ARDUINO Microcontroller

REM_SYS – Asynchronous Elapsed & Real-time Reminder Alerting, v 2.00

User Guide

Contents

Glossary of Terms

Term	Meaning
alert	The triggering of an ETR/RTR when its respective time has elapsed/is due. Alerts are presented to end user code via the scan_RQ() function call on a FIFO basis. That is, the oldest in the queue is presented first.
application /solution	The ARDUINO development environment uses the term 'sketch' to refer to the source C++ code comprising a program. In this guide, rather than 'sketch', the terms 'solution' and 'application' are used as these terms provide a higher level understanding of the intended outcome of an ARDUINO sketch.
end user	In the context of this guide, the term 'end user' refers specifically to designers and developers of ARDUINO solutions and applications.
ET	Elapsed time.
ETR	Elapsed Time Reminder
FIFO	A First-In, First-Out queue.
github	An international repository for developers to lodge and distribute their source code. The REM_SYS source code and associated documentation is held by a github repository.
OOTB	Out of the box.
REM_SYS	Reminder System – a framework for the development of time based Arduino applications.
reminder	The definition (creation) of an entity with time and end user attributes.
reminder type	ETRs and RTRs each support three different reminder types. See Appendix A, Reminder Types.
RQ	Reminder Queue.
RT	Real-time.
RTR	Real-Time Reminder.
subtype	A user defined value used as a part of the creation of ETRs/RTRs and which is returned following a reminder alert for end user processing code.
tab	An Arduino IDE tab containing comments and code.
tag	The name given to a `#define` declaration, for example the tag associated with `#define yes 1`, is "yes", its value is 1.

Introduction

This guide provides a quick and ready reference to understand and implement the REM_SYS framework for designing and building Arduino microcontroller applications based on the concept of asynchronous elapsed and real-time reminder alerting.

That is, if you have a need for date/time based processing, either as elapsed time or real-world real-time, then REM_SYS is well worth a look. REM_SYS can provide a different approach and thinking about application design and is well suited to multi-tasking solutions.

The framework has been derived from an earlier version[1] that dealt exclusively with elapsed time reminders (ETRs). In this version, real-time reminder functionality has been added to the earlier ETR framework, providing a comprehensive suite of capabilities for supporting applications that require asynchronous alerting over periods of time (elapsed and real-time). However, in this version, it is possible to configure REM_SYS with or without the real-time functionality. This is achieved by the setting of a configuration parameter to either 'true' (an RTC is connected) or 'false' (an RTC is not connected) (see below).

The REM_SYS framework is built around the concept of defining reminder entities (which can be either of type elapsed or real-time) which are processed and handled asynchronously, leaving main line code to do ..... whatever it needs to do.

Reminder alerts (ETRs/RTRs) can be considered to be analogous to interrupts, albeit they are produced at defined points in time. Reminder alerts will be processed as a part of the main code loop and can provide comprehensive features that offer the designer a good deal of freedom in decision support and control. The beauty of the framework is that it supports multiple ET and RT reminders concurrently, up to the number defined as part of framework configuration.

In producing this guide, an attempt to balance brevity and the degree of technical detail required to successfully implement the REM_SYS framework has been sought. Hopefully, the reader will find this balance struck?

What Type Of Reminder Alerting Does REM_SYS Support?

REM_SYS supports two 'flavours' of timed reminder alerting - Elapsed Time Reminders (ETRs) and Real Time Reminders (RTRs). Each of these two 'flavours' can be further divided by functionality, by a reminder type ID which determines the specific function. In summary, these are:

Reminder Entity 'flavour'	Reminder Type ID	Functionality
ETR	1	Elapsed Time, One Off Alerting: Will raise a once only reminder alert after the given start time has elapsed. The clock will start at the time the reminder is created.
ETR	2	Elapsed Time, Recurring Alerting: Will raise a recurring reminder alert after the given start time has elapsed and then every specified frequency, indefinitely. The clock will start at the time the reminder is created.
ETR	3	Elapsed Time, Recurring Alerting with Duration Alerting: Will raise a recurring reminder alert after the given start time has elapsed and then every specified frequency until the specified elapsed time has been reached. The clock will start at the time the reminder is created.
RTR	4	Real-time, One Off Alerting: Will raise a once only reminder alert at the specified real-time. The

[1] See User Guide for ....

