# Ag Irrigation Remote Control

*Programmers Manual*

***By: TGIT-Tech***

***Visit my website at: http://www.tgit-tech.com/***



Last Updated: 2020.05.24

# Table of Contents

# 1. INTRODUCTION

## 1.1 Whats Covered

- ✔ **This booklet only needs to be read if**:
  - ○ The project is built from scratch.
    - ▪ Requiring firmware to be uploaded to both the Hand-Remote and Pump-Controller
  - ○ The current setup has changed
    - ▪ Adding new accessories like a digital pressure meter or ultrasonic meter ( water-level ) requires firmware that reflects those additions
  - ○ Special operation is required
    - ▪ Experienced programmers can program the units to operate however they wish

- ✔ This booklet is a guide for
  - ○ Configuring the Xbee RF Module
  - ○ Uploading Firmware onto the Hand-Remote and Pump-Controller's micro-controllers
  - ○ Details about customizing the firmware to match a users needs

## 1.2 About the Ag Irrigation Remote Control

The "***Ag Irrigation Remote Control***" project's goal is to provide RF ( Radio Frequency ) remote control and monitoring abilities to an Agriculture-Industrial 3-Phase 480VAC Irrigation Pump that is typically used in wheel-line and hand-line irrigation setups within the United States.

- ✔ The project implements
  - ○ Digi Xbee RF radio modules ( S3B, 900MHz, 1/4-Watt ) which advertises a broadcast range of up to 28-miles ( *Practical usage at 2-miles at ground level* )
  - ○ Arduino micro-controller boards, LCD display shields, Zigbee shields, and various discrete devices

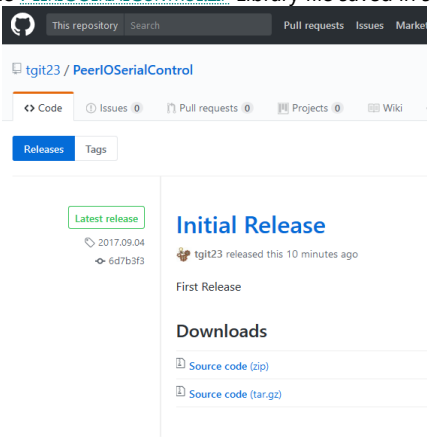
- ✔ The project consists of:
  - ○ Hand-Held "***Hand Remote***"
  - ○ Pole mounted "***Pump Controller***"
  - ○ High-Voltage Control "***Installation Kit***" which consists of components to be added to the pumps electrical panel

- ✔ The project is Open Source and its main page Resides at:
  https://github.com/TGit-Tech/AgIrrigationRemoteControl
  - ○ Releases ( All Finalized Plans, Files and Documentation ) can be downloaded at:
    https://github.com/TGit-Tech/AgIrrigationRemoteControl/releases
  - ○ The online manual can be seen at:
    https://tgit-tech.github.io/AgIrrigationRemoteControl/manual/

## 2. Uploading Firmware

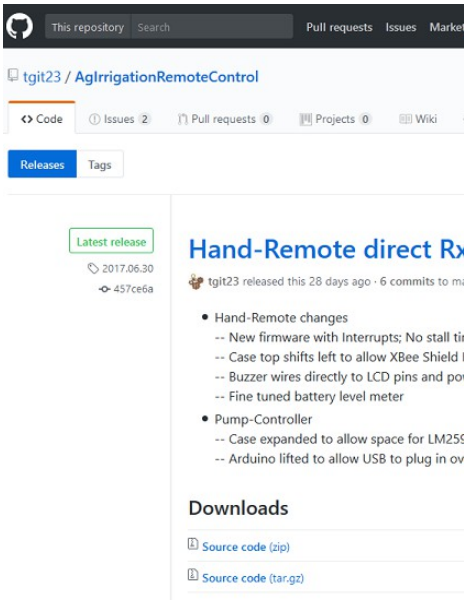
1. Install Arduino Sketch IDE
   a) In a web-browser; go to https://www.arduino.cc/en/Main/Software
   b) Scroll down to 'Download the Arduino IDE' and Select the Windows Installer
   c) Save Install file to a place you'll remember – like Documents or Desktop
   d) Run the saved filed to Install the Arduino Sketch IDE.

