

**(DM MM) Precision Time Keeping Project**  
**by**  
**DuinoMiteMegaAndy**  
**17/12/11 R1.b**

**Precision +/- 2 ppm, 3.3 VDC, TCXO, I2C DS3231 RTC for the DuinoMite Mega and other PIC32 boards.**

**Credits:** Thanks CrackerJack! <----<<<<

His "Analog Clock Application", on this forum, used the I2C Maxim-IC DS1307 RTC on his MaxiMite. He used this battery protected I2C DS1307 RTC to give his analog clock the proper time/date when the power was interrupted or when there was a manual reset.

**Project Goals:**

The question will come up on why I have spent so much time and effort to create this project and all the associated real time clock application and utility files for the DuinoMite / MaxiMite?

I wanted a MM/DM/MM basic microcontroller that would run autonomously, at a remote site, with NO Internet, NO PC, NO GPS or NO other external clock source timebase for eight years without any user supervision **even** for daylight saving times "hour" adjustments! Having pre-programmed projected DST times (Daylight Saving Times) for "spring forward and fall back" twice a year, I could run a precision TXCO DS3231 RTC on battery backup for years and never touch the time and date! Of course, after eight years, you need to replace the battery and correct the time which could be off by eight minutes max! <-----<<<<< WOW

Ever wonder why you lose your PIC32 time and date every time when there is a power interruption or manual reset on your DuinoMite or MaxiMite?

Ever tried to locate your DuinoMite, MaxiMite, MiniMite away from room ambient temperature and the time is way off? (Accurate logging times cannot be performed due to extreme or high/low temperature variations on the PIC32 watch crystal)

The answers to the above questions is because the on-board watch crystal for the PIC32 has no battery backup and the accuracy is only +/- 20 ppm (or worst) at room temperature. What is needed is a precision, +/- 2 ppm, battery protected, +3.3 VDC, TXCO, IC2, RTC. (3.3 VDC - so it can run with a Lipo power source on the DuinoMite Mega in extreme temperatures.) They should outlaw the use of these inaccurate watch crystals. There are better solutions for precision time keeping in the market place.

Maxim-IC has a precision 3.3V drop-in replacement for the 5 v DS1307 RTC (No software changes BUT NOT 100% register compatible) It has a +/- 2 PPM accuracy which is about 1 minute per year. This device is the Maxim-IC DS3231 RTC. (DS3231N is the high temp. version) Note: There are two DS3231 temperature versions and the one with the higher temp. range will have a higher PPM!

**Note1:** The DS1307 RTC is a +5 VDC low cost device with “watch crystal” and does not have a precision timebase TCXO like on the DS3231. This also applies to any of the other watch crystals installed on the DuinoMite, MaxiMite and MiniMite boards. The best accuracy watch crystal is 20 PPM (parts per million) which gives an error of greater than 600 seconds per year! Take any of these PIC32 boards, out of the ambient room temperature environment and you better lookout! The time base errors increases dramatically from the 600 seconds a year! Remote data logging would be troublesome using a watch crystal on the PIC32 or any other I2C RTC!

**Here is a description from Macetech on their ChronoDot DS3231 RTC breakout board..**

The ChronoDot RTC is an extremely accurate real time clock module, based on the DS3231SN temperature compensated RTC (TCXO). It includes a CR1632 battery which should last at least 8 years if the I2C interface is only used while the device has 5V(3.3) power available. No external crystal or tuning capacitors are required.

The DS3231 has an internal crystal and a switched bank of tuning capacitors. The temperature of the crystal is continuously monitored, and the capacitors are adjusted to maintain a stable frequency. Other RTC solutions may drift minutes per month, especially in extreme temperature ranges...the ChronoDot will drift less than a minute per year. This makes the ChronoDot very well suited for time critical applications that cannot be regularly synchronized to an external clock.

Since the ChronoDot needs to read the temperature, it also contains a fairly accurate temperature sensor, accessible through I2C. Stated accuracy is +/- 3C, though the temperature register returns values with a resolution of 0.25C. Calibration may improve accuracy.