Reminder Entity 'flavour'	Reminder Type ID	Functionality
		associated RTC is used for RTR timing.
RTR	5	Real-time, Recurring Alerting: Will raise a recurring reminder alert at the specified real-time and then every specified frequency, indefinitely. The associated RTC is used for RTR timing.
RTR	6	Real-time, Recurring Alerting with Duration Alerting: Will raise a recurring reminder alert at the given start time and then every specified frequency until the specified elapsed time has been reached. The associated RTC is used for RTR timing.

All reminder types are fully programmatic. That is, the designer has complete control in reminder definition/declaration and post-alert actions. Whether you want reminder alerting every 1/10th second, every hour, at specific times of the day, or any other period of time, then REM_SYS can support this. The reminder period for alerting for both elapsed and real-time reminders is fully flexible with unlimited choice of alerting intervals.

Reminders are created by the use of specific function calls, with suitably crafted parameters that represent the reminder type, start time, frequency, duration and user defined data. The parameter list for ETRs and RTRs can appear to be a little foreboding, but once the declaration structure is appreciated, ETRs and RTRs can be defined in a very logical manner and with ease.

ETR and RTR function parameters largely follow the same meaning. The differences only relate to start time:

1. ETR time parameters for start time, frequency and duration are each defined with four values – hour, minute, second and subsecond
2. RTR time parameters are each defined with three values - hour, minute and second. Note, RTRs do not support subsecond alerting
3. All time parameters for ETRs are elapsed time values – ETRs do not know about real world time
4. The start time parameters for RTRs represent real world time. That is they are defined as real-time values, 24 hour clock notation. RTR frequency and duration parameters represent elapsed times, just like ETRs.

In summary, the time parameter differences are:

Time Parameters	ETRs		RTRs	
	Elapsed Time	Real Time	Elapsed Time	Real Time
Start time	Yes	No	No	Yes
Frequency	Yes	No	Yes	No
Duration	Yes	No	Yes	No

All other parameters of the ETR and RTR functions are defined identically and have the same meaning and purpose.

Once declared/defined, the processing of reminder alerts occurs in the main code loop by use of a special function call (scan_RQ()). The main segment of code, OOTB, is configured with a suitable structure to process reminder alerts plus any other end use code needs.

How Do I Set up REM_SYS?

Software and Hardware Requirements

REM_SYS has been exclusively designed, developed and tested under Windows 10 and Arduino IDE v1.8.12 using an Elegoo Mega 2560 R3 16 MHz microcontroller and an Elegoo DS1307 Real Time Clock (RTC). The wiring plan for the microcontroller and RTC can be seen at Appendix C.

Standard Arduino libraries and functions are referenced by default by the framework. The only additional library included is the RTC library <RTClib.h>. This is declared within the tab named "D15_DateTime_Segment". The library should be added to the IDE environment via the 'Tools/Manage Libraries' option.

To assist the developer, a catalogue of the most useful REM_SYS functions and data items are described at Appendix A, Summary of User Functions & Declarations.

Source Code and Documentation

Source code and other documentation can be located on github as below:

Primary documentation location:

https://github.com/ronbentley1/Arduino-REM_SYS-Asynchronous-Timed-Reminders

Source code location:

https://github.com/ronbentley1/Arduino-REM_SYS-Asynchronous-Timed-Reminders/tree/Code-Branch

Start by initially down loading the download instructions from the primary documentation location (see above) and then follow these. This will ensure that you obtain the latest versions.