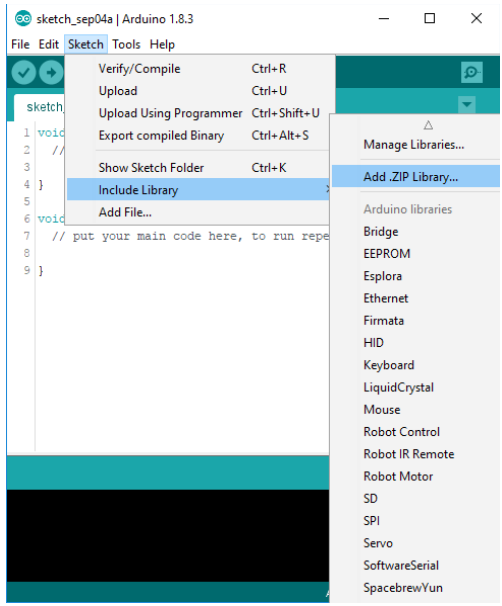

2. Install the PeerIOSerialControl Library
   a) In a web-browser; go to https://github.com/TGit-Tech/PeerIOSerialControl/releases
   b) Under Latest release → Downloads click on Source code (zip)
   c) Save to a place you'll remember – like Documents or Desktop
   d) Run the Arduino Sketch IDE installed in Step #1
   e) Choose Menu Item Sketch → Include Library → Add .zip Library
   f) Select the PeerIOSerialControl.zip Library file saved in step 'c' above




3. Install the SSoftwareSerial Library - Required by the Hand-Remote Firmware
   a) In a web-browser; go to https://github.com/TGit-Tech/SSoftwareSerial/releases
   b) Under Latest release → Downloads click on Source code (zip)
   c) Save to a place you'll remember – like Documents or Desktop
   d) Run the Arduino Sketch IDE installed in Step #1
   e) Choose Menu Item Sketch → Include Library → Add .zip Library
   f) Select the SSoftwareSerial.zip Library file saved in step 'c' above

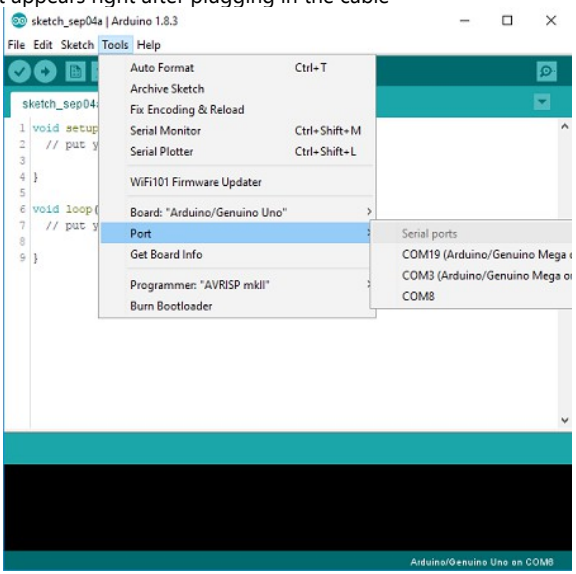
4. Download the Firmware
   a) In a web-browser; go to https://github.com/TGit-Tech/AgIrrigationRemoteControl/releases
   b) Under Latest release → Downloads click on Source code (zip)
   c) Save to a place you'll remember – like Documents or Desktop
   d) Unzip the Folder
   e) In Arduino Sketch IDE choose File → Open and Select
      - File HandRemote.ino for the HandRemote Firmware
        - ~\AgIrrigationRemoteControl-master\AgIrrigationRemoteControl-master\HandRemote
      - File PumpController.ino for the PumpController Firmware
        - ~\AgIrrigationRemoteControl-master\AgIrrigationRemoteControl-master\PumpController\PumpRemote



