

**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.*

Table of Contents

[Introduction 4](#_Toc10545757)

[Hardware Requirements 4](#_Toc10545758)

[Lab 1 6](#_Toc10545759)

[Overview 6](#_Toc10545760)

[Lab Procedure 7](#_Toc10545761)

[Starting MPLAB X IDE 7](#_Toc10545762)

[Project Load, modify with MHC, Generate, build and run 8](#_Toc10545763)

[Lab 2 27](#_Toc10545764)

[Overview 27](#_Toc10545765)

[Lab Procedure 30](#_Toc10545766)

[MPLAB® Harmony TCP/IP Stack 32](#_Toc10545767)

[TCP Module API Function List 32](#_Toc10545768)

[Socket Management Functions 32](#_Toc10545769)

[Transmit Data Functions 32](#_Toc10545770)

[Receive Data Transfer Functions 32](#_Toc10545771)

[UDP Module API Function List 33](#_Toc10545772)

[Socket Management Functions 33](#_Toc10545773)

[Transmit Data Functions 33](#_Toc10545774)

[Receive Data Transfer Functions 33](#_Toc10545775)

[Harmony TCP/IP API Subset For all Lab’s 34](#_Toc10545776)

[TCP Socket Management Functions 34](#_Toc10545777)

[TCPIP\_TCP\_ArrayGet Function 34](#_Toc10545778)

[Function Prototype 34](#_Toc10545779)

[TCPIP\_TCP\_ClientOpen Function 34](#_Toc10545780)

[TCPIP\_TCP\_Close Function 35](#_Toc10545781)

[TCPIP\_TCP\_GetIsReady Function 35](#_Toc10545782)

[TCPIP\_TCP\_IsConnected Function 36](#_Toc10545783)

[TCPIP\_TCP\_PutIsReady Function 36](#_Toc10545784)

[TCPIP\_TCP\_StringPut Function 37](#_Toc10545785)

[TCPIP\_TCP\_WasReset Function 37](#_Toc10545786)

[UDP Socket Management Functions 38](#_Toc10545787)

[TCPIP\_UDP\_ArrayGet Function 38](#_Toc10545788)

[TCPIP\_UDP\_Close Function 39](#_Toc10545789)

[TCPIP\_UDP\_GetIsReady Function 39](#_Toc10545790)

[TCPIP\_UDP\_ServerOpen Function 40](#_Toc10545791)

[TCPIP\_UDP\_SocketInfoGet Function 40](#_Toc10545792)

[UDP\_SOCKET\_INFO Structure 41](#_Toc10545793)

[42](#_Toc10545794)











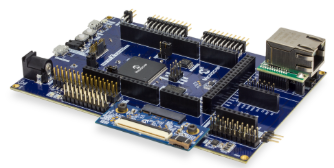
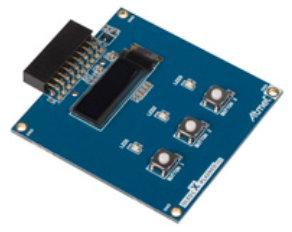
# Introduction

This Lab Manual provides the step by step procedure to complete two labs in the MASTERs 32075 IoT6 Class.

In Lab 1 we will open a TCP project, do some stack re-configuration and a connectivity check and in Lab 2 we will show an Application integration for local access, using the example of a Vending machine. Finally in Lab 3 we will make an Application integration for external access, using the example of a Weather Service.

# 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**

Software Requirements

The following software is required:

* **Microchip MPLAB X IDE v5.20**
  + <http://www.microchip.com/mplab>
* **Microchip MPLAB XC32 Compiler v2.15**
  + <http://www.microchip.com/mplab/compilers>
* **Microchip MPLAB Harmony 3**
  + <http://www.microchip.com/mplab/mplab-harmony>
* **Microchip MPLAB Harmony Configuration (MHC) Tool Plugin v3.3.0.1**
* **Tera Term v4.95**

# Lab 1

## Overview

Lab 1 will show you how to open an existing TCP/IP MPLAB Harmony 3 Project and using the MPLAB Harmony Configuration (MHC) Tool. The project will incorporate basic TCP/IP functionality to allow the SAM E70 Xplained Ultra to connect to an Ethernet Network, along with a simple application to flash a “Heartbeat” LED every 500ms. Once the project is generated and programmed onto the development kit, you will use a number of techniques to validate that the PIC is connected to a network and determine its IP Address. The concepts that will be covered in this lab include:

* Open a SAM E70 MPLAB X Project
* Configuring the MPLAB Harmony path
* Configuring the TCP/IP Stack options, including:
  + Network Configuration of the Host Name
  + TCP/IP Services including Dynamic Host Configuration Protocol Client, ICMPv4 Server (for Ping testing) & Announce Discovery Tool
  + Bandwidth testing with “iperf”
* Configuring the Harmony Console and Command Service for monitoring and control of the TCP/IP stack via a Terminal Client running on a USB CDC Interface (Emulated RS232 COM Port).

Toggling the IO Pin that drives USER\_LED0 on the SAM E70 Xplained UltraSAM E70 Xplained Ultra

* Using the Windows Command Line Ping Tool and the Microchip TCP/IP Discovery tools to test connectivity of your SAM E70 Xplained UltraSAM E70 Xplained Ultra on the network
* Use the Console and Command System to get help on available TCPIP Commands and execute a command to get information about the network configuration.

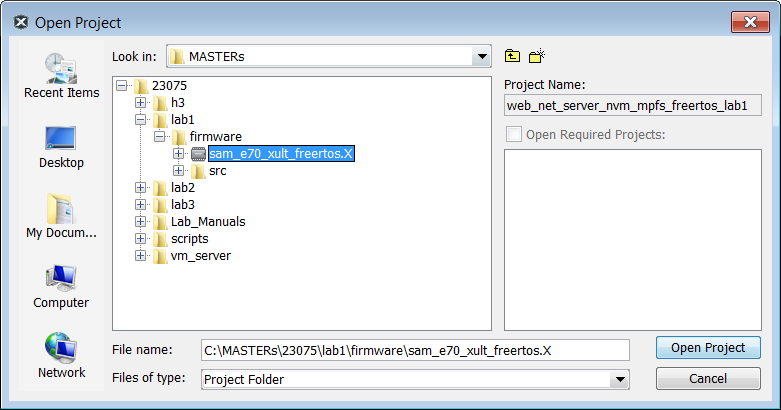
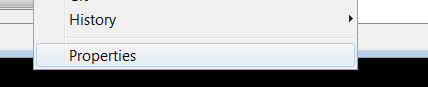
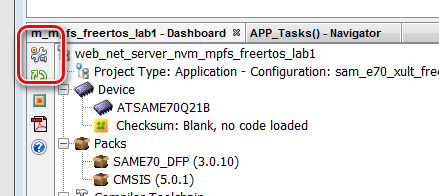
## Lab Procedure

### Starting MPLAB X IDE

* 1. Start MPLAB X IDE by double clicking on the MPLAB X IDE vv5.20.04 icon found on the Windows desktop.