### **DS3231 Breakout Vendors**

There are two vendors that sells this SMD DS3231 RTC, on a breakout board, and it is fairly **EXPENSIVE** but it works under extreme temperatures variations at +3.3 VDC. (Can be used with Lipo power and for data logging in extreme temperatures)

Link 1: (UK)

<https://www.loveelectronics.co.uk/products/137/ds3231-real-time-clock-module>

Link 2: (US) ChronoDot V2.1

[http://macetech.com/store/index.phpmain\\_page=product\\_info&products\\_id=8&zenid=03386458d60b62afd4d6c9f291d6dd10](http://macetech.com/store/index.phpmain_page=product_info&products_id=8&zenid=03386458d60b62afd4d6c9f291d6dd10)

Note: The ChronoDot has fixed breadboard pins which complicates wiring installation.

Manufacturer DS3231 Link:

<http://www.maxim-ic.com/datasheet/index.mvp/id/4627>

Link for a low cost +5 VDC DS1307 I2C real time clock:

[http://www.futurlec.com/Mini\\_DS1307.shtml](http://www.futurlec.com/Mini_DS1307.shtml)

Further thoughts ... Olimex should be able to produce a +-2 PPM DS3231 3.3 V RTC I2C UEXT board at a very low cost!

## RTC Software / Hardware Application Notes:

- Note 1:** Warning - You cannot mix and match +5 VDC and +3,3 VDC I2C devices on the same I2C bus. Either one or the other!
- Note 2:** The required I2C pullups on the DuinoMite Mega are already installed on board?. (4.7K pullup - on board)
- Note 3:** Again, running off of a Lipo battery on the Mega requires a +3.3 VDC I2C RTC device
- Note 4:** SDA1 and SCL1 I2C pins on the DuinoMite Mega are +5 VDC tolerant pins - only on the UEXT connector **NOT the A4 & A5** on the Arduino shield!
- Note 5:** DS3231 operates at 3.3 VDC and 5 VDC. The DS1307 cannot run at 3.3VDC?
- Note 6:** DS3231 clock frequency can be 100K or 400K (fast). The DS1307 is only 100K device.
- Note 7:** Version DS3231S - 0°C to +70°C +- 2 ppm (version. 1 ChronoDot fix battery soldered & Love Electronics with battery clip)
- Note 8:** Version DS3231N -40°C to 85°C +- 3.5 ppm (Version 2 ChronoDot–has battery clip)
- Note 9:** PIC32 RTC **does not** have a DOW – day of week. It can be calculated by my special MMBasic algorithm which uses today's "Date\$" in the calculation.  
Note: Last week, the firmware was changed to include the DOW on the PIC32.
- Note 10:** The DS3231 and DS1307 **does** have a DOW register and it should be set properly!  
DS3231/DS1307 RTC --> Sun(1)Mon(2)Tue(3)Wed(4)Thu(5)Fri(6)Sat(7)  
Both the PIC32 DOW and the DS3231/DS1307 will need to match if you are going to use the day of the week number (dow) or (dow\$) for control. The conversion is automatic my MMBasic software algorithms.

### PPM Ref. Parts per million (Maxim-IC)

- + - 2 PPM (Love,ChronoDot V1) 1.05 minutes per year
- + - 3.5 PPM (ChronoDot V2 with bat. Clip) 1.03 minutes per year
- + - 20 PPM (Standard watch crystal on MM/DM/MM, DS1307, PCF8563) 10.51 minutes per year!

### DuinoMite Mega DS3231 RTC Wiring Only

Just four wires and that's it.

DuinoMite Mega      Love / ChronoDot DS3231 Breakout Boards

I2C SCL1 A5 Note 4	-->	DS3231 I2C SCL
I2C SDA1 A4 Note 4	-->	DS3231 I2C SDA
+3.3 VDC	-->	DS3231 +3.3 VDC
GND	-->	DS3231 GND

- Note1:** Keep the interconnect wiring as short as possible to reduce wiring capacitance and noise.
- Note2:** The DuinoMite Mega has already 4.7 K I2C pullup resistors on-board.  
Do not install any pullups on the breakout boards.
- Note3:** The above DS3231 I2C RTC breakout boards need a Lithium coin cell battery for the RTC backup!
- Note4:** Caution ... 3.3VDC max. only on the Arduino shield pins A4 & A5 but on the SDA1 & SCL1 connections on the UEXT +5 VDC is allowed.