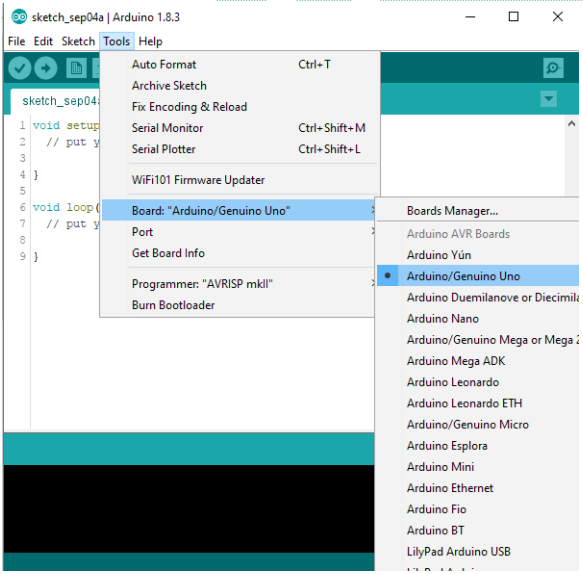



5. Compile and Upload the Firmware

a) Plug in a USB cable from the Computer to the unit
b) Select the Port the units USB is connected to; in Sketch menu TOOLS → PORT
- To determine Port; Open Windows Device Manager → Ports ( COM & LPT ) a new COM?? port appears right after plugging in the cable




c) Select the Board; Sketch menu TOOLS → BOARD → ARDUINO/GENUINO UNO




d) In Arduino Sketch IDE; Press the Right-Arrow next to the Check mark in the Top-Left Corner to upload the firmware onto the unit

## 2.1 Adding Expansions

1. Follow the steps in 2.Uploading Firmware to Step #4 for the Hand-Remote firmware; Hold off on Step #5; "Compile & Upload"

2. Once the file is open in Arduino Sketch IDE; locate the following code-block which should be right at the top

```
//=============================================================================================================
//------------------------------ SIMPLE USER CONFIGURATION SETTINGS -------------------------------
//=============================================================================================================
#define BUILD_VERSION                    20170622  // Release Version used to Build the Unit
( without the dots )
#define TRANSCEIVER_ID                   1         // Unique numeric (ID)entity for this Unit(1-15)
#define XBEECONFIG                       0         // 0 = RUN, 1 = Configure the XBEE using XCTU
#define ULTRASONIC_WATER_LEVEL_INSTALLED 0         // 0 = NO, 1 = YES ( Wire TRIG -> D4, ECHO -> D5 )
#define WATER_PRESSURE_INSTALLED         0         // 0 = NO, 1 = YES ( Wire SENSE -> A3 )

//^^^[ END - SIMPLE USER CONFIGURATION SETTINGS ]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

3. As described within the comments ( *text following '//'* )
   a) Change ULTRASONIC_WATER_LEVEL_INSTALLED to a '1' for water-level sensor ability.
   b) Change WATER_PRESSURE_INSTALLED to '1' for pressure sensor ability.
   c) Then proceed to Step #5 in 2.Uploading Firmware and Compile & Upload the changes just made

# 3. Xbee RF Configuration

The XBEE RF Module is setup from the factory to work without any additional changes. However, often it is very wise to assign non-factory setting in order to keep your RF communications from being interfered with by other XBEE devices in the neighborhood. Changes to the ID NETWORK ID ( described below ) will need to be preformed on all HAND-REMOTES and all PUMP-CONTROLLERS

*Generally, Digi XCTU will work with Pump-Controllers but occasionally there seems to be some issue with XCTU communicating with the Xbee modules on the Hand-Remotes. If XCTU cannot communicate properly with the Xbee module use section 3.3.Configuring Xbee using Putty as an alternative.*

## 3.1 Configuring Firmware for Xbee Configuration Changes

1. Upload Firmware with XBEE Configuration Set
   a) To change the XBEE settings – Follow Steps #1 to #3 in 2.Uploading Firmware

With the Firmware file open find the following line of code; which should be very close to the top

```
#define XBEECONFIG                       0         // 0 = RUN, 1 = Configure the XBEE using XCTU
```

Change it to