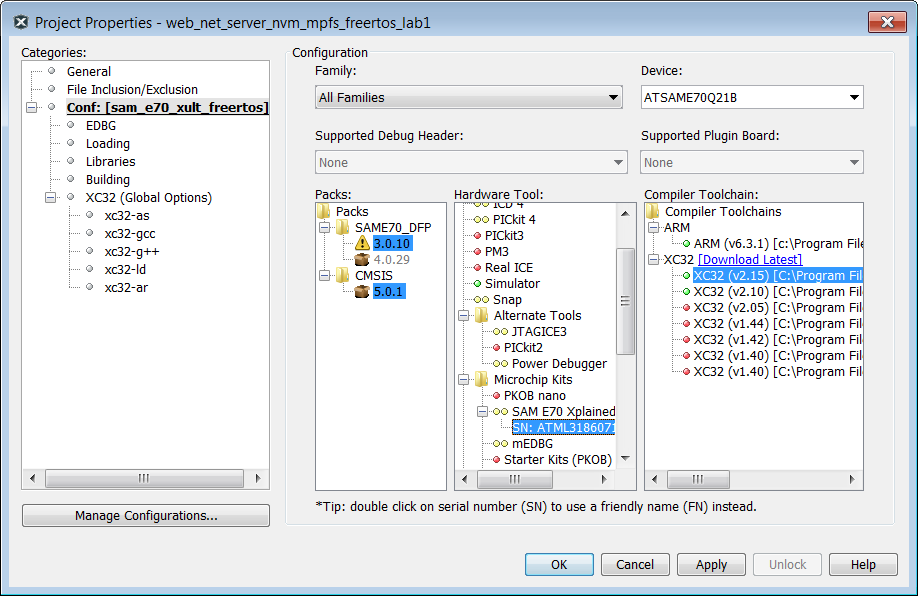
|  |
| --- |
|  |
|  |

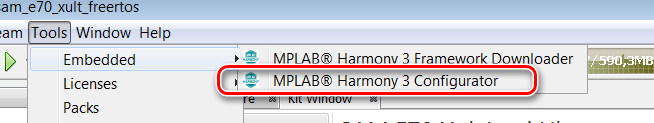
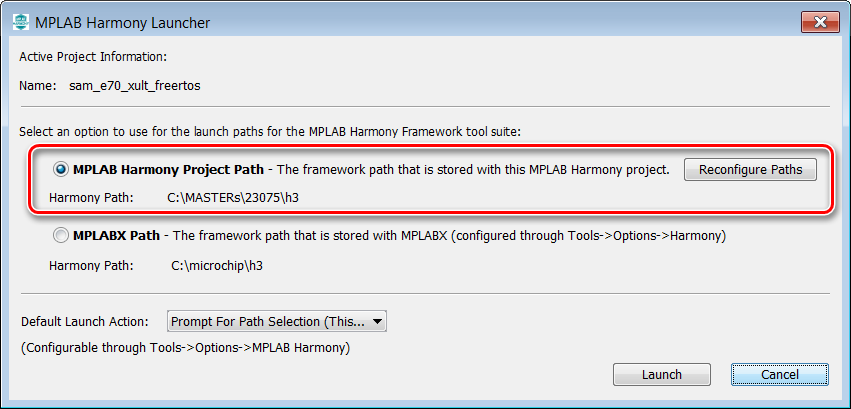
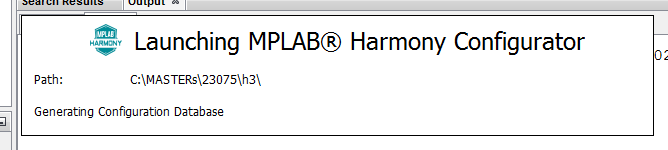
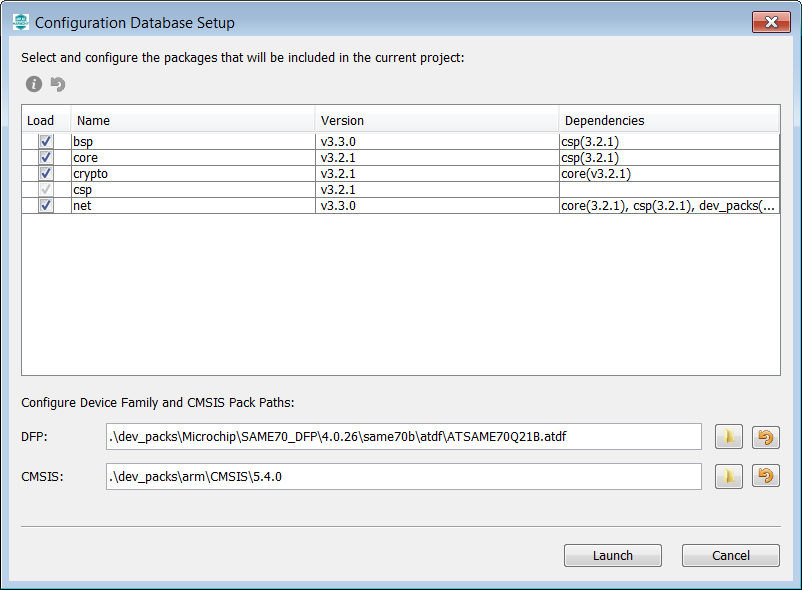
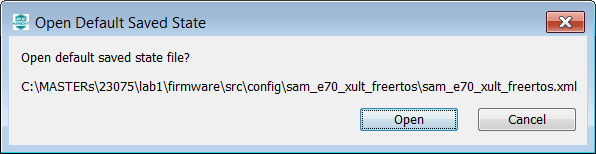
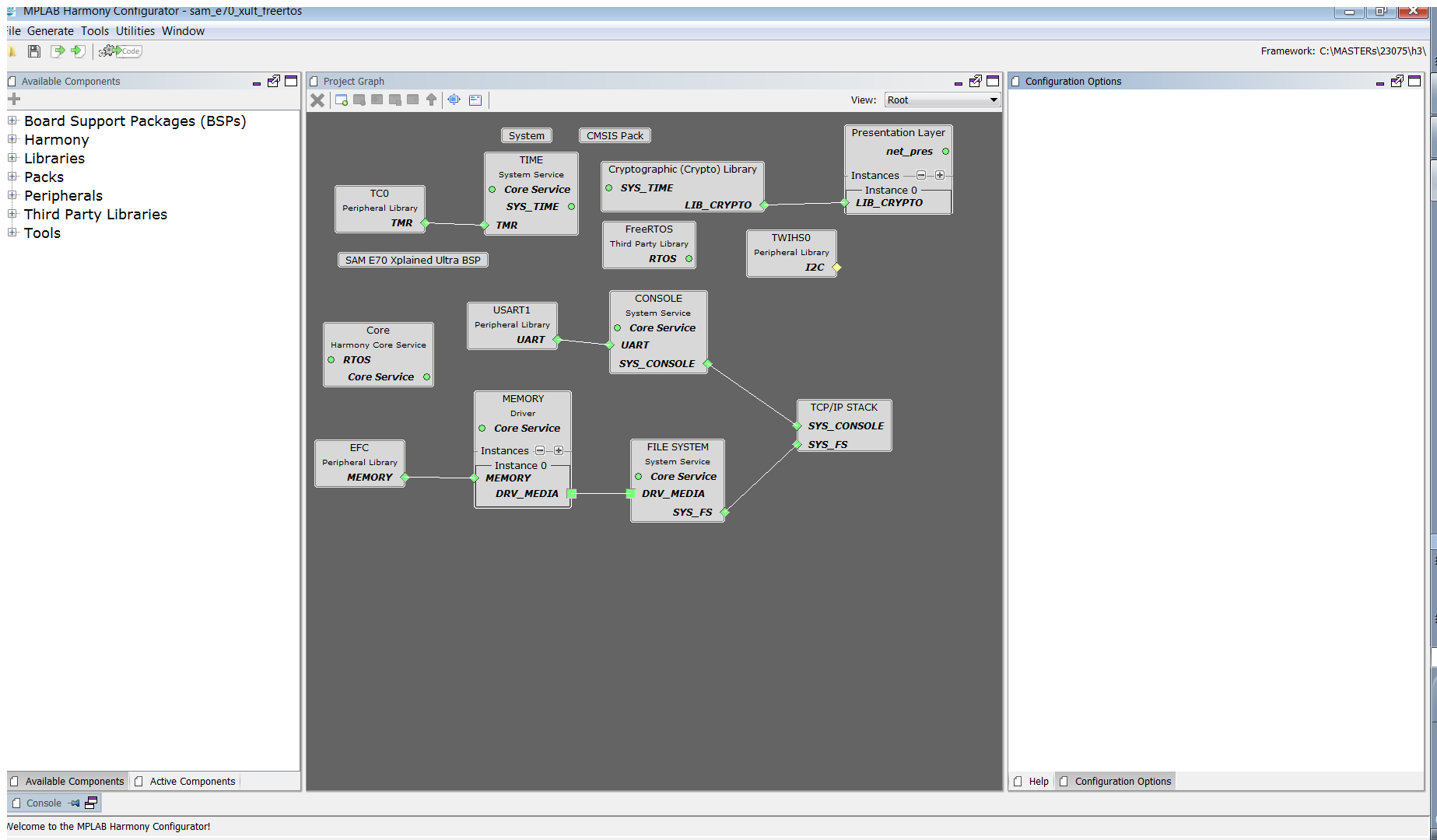
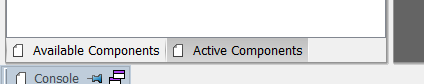