## MMBasic RTC Software Utility Files

**Note:** The MMBasic RTC code is "as is" and it could have bugs in it! Beta testing is on-going! (Put ALL the following files on drive B:(flash) and make sure the DS3231 I2C is wired and connected to the DuinoMite Mega at 3.3 VDC)

Using the standard MMBasic PIC32 RTC, a user has to solve several of the following real time clock problems using my time date software algorithms:

- #1. On reset or power-up the PIC32 clock loses its time and date.
- #1. Is solved by reading and updating the PIC32 RTC with the precision DS3231 RTC during boot-up using an algorithm in the "autorun.bas" file.
- #2. Removing the DuinoMite or MaxiMite from ambient room temperature will cause a large time deviation. (bad for data logging)
- #2. Is solved by reading and updating the PIC32 RTC with the precision DS3231 RTC every day using a periodic interrupt routine in the user "running" application.
- #3. On DST (Daylight savings time) users has to add one hour or subtract one hour to the time ("Spring forward and Fall Back")
- #3. Is solved by reading and updating the PIC32 RTC with the precision DS3231 RTC every day using a periodic interrupt routine in the user running application.  
My special algorithm computes the day of DST and adjusts the hour.
- #4. Remote time/date setup/configure and operation.
- #4. Is solved by just using the following basic utility files remotely (using the MMBasic commands at the command MMBasic prompt)

### Files:

<b>ADD1RTC.BAS</b>	Adds one hour to the DS3231 RTC (For manual DST adjust.)
<b>ARRTC_R1.BAS</b>	I2C read of DS3231 and updates PIC32 RTC.
<b>ASRTC_R1.BAS</b>	Reads PIC32 RTC and sets DS3231 RTC. (PIC32 setup/configuration time/date should be precise)
<b>AUTORUN.BAS</b>	Start-up and configuration file on reset or power on reset. Has an algorithm for reading the DS3231 for POR and resets. Boots to the Main_R1.bas user application or menu_1.bas User selectable by boot.dat file and menu_R1.bas menu selection.
<b>DST_R1.BAS</b>	Checks DST pre-programmed dates and adds/subtracts one hour to DS3231.
<b>Menu_R1.BAS</b>	Main Menu
<b>MSRTC.BAS</b>	Manual prompt driven DS3231 RTC set/configuration time/date routine.
<b>P_RTC.BAS</b>	Reads I2C DS3231 RTC and only prints the time and date.
<b>SUB1RTC.BAS</b>	Subtracts one hour from the DS3231 RTC (For manual DST adjust)
<b>TD_R1.BAS</b>	Time / Date Utility Menu
<b>Main.BAS</b>	Has a periodic interrupt at 1:59 AM to update the PIC32 every day and also checks for DST.
<b>Clock_P.BAS</b>	I2C read of DS3231 and prints both the PIC32 and DS3231 RTC time and dates. Used for drift RTC comparison in different ambient temperature locations. (For remote data logging)

**PIC32DOW.BAS**           Calculates DOW "day of week" from PIC32 RTC.  
Note: New firmware update has this DOW in it?

**SR\_SS\_R1.BAS**           Calculates sunrise and sunset times (to the minute) using  
Latitude, Longitude, Time zone, PIC32 RTC date.  
I chose an American city for this test algorithm.  
Credit: TassyJim <----- Thanks.

### Detailed Basic Application File Descriptions.

I modified the DS1307 files given to me by CrackerJack for his analog clock.  
I enhanced the I2C speed from 100K to 400K (high speed - 4x increase or 400%)  
Provided input error and range checking for user input to configuration the DS3231 RTC  
Time/DOW/Date. I made several enhanced RTC utility files for the DS3231 RTC local setup,  
configuration and reading and also for remote operation.

**File: ASRTC\_R1. bas** Used only as a utility to automatically set / configure the battery  
protected DS3231 RTC with the time/date/dow from the PIC32 RTC. (Time\$ and Date\$)  
First step is to update or make sure the PIC32 on-board RTC has the right time and date.  
Using MMIDE or Tera Term, at the command prompt, to set the PIC32 time and date from the  
PC is the easiest and accurate method.  
The other way is to manually set the Time\$ / Date\$ variable using MMBasic. You could also  
use the Windows time/date window for the best accurate time setting.  
After the time/date is set for the PIC32 RTC then running "Automatic Set Real Time Clock file"  
(ASRTC\_R1. bas) will move the PIC32 time/date to the precision DS3231 RTC. (Use only  
once) Another way is just re-boot the MM/DM and the autorun.bas will update the PIC32  
time/date. This task takes 10 secs.