```
#define XBEECONFIG                       0         // 0 = RUN, 1 = Configure the XBEE using XCTU
```

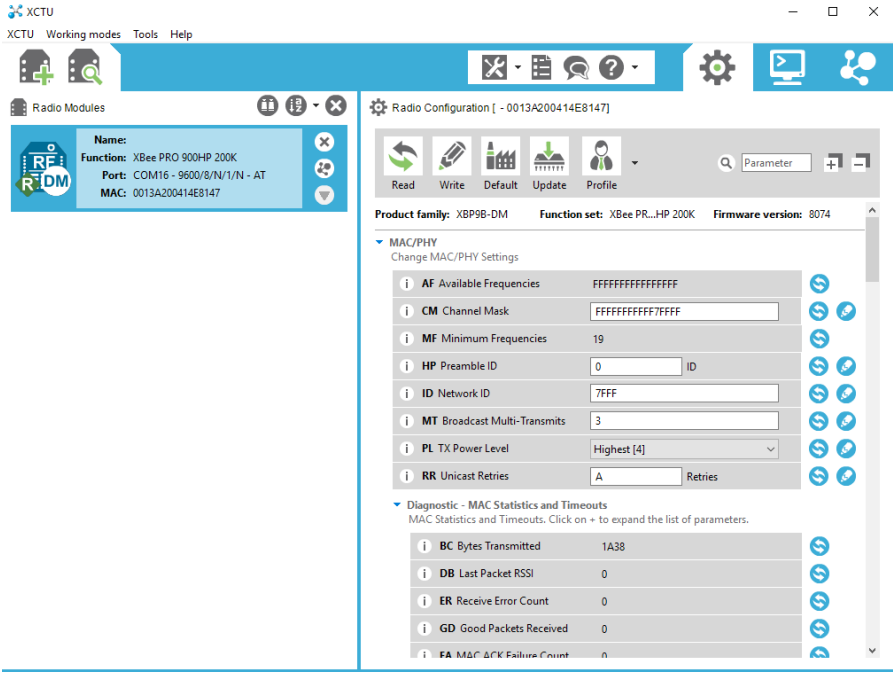
   b) Proceed to upload the firmware to the Unit as outline in Step #4 in 2.Uploading Firmware

## 3.2 Configuring Xbee using Digi XCTU

2. Install Digi-XCTU

a) The Digi XCTU software can be gotten from https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

3. Run Digi-XCTU




a) Click + ( Add Radio ) mark in the top left corner and enter the unit's COM port ( Arduino )
b) The main values that may be of interest are:
- ID Network ID - This sets a unique private "Channel" (Must be same on all units)
- PL TX Power Level - If devices are close; a lower power setting can preserve battery life

4. Save Settings to the Xbee and Reload Firmware

a) Be sure to click the "Write" button in XCTU to save any settings that were made
b) Close XCTU
c) In Arduino Sketch IDE restore the following line to its original

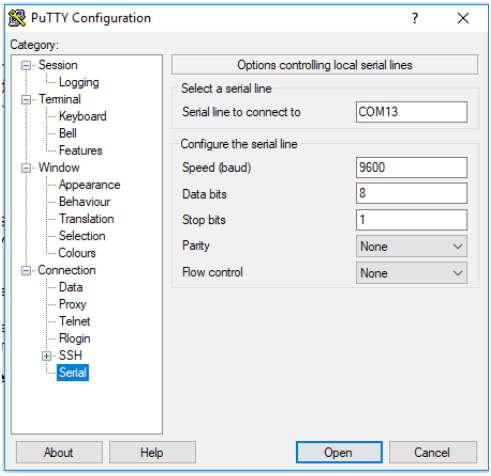
```
#define XBEECONFIG                      0          // 0 = RUN, 1 = Configure the XBEE using XCTU
```

d) Upload the firmware to the Unit as outline in Step #4 in 2.Uploading Firmware

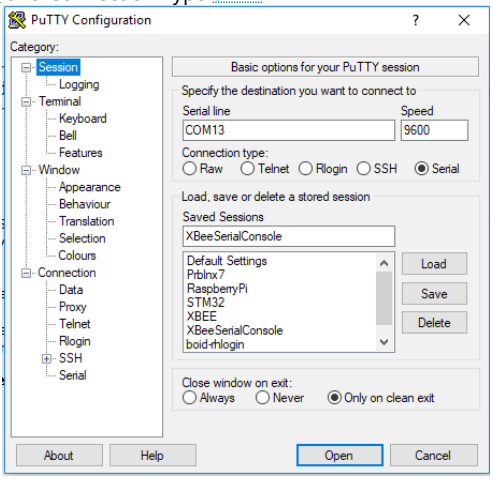
## 3.3 Configuring Xbee using Putty

1. Install Putty
   - Software can be gotten at https://putty.org/

2. Setup Putty
   - Under Category → Connection → Serial check the following settings




- Under SESSION enter the COMxx Port of the Hand-Remote or Pump-Controller, set Speed at 9600 and Connection Type SERIAL




- Then click OPEN to open a Terminal Window

3. Make Xbee Settings in the Terminal Window
   - Press '+++' without hitting enter. Xbee should respond with an 'OK'. You're now in 'AT' Command mode.
   - Type 'ATID' and hit enter. Xbee will respond with the current Xbee Network ID set on the Xbee Module ( i.e. 'AT' – 'ID' )
   - Xbee will escape its 'settings' mode quickly so if it was responding but isn't anymore; enter the '+++' to enter settings mode again.
   - Use command 'ATWR' to write the settings to the Xbee Module permanently
   - Example below shows how one would setup the Xbee with Network ID 1200
     - '+++' (no enter) Xbee Responds with 'OK'
     - ATID 1200 (enter) Xbee Responds with 'OK'
     - ATID (enter) Xbee Responds with '1200' the setting just made
     - ATAP 0 (enter) This sets Xbee in 'Transparent Mode'
     - ATWR (enter) This permanently stores the above settings

- For a full list of settings see the “Command Reference Table” in
  - Location https://www.digi.com/products/xbee-rf-solutions/sub-1-ghz-modules/xbee-pro-xsc#productsupport
  - In PDF https://www.digi.com/resources/documentation/digidocs/pdfs/90002173.pdf

# 4. Customizing Firmware

## 4.1 Pump-Controller

- ✔ The Pump-Controller Firmware preforms the following duties
  - ○ Wait for Hand-Remote Requests and generate responses
  - ○ Calculate values ( Virtual-Pin Values ) for special devices that require a process to obtain a value ( like ping/echo for Ultrasonic level measurements )
  - ○ Retain a Power-Off status after a power failure has occurred at the pump ( Arduino is designed to keep digital pins low after a power failure )

*NOTE: Wiring, Timing, and Almost all system functions are implemented on the HAND-REMOTE Firmware*

### 4.1.1 Virtual Pins

Virtual pins are non-hardware related value identifiers that are created and assigned by the PUMP-CONTROLLER firmware and then accessed by the HAND-REMOTE as a "Pin" value. An example case is the ultrasonic water level; The pump-controller firmware pings and times an echo of the ultrasonic distance meter using its D4 ( Trigger ) and D5 ( Echo ) pins and stores the measured value on "virtual pin 64". When the Hand-Remote queries for a value on pin-64 it actually gets the measured value which is a combination of D4 & D5 already calculated by the PUMP-CONTROLLER.

- ✔ An Example of a Virtual Pin setup on the PUMP-CONTROLLER firmware