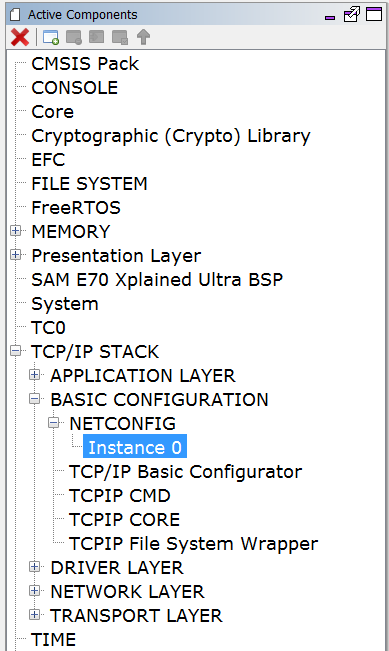
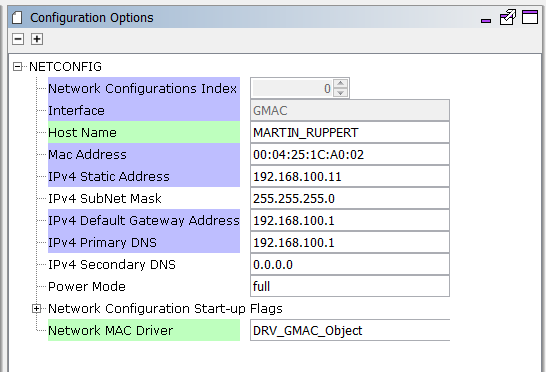
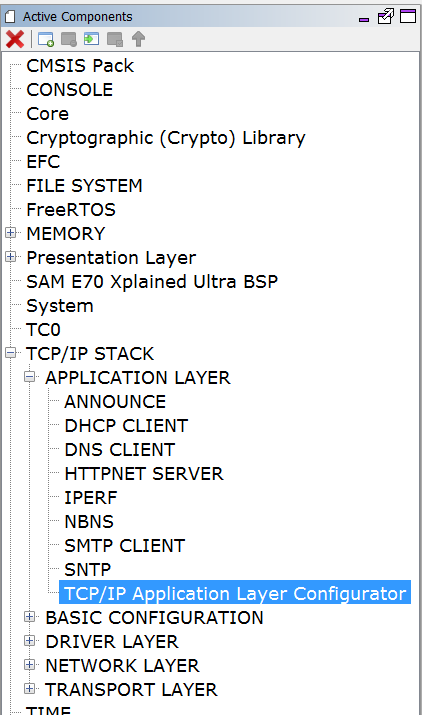
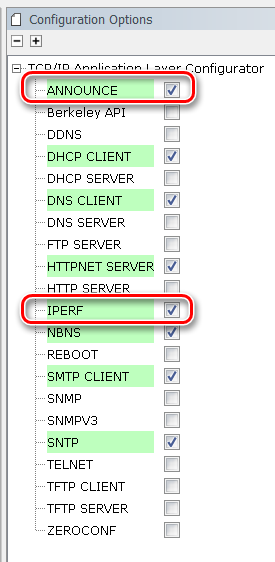
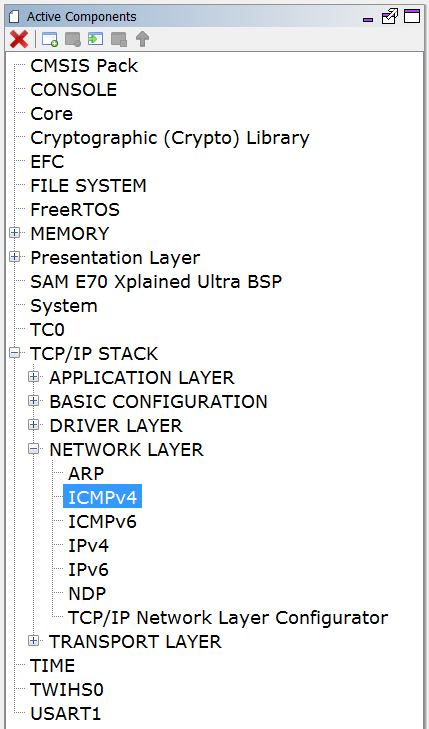
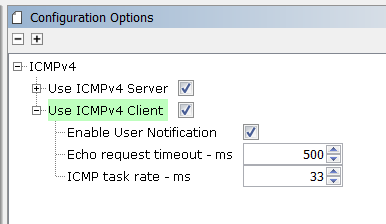
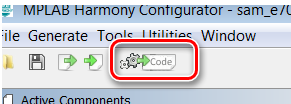
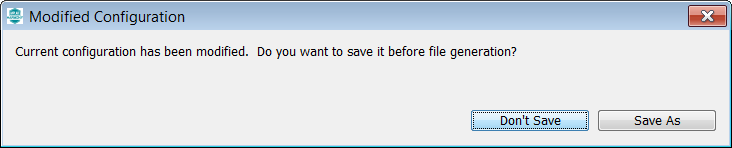
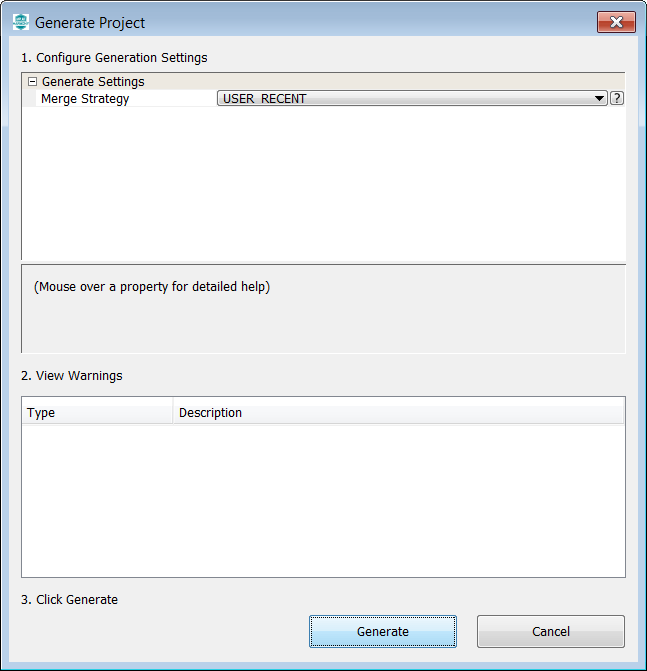
