\_SocketInfoGet Function 84

UDP\_SOCKET\_INFO Structure 85

Network Communications Controller Application Code Modification Solutions 86

TCP Module API Function List 87

Socket Management Functions 87

Transmit Data Functions 87

Receive Data Transfer Functions 87

# Introduction

This Lab Manual provides the step by step procedure to complete two labs in the MASTERs 21070 NET1 Class. In Lab 1 you will configure the MPLAB Harmony TCPIP/IP Stack and test basic network connectivity and in Lab 2 you will learn how to implement and test a network application which will utilise a TCP Client and UDP Server in a real world application. Both Labs have specific hardware and software requirements.

# Hardware Requirements

The following hardware is required:

* **SAM E70 Xpained Ultra** (Microchip Part Number: DM320113)
  + [http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320113](http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320007)
* OLED1 Xplained Pro extension kit (Microchip Part Number: ATOLED1-XPRO)
* **Cat 5 Ethernet Patch Cable**
* **USB Male A to USB Male B Micro Cable**

The following hardware is optional:

* **Multimedia Expansion Board II** (Microchip Part Number: DM320005-2)
  + <http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320005-2>
* **MPLAB ICD3 In-Circuit Debugger** (Microchip Part Number: DV164035)
  + <http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dv164035>
* **USB Male A to USB Male B Cable** (supplied with DV164035)
* **6 Core Modular Cable with RJ11 Connectors** (supplied with DV164035)
* **9V, 500mA Power Supply with 2.5mm Plug**

# Software Requirements

The following software is required:

* **Microchip MPLAB X IDE v5.20.04**
  + <http://www.microchip.com/mplab>
* **Microchip MPLAB XC32 Compiler v2.20**
  + <http://www.microchip.com/mplab/compilers>
* **Microchip MPLAB Harmony v2.03B**
  + <http://www.microchip.com/mplab/mplab-harmony>
* **Microchip MPLAB Harmony Configuration (MHC) Tool Plugin v2.0.3.5**
  + MPLAB X Plugin “*com-microchip-mplab-modules-mhc.nbm”* is bundled with MPLAB Harmony under the microchip\v2\_03b\utilities\mhc folder
* **Tera Term v4.95**
  + <https://ttssh2.osdn.jp/index.html.en>
* **Packet Sender v5.1 (lab 2 only)**
  + [https://packetsender.com](https://packetsender.com/)
* **JSMN JSON Parser (lab 2 only)**
  + <http://www.zserge.com/jsmn.html>

Lab 3

# Introduction

In many IoT applications, JSON is commonly used as a format in order to transport high-level data in an effective way. It is generally an alternative to XML. Consider the following example of describing a person named Raji-Niklas Ruppert in JSON-format:

{

“firstName” : “Raji-Niklas”,

“lastName” : “Ruppert”,

“age” : 30,

“address” : {

“streetAddress” : “2355 W Chandler Blvd”,

“city” : “Chandler”,

“state” : “AZ”,

“postalCode” : “85224”,

}

}

Using this format makes it very easy to communicate between applications requiring information about Raji-Niklas Ruppert.

The advantage of using JSON in embedded applications is that because it is easy to read for humans, it is simple to parse and make use of. Due to this, it is commonly used to transmit data between a server and a web application. In this lab we are going to implement an embedded application fetching weather data from a web server. When the application accesses a specific URL specifying a command with a geographic location, the web server will respond by sending the current weather in JSON-format to the web application. The application will be running on our SAME70-boards.

In this lab we will only do very simple parsing (which is one of the strengths using JSON), using standard string operations. There are however more sophisticated parsers which can be used for more robust and complex applications, while still only consuming a very limited footprint.