Configuring REM_SYS

Once downloaded and installed compiled and uploaded to the microcontroller then OOTB and with no further configuration, REM_SYS will:

- assume that an RTC is available and connected as per wiring plan
- open a serial port ready for diagnostic output (data rate of 115200 baud), and
- support up to 10 concurrent ETRs/RTRs, and
- create a free chain structure of data blocks to support the RQ (FIFO) suitable for 16 concurrently triggered reminder alerts, ahead of asynchronous main line code processing by the scan_RQ() function, and
- select timer0 as the source timer for ETR processing, and
- set the timer drift parameter for ETR processing to 1 second per hour, and
- configure an ETR reminder list scan rate of 100 milliseconds, or 10 Hz.
- have a total footprint of 9,900 bytes (8,854 code and 1,046 global data)

In addition, of the 10 ETR/RTR 'slots' configured in the reminder list, if an RTC is configured then two are pre-allocated and predefined - one to an ETR and one a RTR. If an RTC is not configured then only an ETR is defined, leaving nine reminder slots free. These are configured as follows:

1. A predefined ETR is configured and is used to drive a heart beat monitor that will flash the onboard microcontroller LED on pin 13 at either 1 Hz or 2[2] Hz. This is used as a visual indicator to show that the framework is operating.

[2] A flash frequency of 2 Hz is configured only if the `ETR_R_list_scan_freq` is configured for 1000 Hz, i.e. one scan per second of the remind list.

2. A predefined RTR is configured to alert each midnight (00:00:00 hours) to carry out any daily housekeeping the end user designer may wish. By default, at each midnight, this RTR increments the day number of today's date (the new day) and the day of the week index value for the new day. These two system wide readily available values can be used, as required, when manipulating dates or in decision control/flow without the need for them to be further calculated.

These OOTB configurations are shown below and can be found in the REM_SYS tab titled "C00_Configurations":

```
//  ***************************************************************
//  USER CONFIGURABLE PARAMETERS ARE REFERENCED HERE
//
#define diags_on      true

#define RTC_enabled   true  //Real-time clock enabled if
                              true, false, if not

int ETR_timer_number            = 0;  // 0 for timer0, 2 for timer2

#define ETR_R_list_scan_freq    100   //time (msecs) between
                                      //scans of the ETR list
//
//  The timer_drift_adjustment variable allows the inaccuracy
//  of timer0/2 to be compensated for, but only as far as the
//  drift per hour in seconds. This is ONLY relevant for ETRs
//
long signed int timer_drift_adjustment[3] = 1; //number of seconds
                                               //(+/-) per hour to adjust
#define max_RQ_free_chain_blocks 16 //size of the RQ free chain in blocks

#define max_R_list_entries       10 //num of reminder list entries
//
// END OF USER CONFIGURABLE DATA/PARAMETERS.
// ***************************************************************
//
```

Depending on needs, the above configurations may be adequate for most purposes. However, if not, then make changes as required, but take care to ensure that queue sizing is large enough to accommodate alerts if the ETR scan rate is set high (i.e. > 10 Hz) and many ETRs are defined concurrently with rapid alerting frequencies.

End User Code

End user code should be added where indicated and needed:

- within the setup() function in the set up tab titled "H00_Setup", and
- within the main segment titled "M00_Main_Segment", and/or

[3] It was observed that, during testing, the microcontroller timer0/2 were not accurate. For the microcontroller used, the timers showed a drift of 1 second per hour – they were running fast by this amount. Therefore, to compensate, an adjustment parameter has been built into the ETR create function such that it is possible to accommodate any drift in timing, either + (fast running) or – (slow running). The variable '`timer_drift_adjustment`' is used for this purpose. OOTB it is set to 1 to compensate for the fast running by 1 second per hour. However, if it is found that the microcontroller that REM_SYS is implemented on differs then set this value according, either +/- number of seconds per hour to 'trim' the configured timer back into accuracy.

- additional tabs (and code), suitably named, so that placement ensures that global variables that may be referenced will be in scope

Servicing Reminder Alerts

The main segment structure (M00_Main_Segment tab) is designed so that reminder alerts are handled via repeated cycling calls to the scan_RQ() function and, within this code block, by switch case statements[4]. Conversely, non reminder alerting code can be inserted in the associated 'else' code block.