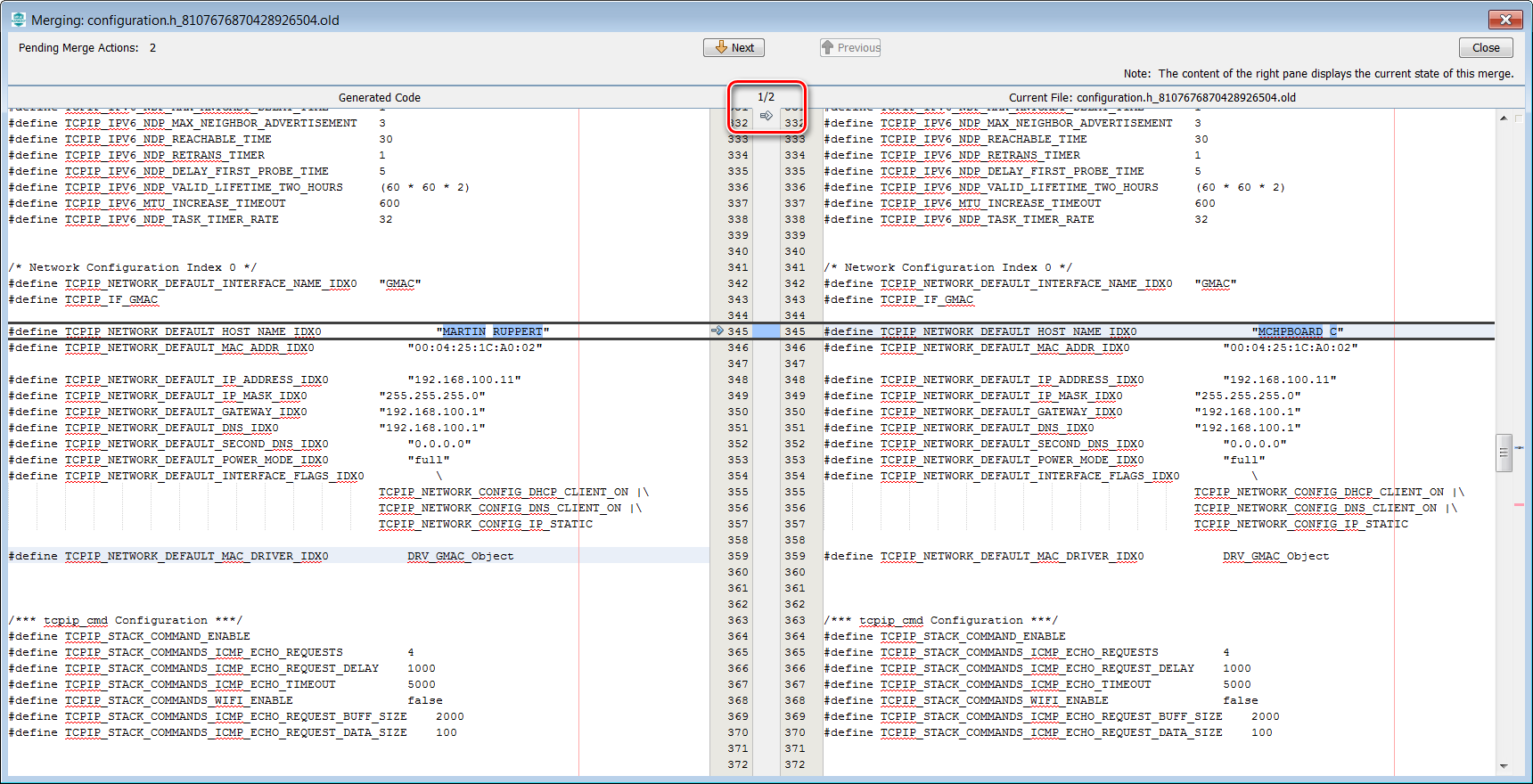
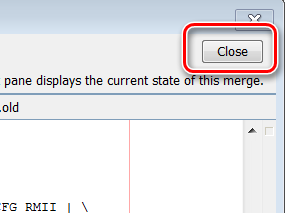
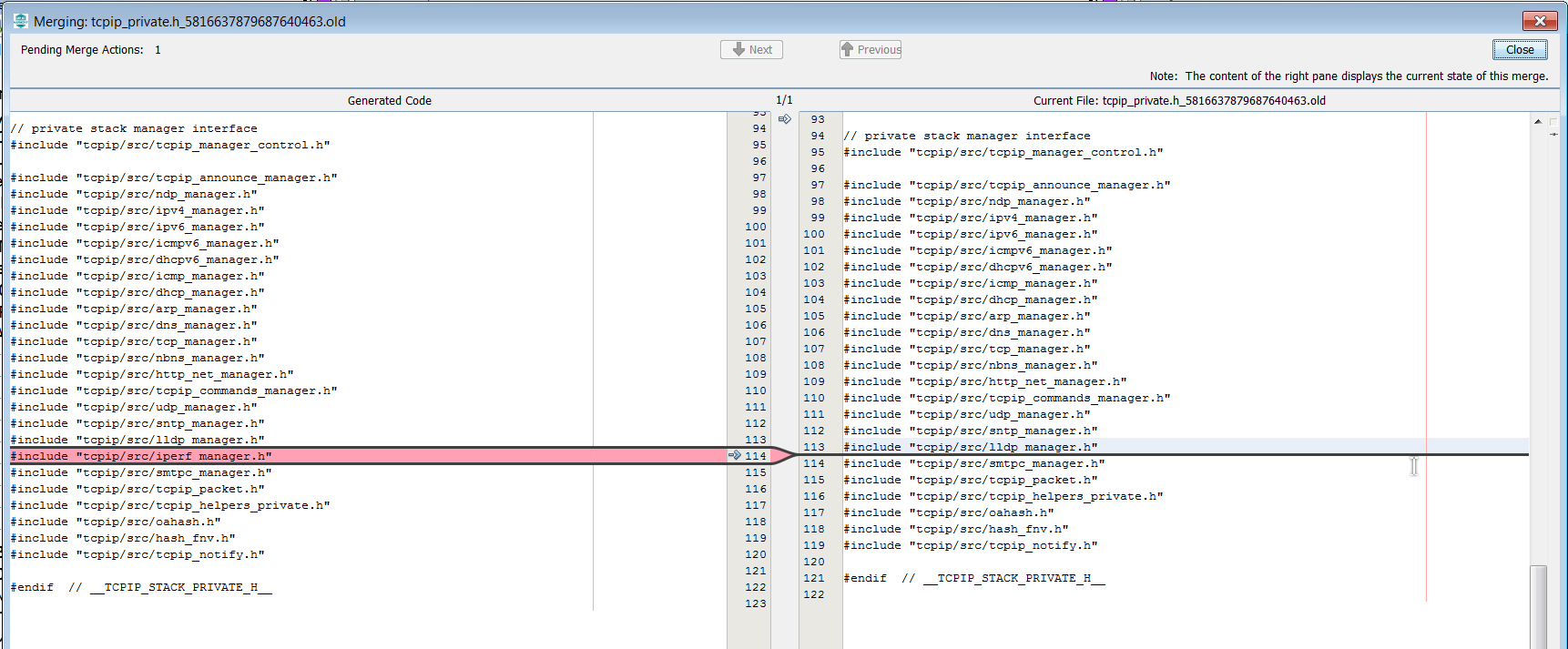
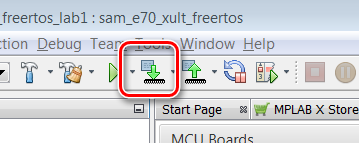
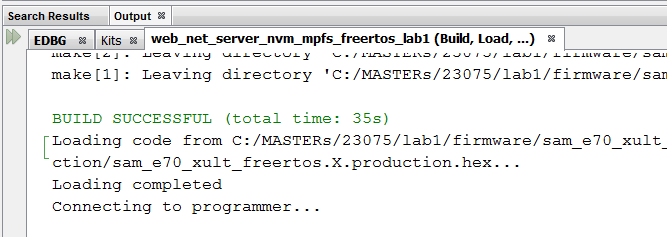
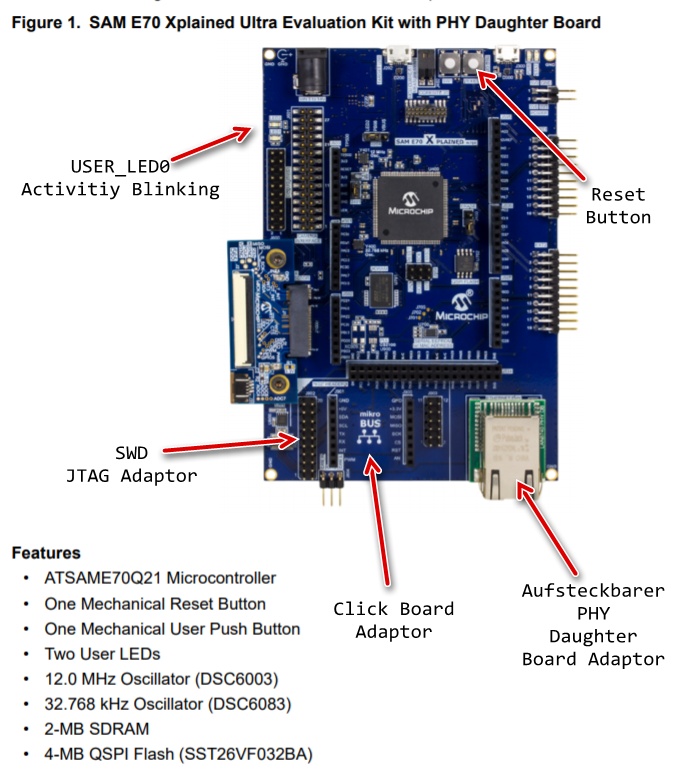
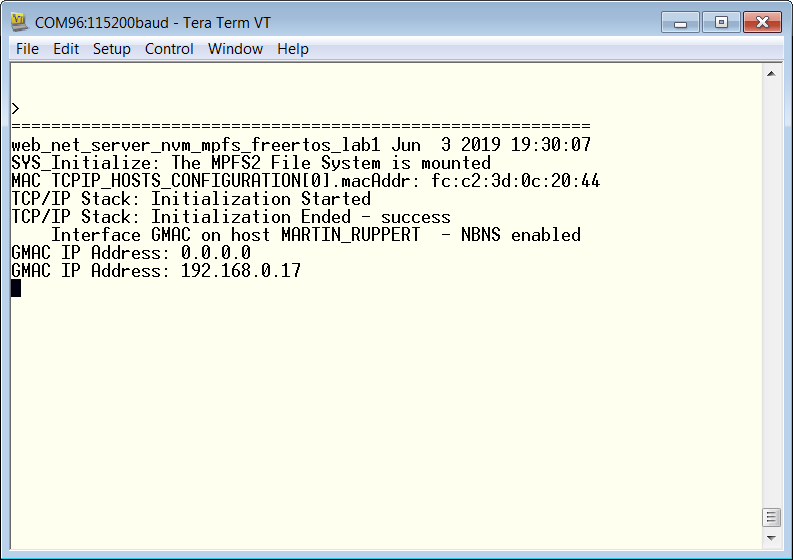
### Project Load, modify with MHC, Generate, build and run