**File: MSRTC\_R1. bas** Used only as a utility to manually set / configure the battery protected  
DS3231 RTC with the time and date.  
Using the command prompts, from this file "Manual Set Real Time Clock file (MSRTC\_R1.  
bas)", enter the time and date "ahead" and wait for the 0 second rollover before hitting the  
carriage return. You need to set the time ahead and wait for the zero rollover.  
You could use the Windows time/date window for the best accurate time setting.  
After manually setting the DS3231 RTC, You need to update the PIC32 RTC using the "run  
file" eg "Read RTC" (RRTC\_R1.bas) file. Another way is just re-boot the MM/DM and the  
autorun will update the PIC32 time/date.

**File: RRTC\_R1 . bas** Used to read the battery backup IC2 DS3231 RTC and to automatically  
update MaxiMite variables Time\$ and Date\$ for the on-board PIC32 RTC. I would suggest  
loading/running this file from the "AutoRun.bas" on drive A for power-up and re-booting.  
Then I would adjust DuinoMite Mega time every day around midnight using an interrupt  
periodic timer or the just the PIC32 clock itself. Of course, you could call this "self run file"  
anytime to make sure the on-board PIC32 RTC is accurate BUT beware of DST! (Daylight  
savings time). You need to compensate the "spring forward and fallback" time adjustments for  
DST twice a year.

**File: PIC32DOW.bas** Calculates the DOW number and string from today's date on the PIC32  
RTC. The PIC32 now does have a "day of week" variable but the DS3231/DS1307  
RTCs also have DOW register.

I created a utility file that does the daylight savings time hour adjustment automatically under a MMBasic periodic interrupt. The projected dates of DST are pre-programmed into the software. Check the **DST\_R1.bas** file to make sure the projected dates are correct for your location and how your government institutes DST! In the US, it is the second Sunday in March and the first Sunday in November at 02:00 AM. (Spring forward +1 Hr and Fall back -1 hour.) (Different country governments have different DST!)

**Note:** The DST\_R1 is only a test file. This algorithm is embedded in the main.bas running demo code.

**\*.Dat FILES** There is no EEPROM on the PIC32, so I used Dat Files to store strings and variables.

**Boot.Dat** Allows bootup to the Main\_R1.bas user application or the Menu\_R1.bas. The Menu\_R1.bas toggles the string text in this file by menu selection. (Default = Main)

**DST.Dat** and **Lockout.DAT** are two files used by the DST basic file algorithm. These files prevent multiple RTC DST updates on the same day of DST and they clear on the next day when there is no DST requirement.

### **PIC32 and DS3231 RTC Demo Code Application**

I tried to create a modular, transparent, easily to use, RTC time and date utilities. Each \*.bas file only does one function like reading from a RTC or writing to a RTC, so other users can easily implement a specific function needed without any "major" modifications. Since the PIC32 does not have EEPROM, I had to use \*.dat files to store and read program flags and variables. By transparent, I mean keeping the PIC32 RTC refreshed by the DS3231 precision RTC is almost invisible. On boot-up, due to a power on reset or manual reset the PIC32 is updated using the "autorun.bas". In the main application "main.bas" a periodic interrupt fires at 1:59 AM every day for refreshing the PIC32 and checking for DST. The size of this code can be easily reduced and polished but what is 56K bytes out of 16 GB of SD flash?

Don't get confused on all the files for the time/date utilities. You only need one file to set the DS3231 RTC and another file to read the DS3231 and refresh the PIC32 RTC. All the other files are used for testing and advanced time keeping algorithms and also provide alternative methods for reading and adjusting the time and date.

Just unzip the attached file and put ALL files on drive B: (Flash)  
Make sure the RTC is hooked up (4 wires) and you are good to go!

On boot-up the DM/MM will enter the main application and loop forever.