Reminder alerts will stay in the RQ until they are serviced via the scan_RQ() function, so it is necessary for end user code design to ensure that the RQ is regularly and routinely serviced and any alerts returned processed. As indicated above, the OOTB framework already contains a structure design for this to be accommodated.

How Do I Create & Use Reminders

The REM_SYS framework provides an efficient and effective method for supporting time based solutions - time based activities are driven by reminder definitions and associated alerting, all within an asynchronous structure.

Reminders are created via the use of two specific functions - create_ET_reminder, and create_RT_reminder. As their name suggests, these functions handle elapsed time and real-time reminder creation, respectively.

To start, it is essential to understand the similarities and differences between ETRs and RTRs. Before continuing, it is important to have read and digested:

- What Type Of Reminder Alerting Does REM_SYS Support?, and
- Appendix B, A Note About Time (hmss) Parameters

The two functions, create_ET_reminder and create_RT_reminder, are used respectively, to create elapsed time and real-time reminders. Their respective parameters lists work largely in the same way. The two differences are:

1. RTRs can be defined to the second and NOT to subsecond, like ETRs, and
2. RTR start time is defined as a real world real-time, whereas ETR start time is specified as an elapsed time - ETRs know nothing about real-time.

All other parameters behave identically between the two functions.

Whilst the parameter list of the ETR/RTR functions may at first seem daunting, with a little use, specifying them will soon become second nature. To assist in design and documentation, Appendix D, Timed Reminder Definition Table can be used to record all ETR/RTR definitions and specifications. This will help to keep a ready record of what is defined.

Once the requirement for ETRs/RTRs is known, understood and documented, then creation is straight forward.

Each ETR/RTR should be considered in at least two parts:

1. Initial definition and creation. This would typically be done within the setup() function to 'start the process', but not exclusively (see example below), and

[4] During testing it was observed that, in some circumstances, main segment switch/case processing did not always perform as expected – sometimes with defined code and processes being ignored. A quick trawl of the internet does suggest that this is a potential issue with the IDE compiler. So if, your code does not do what you think it should within the switch/case structure, consider this observation.

2. Post alerting processing. The scan_RQ() function within the main segment will obtained ETR/RTR alerts as they become due. The OOTB framework provides a switch/case structure, switched by reminder subtype, to handle associated ETR/RTR processing needs.

Of all of the user provided parameters, it is subtype that would normally, but not exclusively, be used to define the purpose for which an ETR/RTR has been created. In conjunction with reminder type, subtype is a very convenient way to provide decision support. Other user provided parameters can be similarly used (i.e. user1-4 parameters).

The above describes a typical approach to establishing ETRs/RTRs perhaps sufficient for 80% of most design objectives. However, ETRs/RTRs can be created anywhere and at any time. Consider the following requirement:

> *"A daily reminder alert is required every day at 06:00:00 hours and from this time, a certain process is to be run every 15 seconds for one hour."*

In this example, an approach would be:

1. Create a type 5 (RTR, recurring indefinitely) reminder within the setup() function to start at 06:00:00 and repeat every 24 hours, say with a subtype of 99 to differentiate this RTR from ny other (it can be any value, 0-255)
2. Craft and insert code within the scan_RQ()/switch/case segment that handles the subtype 99 alert in which a ETR type 3 (recurring with duration) reminder is created that will alert immediately, then every 15 seconds and for one hour. Say the subtype is 199
3. Craft and insert code within the scan_RQ/switch/case segment to handle the subtype 199 alert
4. Note that the last alert can be tested for with R_status = final_alert.

The above approach will meet the above requirement with, at most, two concurrent reminders – one RTR (a standing reminder) and one ETR (a temporary reminder). The steps would be:

Step 1 - setup() definition, include a RTR definition:

```
create_RT_reminder(5, 99, // this subtype will create a temp ETR
 6,0,0,    // start at next 06:00:00 hours
24,0,0,    // repeat alert every 24 hours
 0,0,0,    // duration n/a
 0,0,0,0); // no user params needed
```

Step 2 - main segment tab, subtype 99 processing:

```
create_ET_reminder(5, 199,
 0,0,0,0,  // start immediately
 0,0,15,0, // repeat alert every 15 secs
 1,0,0,0,  // stop after 1 hour
 0,0,0,0); // no user params needed
```

Step 3 – main segment subtype 199 handling:

Create a switch case label of 199 within the scan_RQ() block switch/case block and add the code needed to handling the alert events that will be generated as a result of the step 2.