1. Open Project by choosing File⮞Open Project… from the main menu and select lab1 project
2. Open Project Properties by choosing
   1. File⮞Project Properties from the main menu
   2. Or select with a right click the project node in the project windows and select at the bottom  
      
   3. Or select the toolbox in the Dashboard  
      

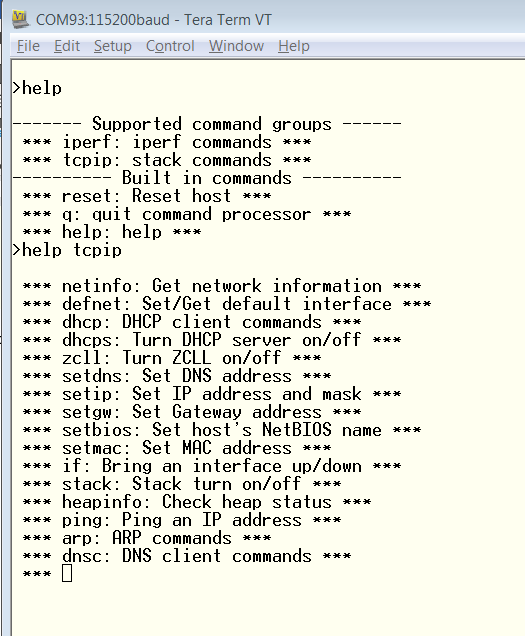


1. Select the XC32 Compiler v2.15
2. Select the SAME70 Xplained by clicking on the SN: Number



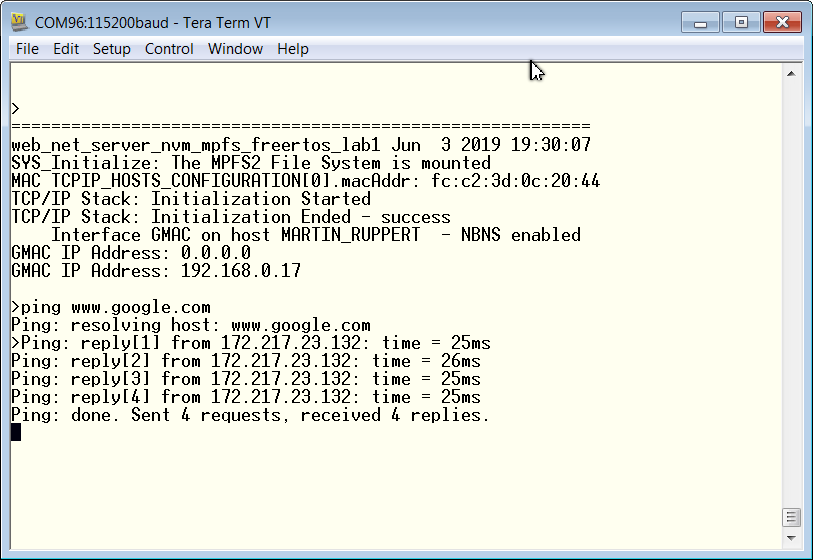
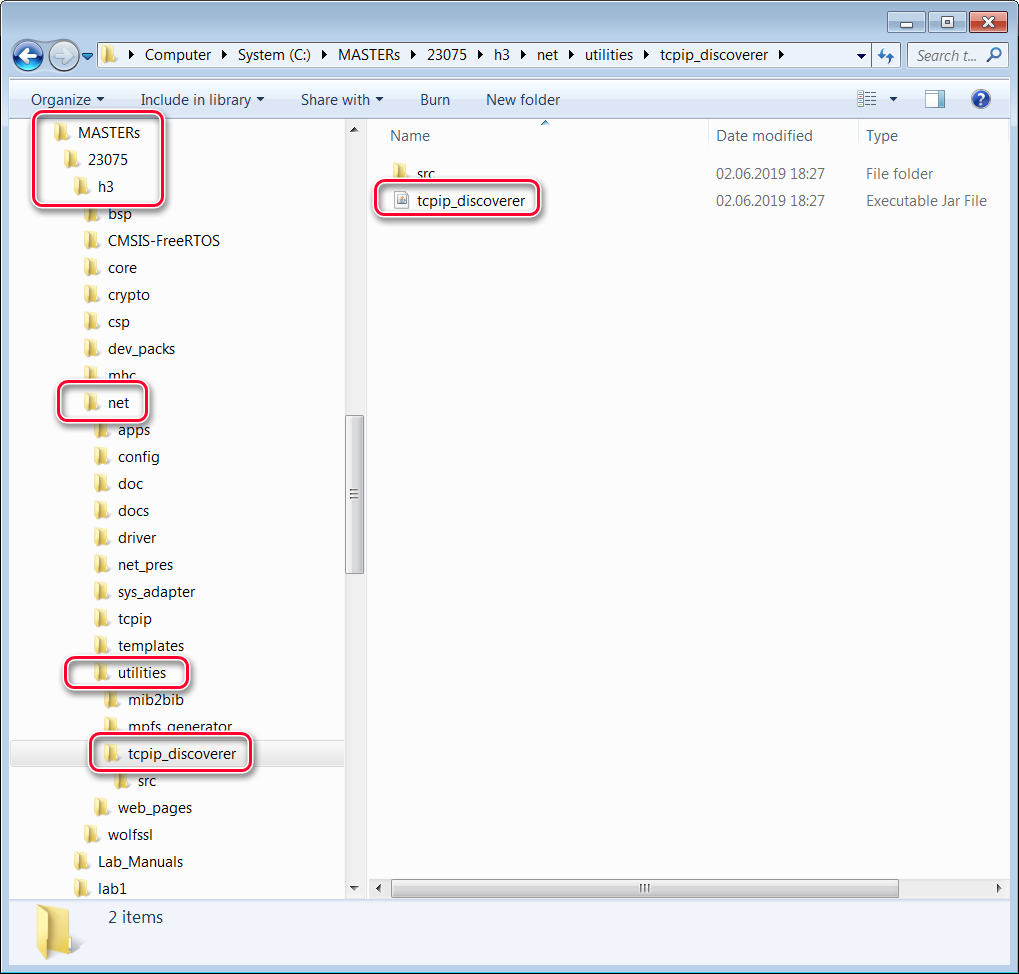
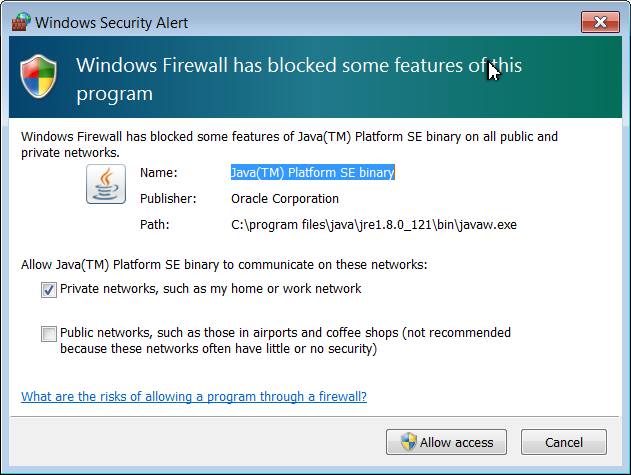
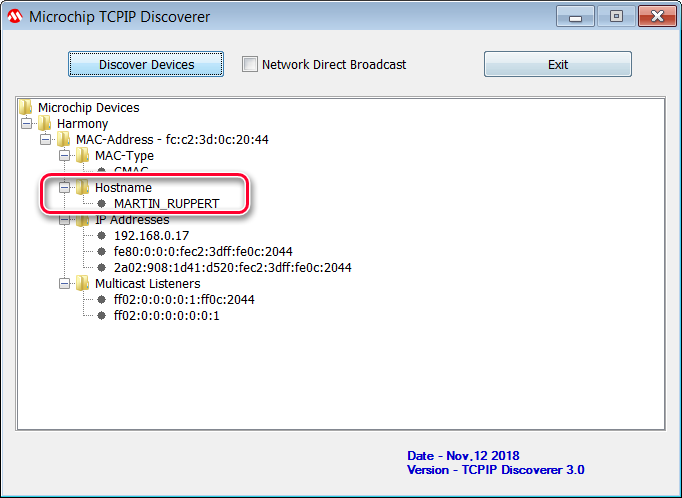
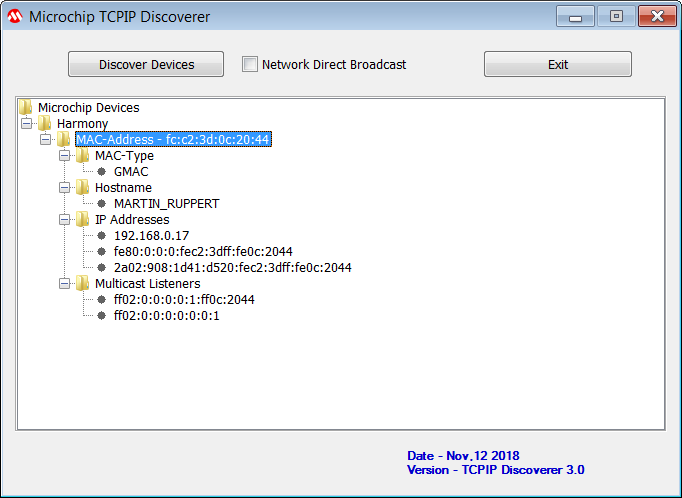
1. Select from the Tools Menu the Harmony 3 Configurator  
   
2. Ensure the H3 Path is set to “C:\MASTERs\23075\h3\”  
   
3. Select “Launch”  
     
   The first time the MHC is started, it can take up to 2 Minutes before the Configuration Database is prepared
4. In the next window the H3 parts and their used Version Numbers are displayed  
     
   Click on “Launch”
5. Open the saved state file  
   
6. The MHC is up and running  
   
7. Select Active Components (left below)  
   
8. Select the Instance 0 in Active Components  
   
9. And change in the Configuration Options (on the right side) the Host Name to something meaningful for you.   
   The Host name can be identified in the Network.  
   