```
void loop(){

  XBee.Available();


#if US_PRESENT>0
  // Read UltraSonic water level
  int ulCurrentTime = millis();
  if ( ulCurrentTime > ulLastPing + 1000 ) {
    XBee.VirtualPin(64, sonar.ping_in() );
    ulLastPing = ulCurrentTime;
  }
#endif

}
```

## 4.2 Hand-Remote

- ✔ The Hand-Remote Firmware preforms the following duties

  - ○ Storing the alarm values in non-volatile memory (EEPROM) so user values and settings aren't lost every-time the power is turned off
  - ○ Updating the current status (MAIN Values) of the Menu-Items and checking for Alarm boundaries.

- Processes User Input
  - UP/DOWN user buttons will increments or decrements the Menu-Items
  - RIGHT/LEFT user buttons will increments or decrements the Sub-Function of each Menu-Item
  - ENTER ( square ) user button will preform a value **SET**; or update MAIN values when they are being displayed

- Menu-Items are associated/connected to a **Device** and its **Hardware Pin**
  - Device - Example; The Hand-Remote running this firmware, or a Pump-Controller device controlling a pump
  - Pin - Example; 7(for D7) or A2 as identified inside the Pump-Controller Unit or by the Arduino UNO board

- Menu-Items have **Sub[?]** Values, such as
  - The actual/read/**MAIN** value of the Menu-Items - Example; Power is currently either ON/OFF ( current status )
  - **SET**ting a Value for the Menu-Item - Example; SET the Power to either ON/OFF
  - Setting a Too **LO**w **ALARM** Value - Example; When pressure is below 10 sound the Low Alarm
  - Setting a Too **HI**gh **ALARM** Value - Example; When pressure is above 100; sound the High Alarm

- Menu-Items can have Option Values ( *An Optional way of handling Menu-Item value assignments* )
  - Options associates a common MAIN-Value to a Text identifier; for example ON/OFF.
  - A Menu-Items value must be a select-able set of Options or a Number but cannot be both.
  - The maximum number of options is limited by program line `#define MAXOPTIONS 2 // Maximum number of Menu Item Options allowed`

- Menu-Item Limits
  - Program line `#define MAX_MENU_ITEMS 15 // Maximum number of Menu Items allowed ( using 71% dynamic memory )`
  - Determines the maximum number of menu-items that can exist; it can be increased slightly but may produce memory warnings.

### 4.2.1 Adding & Deleting Devices

✔ The Hand-Remote Firmware tracks ALL the associated devices it can control and monitor.

- To add another Device to the control and monitoring system
  - Add any Unique Name to the Comma separated list of devices on the Program Line shown below
    - `uDevices HandRemote, CanalPump, DitchPump; // Name and Define all controllable devices in the System`
  - At the top of the *`SetupMenu()`* function; Assign the Unique Named device a *`.Text`* Identifier ( Used in the Menu )
  - At the top of the *`SetupMenu()`* function; Assign the Unique Named unit Unit's *`.Transceiver_ID`* number assigned in the units firmware
    - Example Program Line `CanalPump.Text = "Canal Pump"; CanalPump.TransceiverID`