With care and planning, the possibilities for timer based processing are endless. Further detail regarding the ETR/RTR creation functions can be found in the appendices along with other associated and helpful functions.

Don't Delay

Who doesn't use delay()? It can be a very convenient and useful function to stop and wait for a while. And ... therein lies its problem - whilst it is waiting for the specified time to elapse then nothing else can continue.

REM_SYS relies on two interrupt based timers to provide the underling timing sources for the scanning of ETRs/RTRs and raising of alerts. ETR processing is provided by timer0 and RTR processing by timer1. If implementing REM_SYS OOTB, then the end user will find that the delay() function will not work. This is because it relies on timer0 for its timing source.

Whilst the use of delay() is discouraged, if it is absolutely necessary for delay() to be used, then REM_SYS can be configured to allow this. But be aware that use of delay() will prevent the ETR/RTR scanners from running, halted until the delay() is fully completed. The upshot is that:

- ETR alerts will be delayed for the duration of any delay() call, and
- RTR alerts may be missed, because the real-time trigger time has passed

To configure an alternative timer to timer0 then alter the value of the configuration variable 'int ETR_timer_number' from 0 to 2. This can be found in the C00_Configurations tab. Setting this parameter to 2 will select timer2 to be the source timer for ETR processing. However, be aware of the consequences! Note that timer1 is not reconfigurable.

It may be possible to avoid use of the delay() function by designing code in two parts – pre-delay() processing and post-delay() processing.

The end of any pre-delay() processing can be completed by the creation of an ETR type 1 (one off) reminder, with a suitable subtype and with the required start time delay. However, note the limitations on timing (see Appendix B, A Note About Time (hmss) Parameters).

The post-delay() process can then be defined under the scan_RQ/switch/case code block, switched by subtype.

This is not a perfect solution but is one that may be helpful and keep timing true to source.

Appendix A

Summary of User Functions & Declarations

ETR/RTR Functions

This appendix describes the principal functions associated with ETRs and RTRs.

<table>
<tr><td>Function name:</td><td colspan="3">create_ET_reminder</td></tr>
<tr><td>Function type:</td><td colspan="3">int</td></tr>
<tr><td>Function parameters:</td><td colspan="3">int r_type,
int r_subtype,