10. Select “TCP/IP Application Layer Configuration” in Active Components   
    
11. Ensure that ANNOUNCE and IPERF are selected  
    
12. Select ICMPv4 in the Active Components  
    
13. And ensure that the “Use ICMPv4 Client” is selected  
    
14. Select “Code” (Generate Code)  
    
15. Select Don’t Save  
    
16. Select Generate  
    
17. Some Files will be changed and the MHC is asking in a “diff” window, if the changes should be taken over.  
    Accept all changes in the file by clicking on the Arrow in the middle above.
18. Then click on close in the upper right corner  
    
19. Same for the next 2 diff windows  
    
20. The whole process is displayed with a progress bar  
    
21. Back again in the main window of MPLABX, click on the “Make and Program Device” button  
    
22. After successful build, the SAME70 is programmed automatically   
    
23. The programming take about 30 seconds with on Board DebuggeCheck whether the USER\_LED0 Activity is Blinking  
      
      
      
      
    
24. Open Terra Term Terminal Program  and select under Setup->Serial Port the COM Port (in this case a COM96, but could be a different COM Port number in your case).  
      
    The press the Reset Button of the Board and let the Firmware restart. You should see the start message with the Lab name and the build timestamp at first, followed by the MAC Address from EEPROM.
25. After some small time, the IP Address should be change from 0.0.0.0 to a valid address  
    
26. To check the basic information about the network enter the netinfo command and press Enter.

|  |
| --- |
|  |

1. A help shows the available commands  
   

|  |
| --- |
|  |

|  |
| --- |
|  |

1. As a first simple test you can ping an external Server or anything else you like (Maybe the board of your class neighbor?)  
   
2. Select the **tcpip\_discoverer** tool from   
   
3. If Windows is asking for permissions allow the access  
   
4. The TCP Discover should list all boards in the classroom. We have made this tool to help you to find your board in the network. The source codes of this tool (Java) are part of the H3.   
     
   You can identify your board by the Host Name that has select in an earlier step. The Host Name is also known to the DHCP server and is listed in their typical Web Interfaces as a connected device.   
     
     
   Press the Discover Devices button: The tool will send a UDP broadcast on port 30303, with the packet “Discovery, who is out there?” All H3 devices running the Announce service will respond to this broadcast, by sending a return broadcast on port 30303. The broadcast packet contains data on the type of interface used, the Host Name, MAC and IP Address. The Discover tool listens to all broadcasts on port 30303 and will show found devices under the Microchip Devices tree. You can identify your device by looking for the host name that you entered in MHC Setup process. The Microchip TCPIP Discoverer tool also shows the IP address for your board.
5. A double click on the **MAC-Address** line will put you in your default Internet Browser  
   





























































































































































































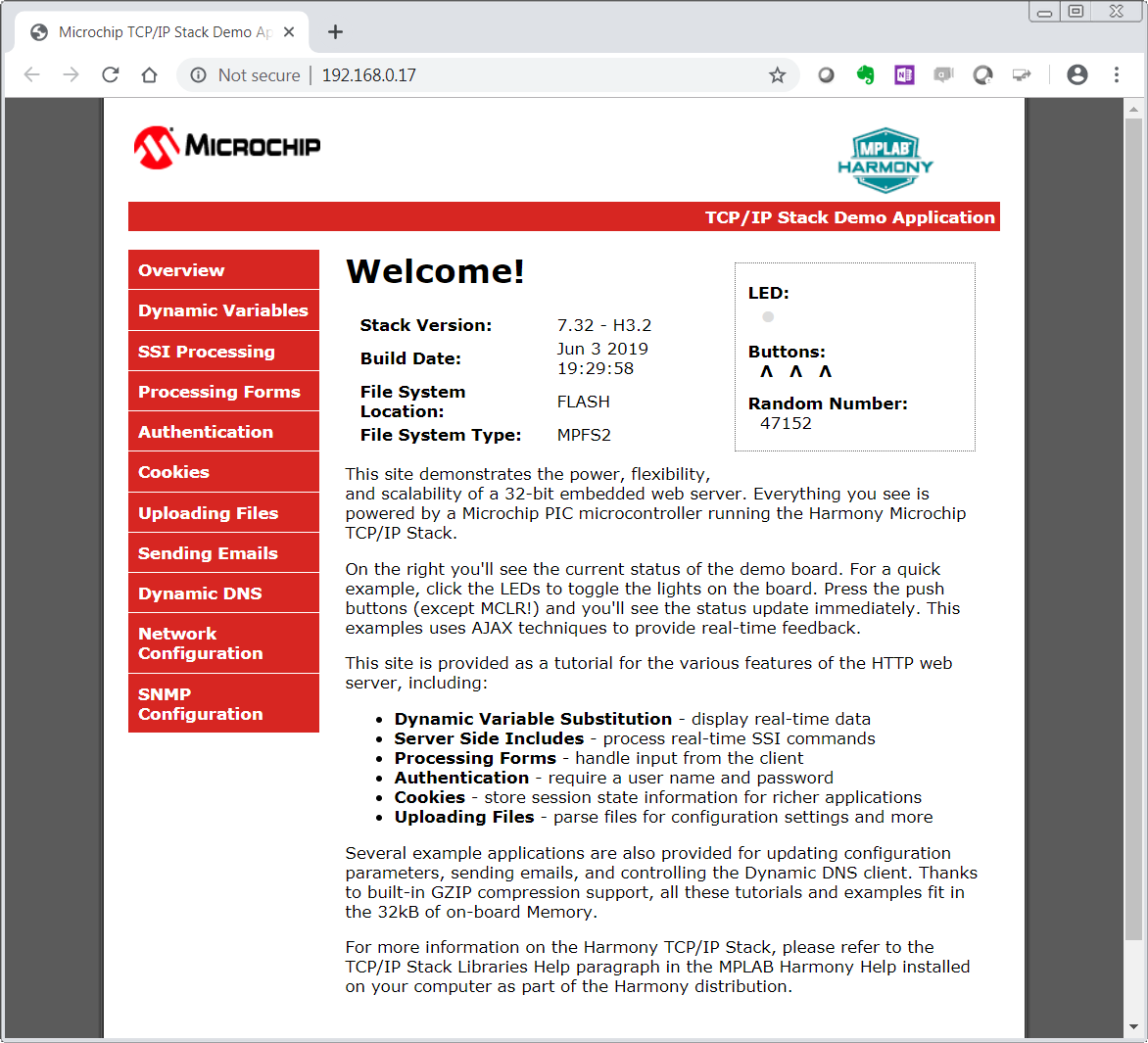










1. And the Webpage is displayed. Please take some time an play with the sub menus to find out the capabilities of or H3 Web Server  
   Congratulations, you have completed Lab 1!











































































































































































































































































































# Lab 2

## Overview

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

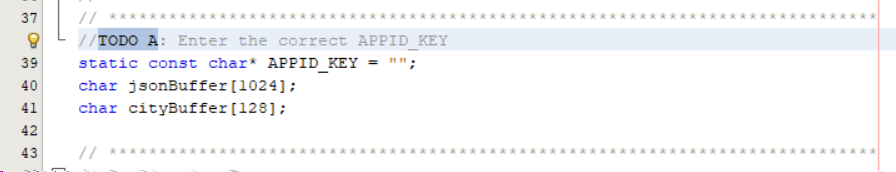
Machine generated alternative text:
Initiate 
Harmony, 
TCPIP and 
Application 
Print resulting 
JSON and 
Current 
Weather of 
chose city 
Choose City 
to retrieve 
data from 
Parse 
retrieved 
JSON data 
Choose 
which City 
to retrieve 
data from 