Just do a control C

Type in run "b:menu\_R1.bas" or run "b:TD\_R1.bas" will get you to the menu time/date utilities.

```

120 PRINT "*****"
130 PRINT "*"      Main Menu
140 PRINT "*****"
150 PRINT "*" [A] <-- Print PIC32 Time Date
160 PRINT "*" [B] <-- Spare
170 PRINT "*" [C] <-- Spare
180 PRINT "*" [D] <-- Spare
190 PRINT "*" [E] <-- Mario_R1.bas Sound Test
200 PRINT "*" [F] <-- Boot To Main_R1.bas
210 PRINT "*" [G] <-- Boot To Menu_R1.bas
220 PRINT "*" [H] <-- Run Main_R1 Application
230 PRINT "*" [I] <-- Time/Date Utilities
240 PRINT "*" [Q]<-- Quit Menu
250 PRINT "*****"

```

Enter "I" selection to get into the time / date utilities.

```

120 PRINT "*****"
130 PRINT "*" PIC32 DS3231 Time / Date Utilities
140 PRINT "*****"
150 PRINT "*" [A] <-- Print PIC32 Time/Date
160 PRINT "*" [B] <-- Print DS3231 Time/Date/DOW
170 PRINT "*" [C] <-- PIC32 RTC <--<<< DS3231 RTC
180 PRINT "*" [D] <-- DST Daylight Savings
190 PRINT "*" [E] <-- Spring FWD +1 Hr. DS3231 RTC
200 PRINT "*" [F] <-- Fall Back -1 Hr. DS3231 RTC
210 PRINT "*" [G] <-- DS3231 RTC <--<<< PIC32 RTC
220 PRINT "*" [H] <-- Manual Set/Config. DS3231 RTC
230 PRINT "*" [I] <-- Print/Compare PIC32 DS3231 RTC
240 PRINT "*" [J] <-- Print PIC32 DOW - Day of Week
250 PRINT "*" [K] <-- Print PIC32 DOY - Day of Year
260 PRINT "*" [L] <-- Print Sunrise / Sunset times
270 PRINT "*" [M]<-- Go To Main Menu
280 PRINT "*" [Q]<-- Quit Menu
290 PRINT "*****"

```

I used the MMIDE to set the time/date on the PIC32 and then use Menu [G] <-- DS3231 RTC <--<<< PIC32 RTC to set the precision DS3231 RTC from the PIC32. In 10 seconds you are finished!

You could also manually set the DS3231 by using Menu [H] Manual Set/Config. DS3231 RTC Reset or reboot the MM/DM to refresh the PIC32 RTC. The autorun.bas file calls Menu [C]

## WARNINGS:

There could be a “setup” utility for the DuinoMite Mega that changes the Day-Month-Year “European date format” to the “American Month-Day-Year format”. This utility makes it simple to allows different Olimex UEXT RTC boards to work with the DuinoMite. **This might throw a monkey wrench** in all of my time/date algorithms. I prefer to keep the European date standard! If you change to the US date format be prepared to change a lot of software. Just keep the setup utility in the European date format!

All time keeping is done in Military Time for easy programming convenience.  
(13:00 = 1:00 PM) If you chose otherwise, plan to spend hours making it work in civilian time!  
Note: The MM/DM uses military time!

The question will come up on how did CrackerJack make his +5 VDC DS1307 RTC work on his MaxiMite? Since the DuinoMite & MaxiMite can handle both +5 VDC devices and +3.3 VDC devices and the I2C bus on-board pullups, on the DuinoMite, is tied to 3.3 VDC. Then since the DS1307 is a +5 VDC ONLY device and it cannot run at 3.3 VDC (per specs. but it is possible?), then the only other possible explanation is his MaxiMite had external +5 VDC I2C bus pullups pins. The I2C bus pins on the DuinoMite I2C bus is +5 VDC tolerant BUT keep in mind that the DuinoMite I2C bus pins pullups are on-board and are tied to 3.3 VDC! To make this very low cost DS1307 RTC work on the DuinoMite Mega, you need to hack it. Another **warning** - If you use a +5 VDC RTC then switching to the DM Mega Lipo battery source, your +5 VDC will be gone! The best possible solution is to stick with a 3.3 VDC RTC, to be safe.