LSI start_hrs, LSI start_mins,
LSI start_secs,LSI start_ssecs,

LSI freq_hrs, LSI freq_mins,
LSI freq_secs,LSI freq_ssecs,

LSI duration_hrs, LSI duration_mins,
LSI duration_secs, LSI duration_ssecs,

int user1, int user2, int user3, int user4</td></tr>
<tr><td>Purpose:</td><td colspan="3">To create a real-time reminder (RTR)</td></tr>
<tr><td>Return Values:</td><td>#define ‘tags’</td><td>Value</td><td>Notes</td></tr>
<tr><td></td><td>invalid_R_type</td><td>-1</td><td rowspan="12">#define tag names are self explanatory</td></tr>
<tr><td></td><td>invalid_R_subtype</td><td>-2</td></tr>
<tr><td></td><td>invalid_R_start_in</td><td>-3</td></tr>
<tr><td></td><td>invalid_start_in_mins</td><td>-4</td></tr>
<tr><td></td><td>invalid_start_in_secs</td><td>-5</td></tr>
<tr><td></td><td>invalid_start_in_subsecs</td><td>-6</td></tr>
<tr><td></td><td>invalid_duration_mins</td><td>-7</td></tr>
<tr><td></td><td>invalid_duration_secs</td><td>-8</td></tr>
<tr><td></td><td>invalid_duration_subsecs</td><td>-9</td></tr>
<tr><td></td><td>invalid_freq_mins</td><td>-10</td></tr>
<tr><td></td><td>invalid_freq_secs</td><td>-11</td></tr>
<tr><td></td><td>invalid_freq_subsecs</td><td>-12</td></tr>
<tr><td></td><td>invalid_freq</td><td>-13</td></tr>
<tr><td></td><td>reminder_list_full</td><td>-99</td><td>No further free space in the RQ – too many ETR/RTR definitions active</td></tr>
<tr><td>Notes:</td><td colspan="3">1. r_type, r_subtype, user1, user2, user 3 and user4 are all of type int. All other parameters are of type LSI - long signed int
2. r_type defines the type of the ETR - 1, 2 or 3, see table below.
3. r_subtype is a user definable value in the range 0 to 255
4. start_hrs/ start_mins/ start_secs/ start_ssecs defines the time to elapse before the first reminder alert is generated. The elapse time countdown will start from the moment that the ETR is created
5. freq_hrs/ freq_mins/ freq_secs/ freq_ssecs defines the frequency that reminder alerts will be generated after the first (defined by start_hrs/ start_mins/ start_secs/ start_ssecs)
6. duration_hrs/ duration_mins/ duration_secs/ duration_ssecs defines the elapsed duration that reminder alerts will be</td></tr>
</table>

<table>
<tr><td>Function name:</td><td colspan="2">create_ET_reminder</td></tr>
<tr><td></td><td colspan="2">generated for, from the initial start time. A duration of 0:0:0:0 is permissible, see below.</td></tr>
<tr><td></td><td colspan="2">Permissible RTR Types</td></tr>
<tr><td></td><td>RTR Types</td><td>Meaning</td></tr>
<tr><td></td><td>1</td><td>Type 1 defines a one off ETR. Once the alert is raised the ETR is deleted and no other alerts will be raised</td></tr>
<tr><td></td><td>2</td><td>Type 2 defines a recurring ETR without end, unless programmatically deleted (see delete_reminder())</td></tr>
<tr><td></td><td>3</td><td>Type 3 defines a recurring ETR with a fixed duration. Reminder alerts will be generated at each frequency. Once the duration has elapsed the ETR is deleted.

Note: for reminder type 3 only if frequency is set to 0:0:0:0 (i.e. no frequency) then the reminder will finish once the duration time is reached.</td></tr>
<tr><td colspan="3">Examples</td></tr>
<tr><td colspan="3">Example 1 - one off ET reminder:


```
int result;
result = create_ET_reminder(1,7,
         0, 1, 0,0,   // start in 1 min
         0, 0, 0,0,   // n/a for type 1 rems
         0, 0, 0,0,   // n/a for type 1 rems
         5, 0, 0,0);  // end user data
if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```

In this example, a type 1 (one off) elapsed time reminder is created. The subtype is specified as 7 to uniquely index the reminder and with other user data specified by parameters user1-4.

The first and only reminder alert will be generated 1 minute from its creation.

Example 2 – recurring ET reminder:

```
int result;
result = create_ET_reminder(2, 127,
         0, 0, 0, 0,   // start immediately
         0, 0,30, 0,   // generate 30 sec alerts
         0, 0, 0, 0,   // not used for type 2 rems
        10,32,904,0);  // end user data
if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```

In this example, a type 2 (recurring without end) elapsed time reminder is created with a recurring frequency of 30 seconds. The subtype is specified as 127 to uniquely index the reminder and with other user data specified by parameters user1-4.

The first reminder alert will be generated immediately from time of creation and thereafter every 30</td></tr>
</table>

Function name:	create_ET_reminder

seconds indefinitely.