```
= 10;
```

- Delete devices by removing the items in the adding devices listed above.
  - Note: If there is no conflict of Transceiver_ID's or naming there really isn't a substantial need to remove a "missing" device from the system.

## 4.2.2 Creating a Menu-Item

1. Find the **SetupMenu()** function in the Hand-Remote firmware file; This is where all the Menu-Items are defined

2. Indexing - Determine where in the list of Menu-Items you'd like the new item to appear
   a) All Items MUST BEGIN with line `MenuItemsIdx++;` except the very first item ( Usually the Hand-Remote battery status )
   b) The line above increments the Indexing ( line count ) to allow the new menu-item to exist
   c) Items that do not have [MenuItemsIdx] indexing are used in other places of the firmware and should ***NEVER BE CHANGED!***

3. Device - Assign the device this Menu-Item is attached to
   a) This is the Device's Unique Name given in Step #A|Adding & Deleting Devices
   b) The unit is generally a named Pump-Controller device with an assigned Transceiver_ID.

4. Pin – Assign the Hardware Pin of the device that the Menu-Item will control or monitor
   a) The hardware pin is the terminal inside the controlling device box ( pump-controller ) that the control/monitor equipment is wired to
   b) NOTE: Digital Pins ( D4 → D7 ) are identified by only their number ( e.g. 4 = D4 ) whereas Analog pins require both letter and number ( e.g. A2 ).
      - Digital Example; `Menu[MenuItemsIdx].Pin=7;`
      - Analog Example; `Menu[MenuItemsIdx].Pin=A3;`

5. Text - Assign the Menu-Item a defining Text
   a) The text is what will be displayed on the Hand-Remote display
      - Example; `Menu[MenuItemsIdx].Text="Power";`

6. Options - Determine IF the Menu-Item will have textual "Options" or is just a numerical value
   a) If the Menu-Item will report numerical values such as Voltage, Water Level, Pressure etc... Skip to step #5
   b) Determine the Menu-Items Options and assign each a;
      - Text Example; `Menu[MenuItemsIdx].Option[0].Text = "On";`
      - Value Example; `Menu[MenuItemsIdx].Option[0].Value = HIGH;`

7. Sub[SET] – Determine IF the user should be allowed to set a value on the Location ( Output Pins like turning Power ON/OFF )

   a) If the Menu-Item is for monitoring status and the user will not be setting its value (INPUT)... Skip to step #7
   b) If the Menu-Item needs to allow the user the ability to set the value ( OUTPUT )
      - Set the "State" to SETTABLE to allow the user to SET the value
      - Example; `Menu[5].Sub[SET].State = SETTABLE;`

8. Sub[??ALARM] – Determine if the value should be monitored with an alarm
   a) LOALARM
      - If the Menu-items value is numeric and the value should be checked for getting too small
      - OR If the Menu-item value is an "option" and the value should be checked to see if it is EQUAL
        - Assign an LOALARM identifier Example; `Menu[5].Sub[LOALARM].ID = 'c';`

   b) HIALARM
      - If the Menu-items value is numeric and the value should be checked for getting too large
      - OR If the Menu-Item value is an "option" and the value should be checked to see if it is NOT-EQUAL
        - Assign a HIALARM identifier Example; `Menu[5].Sub[HIALARM].ID = 'C';`

9. Proceed to #4.3.2.3.Updating Changes|outline

### 4.2.3 Deleting a Menu-Item

1. Find the menu item you'd like to delete by identifying it by its' `Menu[MenuItemIdx].Text` setting
2. Select ALL the items from `MenuItemsIdx++;` up to but no including the next `MenuItemsIdx++;`
3. Press delete

#### A. Menu-Item Setup ( Example )

*The menu items are defined in the* **SetupMenu()** *Function identified by line 'void SetupMenu() {'. Each item in the Menu has a numeric index (i.e. Menu[index-#] below the constant 'MONITOR' is assigned index-0 and 'PUMPIDX' is assigned index-1).*