The weather service used in this lab is <https://openweathermap.org/>. With OpenWeatherMap, there are several services such as hourly forecast, UV Index, Air pollution and more, all outputting in JSON. With the free account there are limited option to only use the “Current Weather Data” service. With this service you can request the current weather from different geographic locations. Depending on by which method (City ID, ZIP Code, Coordinates etc.) the URL call will be slightly different. A full description of the API can be found here: <https://openweathermap.org/current>. For this lab we will fetch current weather by city. The following URL for this is:

http://api.openweathermap.org/data/2.5/weather?q=**{CITY}**&APPID=**{API Key}**

The API Key is unique to each user. This is also how OpenWeatherMap tracks how many requests you attempt. The API Key is a 15-byte long hexadecimal string. It can look like this:

ed3da58111974261002c2af4f8e8e81f

In most JSON API:s there is also a well defined format specified, which tells you where the different objects and strings are located in the JSON-message. From OpenWeatherMap:

{"coord":{"lon":-122.09,"lat":37.39},

"sys":{"type":3,"id":168940,"message":0.0297,"country":"US","sunrise":1427723751,"sunset":1427768967},

"weather":[{"id":800,"main":"Clear","description":"Sky is Clear","icon":"01n"}],

"base":"stations",

"main":{"temp":285.68,"humidity":74,"pressure":1016.8,"temp\_min":284.82,"temp\_max":286.48},

"wind":{"speed":0.96,"deg":285.001},

"clouds":{"all":0},

"dt":1427700245,

"id":0,

"name":"Mountain View",

"cod":200}

Application Flow

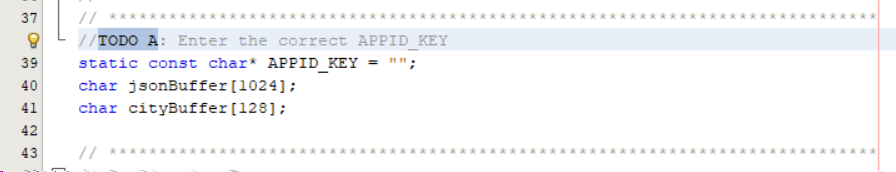
Machine generated alternative text:
Initiate 
Harmony, 
TCPIP and 
Application 
Print resulting 
JSON and 
Current 
Weather of 
chose city 
Choose City 
to retrieve 
data from 
Parse 
retrieved 
JSON data 
Choose 
which City 
to retrieve 
data from 

Lab Outline

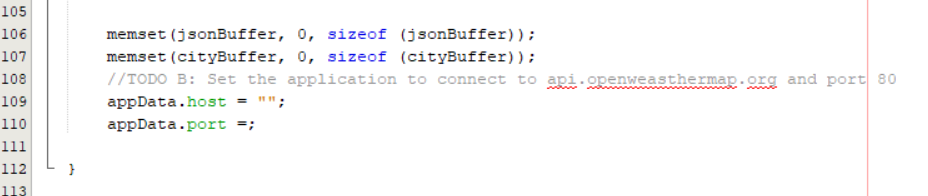
* The pre-made template is built from the Harmony example project, tcpip\_tcp\_client.
* First, we will need to declare the APPID\_KEY.
* We will then set the host & port of the remote connection static as we will only connect to OpenWeatherMap.
* After this, we will redirect the user input from the command console to a char\* buffer to be used in the application.
* Now we have all information required to build the URL from the introduction.
* When we have connected and requested the data, we need to parse the resulting JSON- string (the whole JSON containing the current weather will be in one string).
* Typically, good practice when you debug JSON-strings is to print the resulting string for you to view with your own eyes that it looks correct.
* At last, redirect the application to go back to accepting user input.

# Lab Procedure

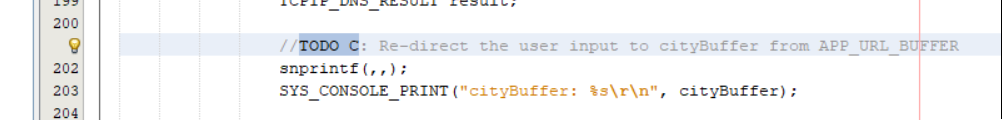
1. Start by closing any open projects in MPLAB X IDE.
2. Open a new project and choose lab3 -> Firmware -> sam\_e70\_xult\_freertos.X.
3. Open the file app.c located under source files.
4. Go to (CTRL+F) “TODO A”. Enter the correct APPID\_KEY. Either you create your own account on OpenWeatherMap or you take the one written I the class.



1. Now scroll down to “TODO B”, the function APP\_Initialize.
2. Set the application to connect to the host api.openweathermap.org and the port to 80. This is set to 80 because this call will be over HTTP.