Example 3 – recurring with duration ET reminder:

```
int result;
result = create_ET_reminder(3, 255,
         1, 0, 0, 0, // start in 1 hour
         0, 0,45, 0, // repeat every 45 secs
         2, 30,0, 0, // repeat for 2:30 hrs
         5, 0, 0, 0);// end user data
if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```

In this example, a type 3 (recurring with duration) elapsed time reminder is created. The subtype is specified as 255 to uniquely index the reminder and with other user data specified by parameters user1-4.

The first reminder alert will be generated after one hour from time of creation and thereafter every 45 seconds until 2 hours 30 minutes has elapsed from the first alert.

Function name:	create_RT_reminder		
Function type:	int		
Function parameters:	int r_type, int r_subtype, LSI start_hrs, LSI start_mins, LSI start_secs, LSI freq_hrs, LSI freq_mins, LSI freq_secs, LSI duration_hrs, LSI duration_mins, LSI duration_secs, int user1, int user2, int user3, int user4		
Purpose:	To create a real-time reminder (RTR)		
Return Values:	#define 'tags'	Value	Notes
	invalid_R_type	-1	#define tag names are self explanatory
	invalid_R_subtype	-2	
	invalid_R_start_in	-3	
	invalid_start_in_mins	-4	
	invalid_start_in_secs	-5	
	invalid_start_in_subsecs	-6	
	invalid_duration_mins	-7	
	invalid_duration_secs	-8	
	invalid_duration_subsecs	-9	
	invalid_freq_mins	-10	
	invalid_freq_secs	-11	
	invalid_freq_subsecs	-12	
	invalid_freq	-13	
	invalid_RT_hrs	-14	Only relevant for RTRs
	invalid_RT_mins	-15	Only relevant for RTRs
	invalid_RT_secs	-16	Only relevant for RTRs
	reminder_list_full	-99	No further free space in the RQ – too many ETR/RTR

<table>
<tr><td>Function name:</td><td colspan="3">create_RT_reminder</td></tr>
<tr><td></td><td></td><td></td><td>definitions active</td></tr>
<tr><td></td><td>RTC_not_configured</td><td>-100</td><td>An attempt to create a RTR when no RTC is configured</td></tr>
<tr><td>Notes:</td><td colspan="3">1 this function will ignore any calls made to it if the RTC is not configured
2 r_type, r_subtype, user1, user2, user 3 and user4 are all of type int. All other parameters are of type LSI - long signed int
3 r_type defines the type of the RTR - 4, 5 or 6, see table below
4 r_subtype is a user definable value in the range 0 to 255
5 start_hrs/mins/secs defines the real-time (24 hour clock) of the first reminder alert.
6 freq_hrs/start_mins/start_secs define the frequency that reminder alerts will be generated after the first (defined by start_hrs/start_mins/start_secs).
7 duration_hrs/duration_mins/duration_secs define the elapsed duration that reminder alerts will be generated for, from the initial start time. A duration of 0:0:0:0 is permissible, see below.</td></tr>
<tr><td></td><td colspan="3">Permissible RTR Types
RTR Types | Meaning
4 | Type 4 defines a one off RTR. Once the alert is raised at the specified time the RTR is deleted and no other alerts will be raised.
5 | Type 5 defines a recurring RTR without end, unless programmatically deleted (see delete_reminder()).
6 | Type 6 defines a recurring RTR with a fixed duration. Reminder alerts will be generated at each frequency. Once the duration has elapsed the RTR is deleted.

Note: for reminder type 6, only, if frequency is set to 0:0:0:0 (i.e. no frequency) then the reminder will finish once the duration time is reached.</td></tr>
<tr><td colspan="4">Examples</td></tr>
<tr><td colspan="4">Example 1 - one off RT reminder:


```
int result;
result = create_RT_reminder(
         4, 29,        // type 4 - one off RTR
         0,  0, 0,     // start at next midnight
         0,  0, 0,     // n/a for type 4 rems
         0,  0, 0,     // n/a for type 4 rems
         1,255, 0,0); // end user data

if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```


In this example, a type 4 (one off, without end) real-time reminder is created to generate its first and only reminder alert at the next midnight (00:00:00 hrs). The subtype is specified as 29 to uniquely index the reminder and with other user data specified by parameters user1-4.