Lab Outline

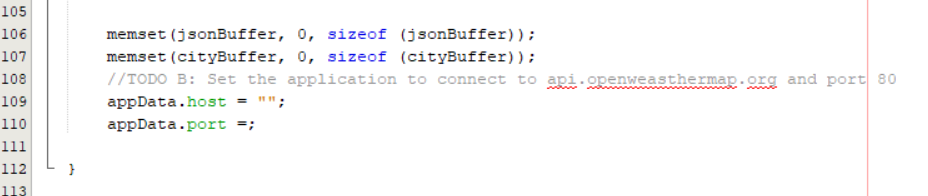
* The pre-made template is built from the Harmony example project, tcpip\_tcp\_client.
* First, we will need to declare the APPID\_KEY.
* We will then set the host & port of the remote connection static as we will only connect to OpenWeatherMap.
* After this, we will redirect the user input from the 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

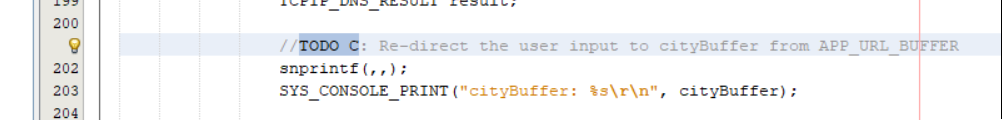
1. Start by closing any open projects in MPLAB X IDE.
2. Open a new project and choose lab3 -> Firmware -> sam\_e70\_xult\_freertos.X.
3. Open the file app.c located under source files.
4. Go to (CTRL+F) “TODO A”. Enter the correct APPID\_KEY. Either you create your own account on OpenWeatherMap or you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

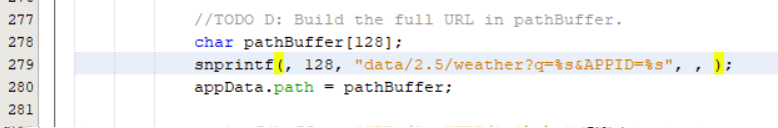


1. Re-direct the user input from APP\_URL\_BUFFER to the cityBuffer array. This can be done in several ways, but one is to use the built-in C function **snprintf**(char\* dest, size\_t size, const char \*format, …). The first argument is the destination buffer (cityBuffer), the second one is the max 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”.

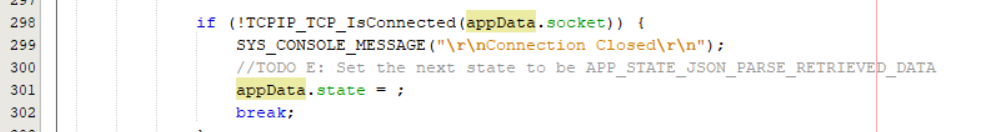


1. Scroll down to “TODO D”, the state APP\_TCPIP\_WAIT\_FOR\_CONNECTION. In this state we will wait for a connection to be established. Once established we will send a GET command with the full URL in the format specified in the introduction:

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



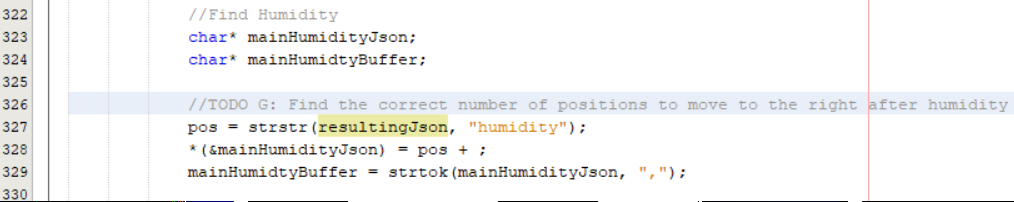
1. Once the request is sent to the server, the application will go into the APP\_TCIPIP\_WAIT\_FOR\_RESPONSE state. Once the connection is closed, set the next state to be APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA.



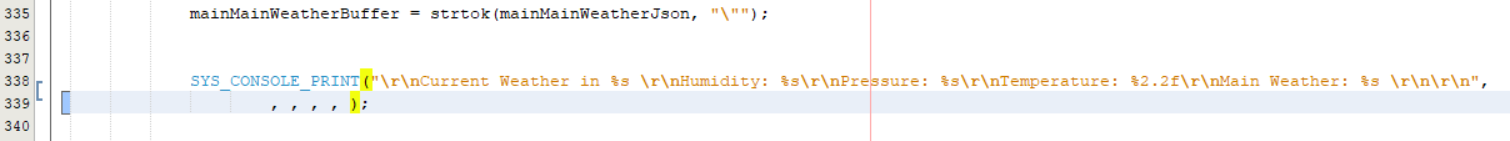
1. Now go down in the state APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA. One of the first things we want to do after we have sorted out the JSON-part of the retrieved data is to print the raw JSON-string. This helps us debug & analyse.



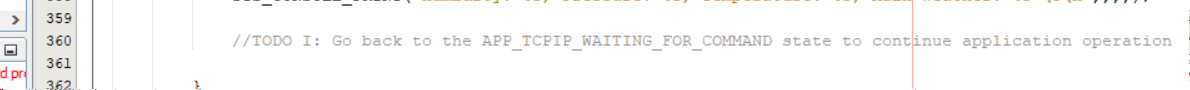
1. In a real application, we would need to first know the format of the JSON message in order to be able to parse it correctly. To make this lab more efficiently, we will do this backwards. If you look on this example piece of API response from OpenWeatherMap found in the introduction section to this 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.



# MPLAB® Harmony TCP/IP Stack

## TCP Module API Function List

### Socket Management Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_ServerOpen** | Opens a TCP socket as a server. |
| **TCPIP\_TCP\_ClientOpen** | Opens a TCP socket as a client. |
| **TCPIP\_TCP\_Close** | Disconnects an open socket and destroys the socket handle, releasing the associated resources. |
| **TCPIP\_TCP\_Connect** | Connects a client socket. |
| **TCPIP\_TCP\_Bind** | Binds a socket to a local address. |
| **TCPIP\_TCP\_RemoteBind** | Binds a socket to a remote address. |
| **TCPIP\_TCP\_IsConnected** | Determines if a socket has an established connection. |
| **TCPIP\_TCP\_WasReset** | Self-clearing semaphore indicating socket reset. |
| **TCPIP\_TCP\_Disconnect** | Disconnects an open socket. |
| **TCPIP\_TCP\_Abort** | Aborts a connection. |
| **TCPIP\_TCP\_OptionsGet** | Allows getting the options for a socket like: current RX/TX buffer size, etc. |
| **TCPIP\_TCP\_OptionsSet** | Allows setting options to a socket like adjust RX/TX buffer size, etc. |
| **TCPIP\_TCP\_SocketInfoGet** | Obtains information about a currently open socket. |
| **TCPIP\_TCP\_SocketNetGet** | Gets the current network interface of an TCP socket. |
| **TCPIP\_TCP\_SocketNetSet** | Sets the interface for an TCP socket |
| **TCPIP\_TCP\_SignalHandlerDeregister** | Deregisters a previously registered TCP socket signal handler. |
| **TCPIP\_TCP\_SignalHandlerRegister** | Registers a TCP socket signal handler. |
| **TCPIP\_TCP\_Task** | Standard TCP/IP stack module task function. |

### Transmit Data Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_Put** | Writes a single byte to a TCP socket. |
| **TCPIP\_TCP\_PutIsReady** | Determines how much free space is available in the TCP TX buffer. |
| **TCPIP\_TCP\_StringPut** | Writes a null-terminated string to a TCP socket. |
| **TCPIP\_TCP\_ArrayPut** | Writes an array from a buffer to a TCP socket. |
| **TCPIP\_TCP\_Flush** | Immediately transmits all pending TX data. |
| **TCPIP\_TCP\_FifoTxFullGet** | Determines how many bytes are pending in the TCP TX FIFO. |
| **TCPIP\_TCP\_FifoTxFreeGet** | Determines how many bytes are free and could be written in the TCP TX FIFO. |

### Receive Data Transfer Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_ArrayFind** | Searches for a string in the TCP RX buffer. |
| **TCPIP\_TCP\_Find** | Searches for a byte in the TCP RX buffer. |
| **TCPIP\_TCP\_Get** | Retrieves a single byte to a TCP socket. |
| **TCPIP\_TCP\_Peek** | Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer. |
| **TCPIP\_TCP\_Discard** | Discards any pending data in the RCP RX FIFO. |
| **TCPIP\_TCP\_FifoRxFreeGet** | Determines how many bytes are free in the RX buffer/FIFO. |
| **TCPIP\_TCP\_FifoSizeAdjust** | Adjusts the relative sizes of the RX and TX buffers. |
| **TCPIP\_TCP\_FifoRxFullGet** | Determines how many bytes are pending in the RX buffer/FIFO. |
| **TCPIP\_TCP\_GetIsReady** | Determines how many bytes can be read from the TCP RX buffer. |
| **TCPIP\_TCP\_ArrayGet** | Reads an array of data bytes from a TCP socket's RX buffer/FIFO. |
| **TCPIP\_TCP\_ArrayPeek** | Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer. |

## 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. |

# Harmony TCP/IP API Subset For all Lab’s

## TCP Socket Management Functions

## TCPIP\_TCP\_ArrayGet Function

This function reads an array of data bytes from a TCP socket's RX buffer/FIFO. The data is removed from the FIFO in the process.

### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_ArrayGet**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**,  uint8\_t\* **buffer**,  uint16\_t **len**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| ****hTCP**** | The socket from which data is to be read. |
| ****buffer**** | Pointer to the array to store data that was read. |
| ****len**** | Number of bytes to be read. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected. |

### TCPIP\_TCP\_ClientOpen Function

Provides a unified method for opening TCP client sockets. Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP\_TCP\_Abort or TCPIP\_TCP\_Close. If the remoteAddress != 0 (and the address pointed by remoteAddress != 0) then the socket will immediately initiate a connection to the remote host.

#### Function Prototoype

|  |
| --- |
| [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **TCPIP\_TCP\_ClientOpen**(  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType**,  [TCP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22208.html) **remotePort**,  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* **remoteAddress**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **addType** | The type of address being used. Valid values are: IP\_ADDRESS\_TYPE\_IPV4 or IP\_ADDRESS\_TYPE\_IPV6 |
| **remotePort** | TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module. |
| **remoteAddress** | The remote address to be used |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) | Handle - Save this handle and use it when calling all other TCP APIs. If no sockets of the specified type were available to be opened, the handle will contain a value equal to INVALID\_SOCKET. |

### TCPIP\_TCP\_Close Function

Graceful Option Set: If the graceful option is set for the socket (default), a TCPIP\_TCP\_Disconnect will be tried. If the linger option is set (default) the TCPIP\_TCP\_Disconnect will try to send any queued TX data before issuing FIN. If the FIN send operation fails or the socket is not connected the abort is generated.

Graceful Option Not Set: If the graceful option is not set, or the previous step could not send the FIN, a TCPIP\_TCP\_Abort is called, sending a RST to the remote node. Communication is closed, the socket is no longer valid and the associated resources are freed.

#### Function Prototype

|  |
| --- |
| **void** **TCPIP\_TCP\_Close**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP socket should have been opened with TCPIP\_TCP\_ServerOpen/TCPIP\_TCP\_ClientOpen.

hTCP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | Handle to the socket to disconnect and close. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| Void | None |

### TCPIP\_TCP\_GetIsReady Function

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_GetIsReady**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes available to be read from the TCP RX buffer. |

### TCPIP\_TCP\_IsConnected Function

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP\_TCP\_ServerOpen()/TCPIP\_TCP\_ClientOpen() to determine when the connection is set up and ready for use.

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_TCP\_IsConnected**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | The TCP socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | True: the socket is connected  False: the socket is disconnected |

### TCPIP\_TCP\_PutIsReady Function

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_PutIsReady**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket from which data is to be written. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes available to be written in the TCP TX buffer. |

### TCPIP\_TCP\_StringPut Function

This function writes a null-terminated string to a TCP socket. The null-terminator is not copied to the socket.

#### Function Prototype

|  |
| --- |
| **const** uint8\_t\* **TCPIP\_TCP\_StringPut**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**,  **const** uint8\_t\* **Data**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket from which data is to be written. |
| **const** uint8\_t\* | **Data** |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| **const** uint8\_t\* | Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NULL byte, the buffer became full or the socket is not connected. |

### TCPIP\_TCP\_WasReset Function

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function works for all possible disconnections: a call to [TCPIP\_TCP\_Disconnect](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24093.html), a FIN from the remote node, or an acknowledgment timeout caused by the loss of a network link. It also returns true after the first call to TCPIP\_TCP\_Initialize. Applications should use this function to reset their state machines.

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_TCP\_WasReset**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | The TCP socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | true: the socket was disconnected since the previous call  false: the socket remained connected since the previous call |

## UDP Socket Management Functions

### TCPIP\_UDP\_ArrayGet Function

This function reads an array of bytes from the UDP socket, while adjusting the current read pointer and decrementing the remaining bytes available. TCPIP\_UDP\_GetIsReady should be used before calling this function to get the number of the available bytes in the socket.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_UDP\_ArrayGet**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**,  uint8\_t \* **cData**,  uint16\_t **wDataLen**  ); |

#### Preconditions

UDP socket should have been opened with TCPIP\_UDP\_ServerOpen/TCPIP\_UDP\_ClientOpen.

hUDP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |
| cData | The buffer to receive the bytes being read. If NULL, the bytes are simply discarded |
| wDataLen | Number of bytes to be read from the socket. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [uint16\_t](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available. |

### TCPIP\_UDP\_Close Function

Closes a UDP socket and frees the handle. Call this function to release a socket and return it to the pool for use by future communications.

#### Function Prototoype

|  |
| --- |
| **void** **TCPIP\_UDP\_Close**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**  ); |

#### Preconditions

UDP socket should have been opened with TCPIP\_UDP\_ServerOpen/TCPIP\_UDP\_ClientOpen.

hUDP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [void](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | None |

### TCPIP\_UDP\_GetIsReady Function

This function will return the number of bytes that are available in the specified UDP socket RX buffer. The UDP socket queues incoming RX packets in an internal queue. If currently there is no RX packet processed (as a result of retrieving all available bytes with TCPIP\_UDP\_ArrayGet, for example), this call will advance the RX packet to be processed to the next queued packet. If a RX packet is currently processed, the call will return the number of bytes left to be read from this packet.

**Function Prototype**

|  |
| --- |
| uint16\_t **TCPIP\_UDP\_GetIsReady**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**  ); |

#### Preconditions

UDP socket should have been opened with [TCPIP\_UDP\_ServerOpen](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24228.html)/[TCPIP\_UDP\_ClientOpen](mk:@MSITStore:D:\\microchip\\harmony\\v1_08\\doc\\help_harmony.chm::/24205.html).

hUDP parameter is a valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [uint16\_t](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | The number of bytes that can be read from the socket. |

### TCPIP\_UDP\_ServerOpen Function

Provides a unified method for opening UDP server sockets.

#### Function Prototype

|  |
| --- |
| [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **TCPIP\_UDP\_ServerOpen**(  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType**,  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **localPort**,  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* **localAddress**  ); |

#### Preconditions

UDP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType** | The type of address being used.  IP\_ADDRESS\_TYPE\_IPV4 or IP\_ADDRESS\_TYPE\_IPV6. |
| [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) localPort | UDP port on which to listen for connections |
| [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* localAddress | Local IP address to use. Can be 0 (NULL) if any incoming interface will do. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | Handle - Save this handle and use it when calling all other UDP APIs. If no sockets of the specified type were available to be opened, the handle will contain a value equal to INVALID\_SOCKET. |

### TCPIP\_UDP\_SocketInfoGet Function

This function will fill a user passed [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) structure with status of the selected socket

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_UDP\_SocketInfoGet**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**,  [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html)\* **pInfo**  ); |

#### Preconditions

UDP socket should have been opened with [TCPIP\_UDP\_ServerOpen](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24228.html)()/[TCPIP\_UDP\_ClientOpen](mk:@MSITStore:D:\\microchip\\harmony\\v1_08\\doc\\help_harmony.chm::/24205.html)()().

hUDP - valid socket

pInfo - valid address of a [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) structure

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |
| pInfo | Pointer to [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) to receive socket information |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | true if call succeeded  false if no such socket or invalid pinfo. |

### UDP\_SOCKET\_INFO Structure

Holds information about a UDP Socket

#### Structure

|  |
| --- |
| **typedef** **struct** {  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addressType**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **remoteIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **localIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **sourceIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **destIPaddress**;  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **remotePort**;  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **localPort**;  [TCPIP\_NET\_HANDLE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/23778.html) **hNet**;  } UDP\_SOCKET\_INFO; |

#### Members

|  |  |  |
| --- | --- | --- |
| **Type** | **Member Name** | **Description** |
| IP\_ADDRESS\_TYPE | addressType | address type of the socket |
| IP\_MULTI\_ADDRESS | remoteIPaddress | current socket destination address |
| IP\_MULTI\_ADDRESS | localIPaddress | current socket source address |
| IP\_MULTI\_ADDRESS | sourceIPaddress | source address of the last packet |
| IP\_MULTI\_ADDRESS | destIPaddress | destination address of the last packet |
| UDP\_PORT | remotePort | Port number associated with remote node |
| UDP\_PORT | localPort | local port number |
| TCPIP\_NET\_HANDLE | hNet | associated interface |