```
//=============================================================================================================
//---------------------------- MENU STRUCTURE ( ADVANCED CONFIGURATION ) ---------------------------
//=============================================================================================================
//*************************************************************************************************//
**
 * @brief  Setup the LCD menu
 * @remarks
 * - Allows a single spot customization to the user interface
 * - Display will show the items in the same order as they are defined here
 * @code
 *   exmaple code
 * @endcode
*************************************************************************************************/
uDevices HandRemote, CanalPump, DitchPump;        // Name and Define all controllable devices
void SetupMenu() {

  HandRemote.Text = "Hand Remote"; HandRemote.TransceiverID = TRANSCEIVER_ID;
  DitchPump.Text = "Ditch Pump"; DitchPump.TransceiverID = 10;
  CanalPump.Text = "Canal Pump"; CanalPump.TransceiverID = 11;

  //BATT (idx-0) ----------------------------
  Menu[BATT].Device = HandRemote;
  Menu[BATT].Pin = A1;                            // Battery level from the Hand-Remote pin A1
  Menu[BATT].Text = "Battery(B)";                 // Create a menu item for monitoring the Battery
  Menu[BATT].ValueModifier = BATTVOLTS;           // Modify raw value to show voltage
  Menu[BATT].Sub[LOALARM].ID = 'b';               // A Low Alarm is identified by a lower-case 'b'

  //----------------------------------------
  MenuItemsIdx++;
  idx = MenuItemsIdx;                             // !!!-- Set where the Menu will start --!!!
```

```
  Menu[MenuItemsIdx].Device = DitchPump;
  Menu[MenuItemsIdx].Pin = 7;                       // Power is on all Pump-Controller's on pin [D7]
  Menu[MenuItemsIdx].Text = "Power(P)";             // Create a menu item for Power Control
  Menu[MenuItemsIdx].Sub[SET].State = SETTABLE;     // Allow this Value to be 'SET' by the user
  Menu[MenuItemsIdx].Sub[LOALARM].ID = 'p';         // A Low Alarm is identified by a lower-case 'p'
  Menu[MenuItemsIdx].Sub[HIALARM].ID = 'P';         // A High Alarm is identified by an upper-case 'P'
  Menu[MenuItemsIdx].Option[0].Text = "Off";        // Power can be "Off"             - Option #0 = Off
  Menu[MenuItemsIdx].Option[0].Value = LOW;         // "Off" will be the value 'LOW"  - Off = LOW
  Menu[MenuItemsIdx].Option[1].Text = "On";         // Power can be "On"              - Option #1 = On
  Menu[MenuItemsIdx].Option[1].Value = HIGH;        // "On" will be the value 'HIGH'  - On = HIGH
  Menu[MenuItemsIdx].LastOptionIdx = 1;             // Last Option Index defined (Number of Options - 1)