**Pictures:**

ChronoDot V2 (US) (DS3231N) +- 3.5 PPM

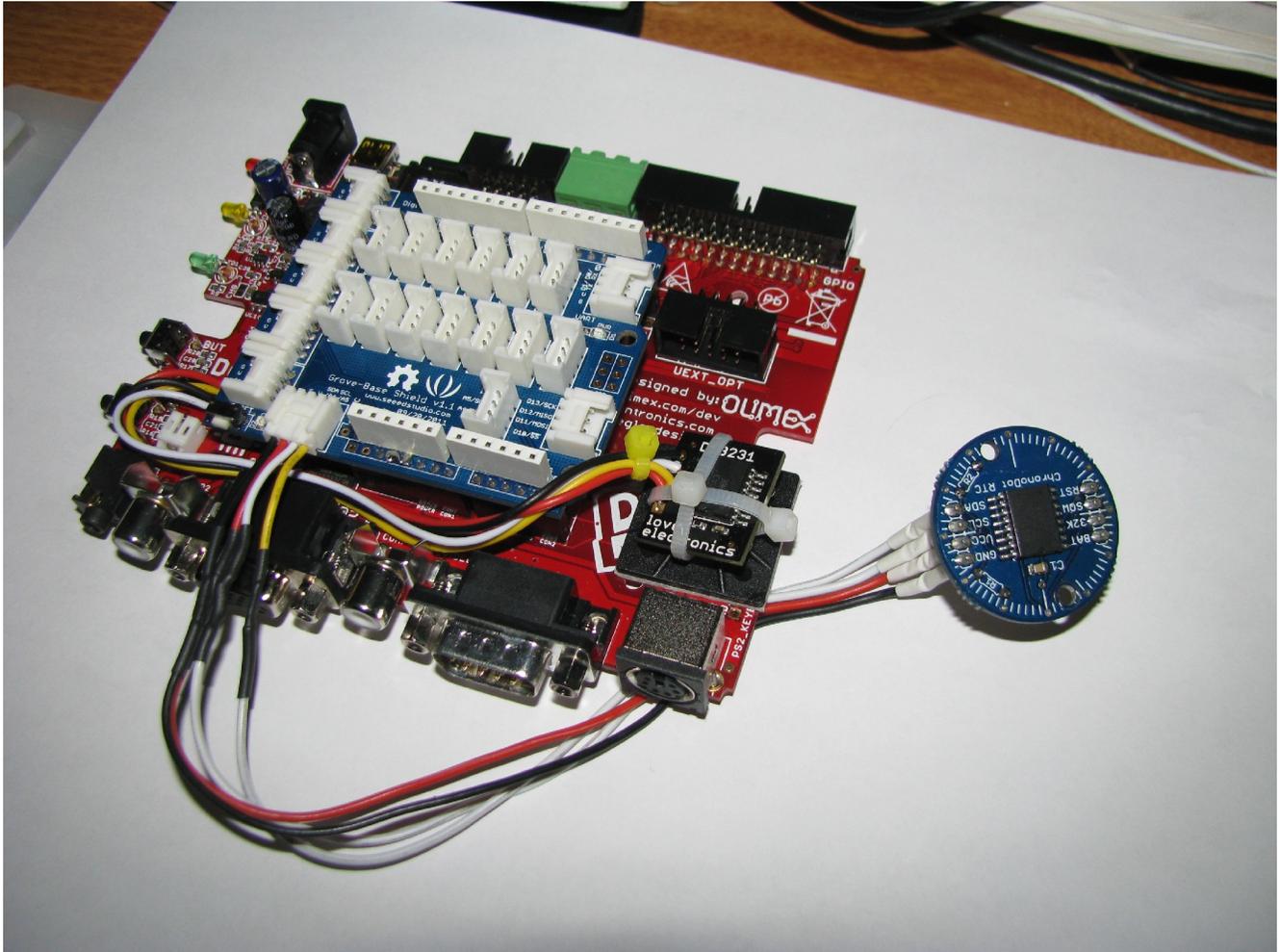


Love Electronics (UK) (DS3231S) +- 2 ppm (I used this device in testing)





Picture of DuinoMite Mega with the two thoroughly tested DS3231 real time clocks.



Disclaimer: This document and demo attached basic files are "as is".  
No warranty or guarantee implied whatsoever. In fact, this document and basic code could have bugs and errors in it. If you use this code and your house burns down, your wife and children become homeless and worst of all "the dog" dies, please do not hold me responsible!