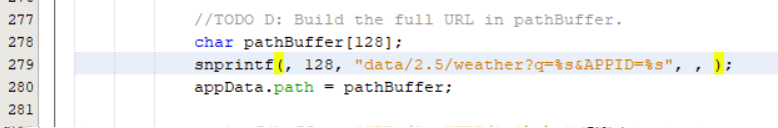


1. Re-direct the user input from APP\_URL\_BUFFER to the cityBuffer array. This can be done in several ways, but one is to use the built-in C function **snprintf**(char\* dest, size\_t size, const char \*format, …). The first argument is the destination buffer (cityBuffer), the second one is the max size to be copied (128, because that is specified in the declaration) and the formatted input in this scenario is APP\_URL\_BUFFER. This can be found in “TODO C”.

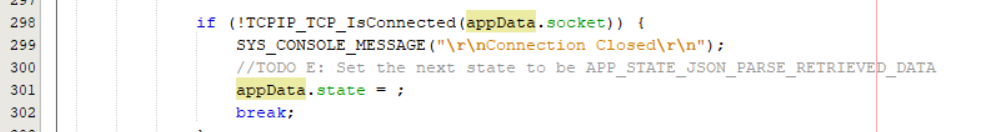


1. Scroll down to “TODO D”, the state APP\_TCPIP\_WAIT\_FOR\_CONNECTION. In this state we will wait for a connection to be established. Once established we will send a GET command with the full URL in the format specified in the introduction:

http://api.openweathermap.org/data/2.5/weather?q=**{CITY}**&APPID=**{API Key}**.



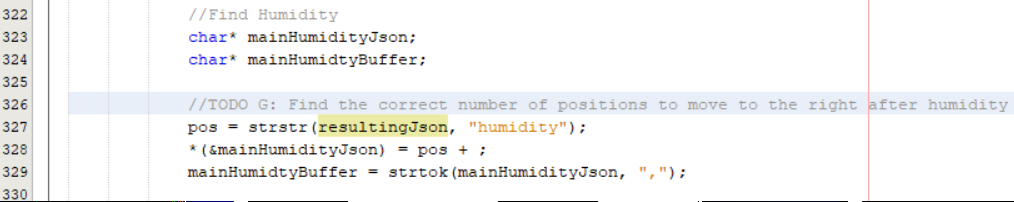
1. Once the request is sent to the server, the application will go into the APP\_TCIPIP\_WAIT\_FOR\_RESPONSE state. Once the connection is closed, set the next state to be APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA.



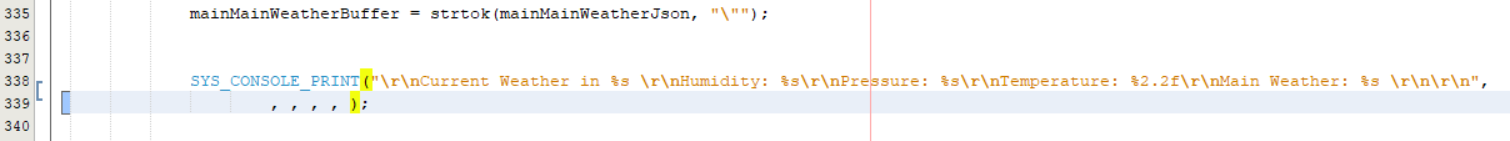
1. Now go down in the state APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA. One of the first things we want to do after we have sorted out the JSON-part of the retrieved data is to print the raw JSON-string. This helps us debug & analyse.



1. In a real application, we would need to first know the format of the JSON message in order to be able to parse it correctly. To make this lab more efficiently, we will do this backwards. If you look on this example piece of API response from OpenWeatherMap found in the introduction section to this lab. Looking at the format from the API, we need to calculate in what position the value of humidity start. The function strstr will cut the resultingJson string at the first occurrence of “humidity”. A hint is to look at the other blocks where you parse the temperature, pressure and main weather.



1. Once the parsing is done, we wish to print the values of the main weather, pressure, temperature and humidity.



1. Now to complete the loop, we want to go back to the APP\_TCPIP\_WAITING\_FOR\_COMMAND state once the JSON-parsing and printing is done.