  //----------------------------------------
#if ULTRASONIC_WATER_LEVEL_INSTALLED>0
  MenuItemsIdx++;
  Menu[MenuItemsIdx].Device = DitchPump;
  Menu[MenuItemsIdx].Pin = 64;                      // Water Level from VIRTUAL (Pump-Controller)
  Menu[MenuItemsIdx].Text = "Water (L)";            // Create a menu item for Water Level Transducer
  Menu[MenuItemsIdx].Sub[LOALARM].ID = 'l';         // A Low Alarm is identified by a lower-case 'l'
  Menu[MenuItemsIdx].Sub[HIALARM].ID = 'L';         // A High Alarm is identified by an upper-cse 'L'
#endif
  //----------------------------------------
#if WATER_PRESSURE_INSTALLED>0
  MenuItemsIdx++;
  Menu[MenuItemsIdx].Device = DitchPump;
  Menu[MenuItemsIdx].Pin = A3;                      // The 'signal' is on Pump-Controllers on pin [A3]
  Menu[MenuItemsIdx].Text = "Pressure(R)";          // Create a menu item for the Primary Pressure
  Menu[MenuItemsIdx].ValueModifier = PRESSURE;      // Modify value to display PSI instead of MPa
  Menu[MenuItemsIdx].Sub[LOALARM].ID='r';           // Low Pressure alarm is identified lower-case 'r'
  Menu[MenuItemsIdx].Sub[HIALARM].ID='R';           // High Pressure alarm is identified upper-case 'R'
#endif
/*// UN-COMMENT BELOW FOR A SECOND PUMP-CONTROLLER ( NAMED "Canal Pump" ) WITH TRANSCEIVER_ID = 11
  //----------------------------------------
  MenuItemsIdx++;
  Menu[MenuItemsIdx].Text = "Power(C)";             // Create a menu item for Power Control
  Menu[MenuItemsIdx].Device = CanalPump;
  Menu[MenuItemsIdx].Pin = 7;                       // Power is on all Pump-Controller's on pin [D7]
  Menu[MenuItemsIdx].Sub[SET].State = SETTABLE;     // Allow this Value to be 'SET' by the user
  Menu[MenuItemsIdx].Sub[LOALARM].ID = 'c';         // A Low Alarm is identified by a lower-case 'p'
  Menu[MenuItemsIdx].Sub[HIALARM].ID = 'C';         // A High Alarm is identified by an upper-case 'P'
  Menu[MenuItemsIdx].Option[0].Text = "Off";        // Power can be "Off"             - Option #0 = Off
  Menu[MenuItemsIdx].Option[0].Value = LOW;         // "Off" will be the value 'LOW"  - Off = LOW
  Menu[MenuItemsIdx].Option[1].Text = "On";         // Power can be "On"              - Option #1 = On
  Menu[MenuItemsIdx].Option[1].Value = HIGH;                // "On" will be the value 'HIGH'  - On =
HIGH
  Menu[MenuItemsIdx].LastOptionIdx = 1;             // Last Option Index defined (Number of Options - 1)

  //----------------------------------------
  MenuItemsIdx++;
  Menu[MenuItemsIdx].Text = "Pressure(F)";          // menu item for the Primary Pressure Transducer
  Menu[MenuItemsIdx].Device = CanalPump;
  Menu[MenuItemsIdx].Pin = A3;                      // 'signal' is gotten Pump-Controllers on pin [A3]
  Menu[MenuItemsIdx].ValueModifier = PRESSURE;      // Modify value to display PSI instead of MPa
  Menu[MenuItemsIdx].Sub[LOALARM].ID='f';           // Low Pressure alarm is lower-case 'r'
  Menu[MenuItemsIdx].Sub[HIALARM].ID='F';           // High Pressure alarm is upper-case 'R'

  //----------------------------------------
  MenuItemsIdx++;
  Menu[MenuItemsIdx].Text = "Pressure(S)";          // menu item for the Secondary Pressure Transducer
  Menu[MenuItemsIdx].Device = CanalPump;
  Menu[MenuItemsIdx].Pin = A4;                      // Menu-item is Pump-Option #0 (Canal) on Pin (A4)
  Menu[MenuItemsIdx].ValueModifier = PRESSURE;      // Modify value to display PSI instead of MPa
  Menu[MenuItemsIdx].Sub[LOALARM].ID='s';           // Low Pressure alarm is lower-case 's'
  Menu[MenuItemsIdx].Sub[HIALARM].ID='S';           // High Pressure alarm is upper-case 'S'
*/
  //------------[ Start-Up the Display ( DO NOT CHANGE! )]-------------
  for ( int i = 0; i <= MenuItemsIdx; i++ ) {
    if ( Menu[i].Sub[LOALARM].ID != NULL || Menu[i].Sub[HIALARM].ID != NULL ) {
      EEPROMGet(i);                                         // Load Alarm values from EEPROM
    }
  }
  GetItem();                                                // Get starting Menu item
  LCD_display();                                            // Update the display
}
```