Example 2 – recurring RT reminder:</td></tr>
</table>

Function name:	create_RT_reminder

```
int result;
result = create_RT_reminder(
         5, 11,        // type 5 - recurring RTR
        13, 30, 0,     // start at 13:30:00 hrs
         0, 10, 0,     // generate alerts every 10 mins indefinitely
         0,  0, 0,     // not used for type 5 rems
         0,101,15,0); // end user data

if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```

In this example, a type 5 (recurring without end) real-time reminder is created with a recurring frequency of 10 minutes. The first reminder alert will be generated at the next 13:30:00 hrs and thereafter every 10 minutes indefinitely.

The subtype is specified as 11 to uniquely index the reminder and with other user data specified by parameters user1-4.

Example 3 – recurring with duration RT reminder:

```
int result;
result = create_RT_reminder(
         6, 2,         // type 6 - recurring RTR with a duration
        22, 0,  0,     // start at next 22:00:00 hrs
         1, 0,  0,     // generate alerts every 1 hr
        23, 0,  0,     // repeat alerts for 23 hrs
         6, 10,-1,0); // end user data

if (diags_on && result < 0){
// error reported, result defines what the error is
   ...
}
```

In this example, a type 6 (recurring with duration) real-time reminder is created. The first reminder alert will be raised at the next 22:00:00 hours, and then every 1 hour for 23 hours.
The subtype is specified as 2 to uniquely index the reminder and with other user data specified by parameters user1-4.

Function name:	delete_reminder
Function type:	void
Function parameters:	int type, int subtype
Purpose:	The function will delete (remove) an ETR/RTR from the reminder list with the given reminder type and subtype
Return Values:	1 = success, for successful deletion -1 = fail, for no entry found with given type/subtype combination
Notes:	1. If there is more than a single ETR/RTR in the reminder list of the same type/subtype, then a call to this function will only remove the first such entry 2. Careful planning should ensure that ETRs/RTRs are unique, i.e. have unique type/subtype combinations

Examples

```
int result;
// delete reminder type = 5 (RTR, recurring), subtype = 127
result = delete_reminder(5,127);
if (result == fail){
    // not found!
    ...
} else {
    // reminder deleted
    ...
}
```

Function name:	print_free_chain
Function type:	void
Function parameters:	None
Purpose:	The function will print the current state of the free chain used to support the reminder queue
Return Values:	None
Notes:	1. For the function to produce output the Boolean variable 'diags_on' must be set to true 2. Interrupts are disabled whilst the function completes its processes

Examples

```
print_free_chain();
```

Function name:	print_reminder
Function type:	`void`
Function parameters:	`int r_entry`
Purpose:	To print the data held in the given remind list entry (supplied parameter)
Return Values:	None
Notes:	1. For the function to produce output the Boolean variable 'diags_on' must be set to true 2. Interrupts are disabled whilst the function completes its processes 3. If a reminder entry is not used (empty) this is also reported

Examples

```
int R_entry;
for (R_entry = 0; R_entry < max_reminders; R_entry++){
  print_reminder(R_entry);
}
```

Function name:	print_RQ
Function type:	`void`
Function parameters:	None
Purpose:	The function will print the active Reminder Queue and show details of any outstanding (unprocessed) reminder alerts
Return Values:	None
Notes:	1. For the function to produce output the Boolean variable 'diags_on' must be set to true. 2. Interrupts are disabled whilst the function completes its processes 3. Used in conjunction with the `print_free_chain()` function, `print_RQ()` can provide an entire 'picture' of the status of the reminder queue

Examples

```
print_free_chain();
print_RQ();
```

Function name:	resume_reminders
Function type:	`void`
Function parameters:	None
Purpose:	The function resumes the processing of ETRs and RTRs following a call to the `suspend_reminders()` function
Return Values:	None
Notes:	1. The routine is analogous to interrupts() 2. Resumes processing of ETRs and RTRs following a call to the `suspend_reminders()` function

Examples

```
resume_reminders();
```

Function name:	scan_RQ
Function type:	int
Function parameters:	None
Purpose:	The function examines the Remind Queue (RQ) and obtains the next reminder alert from it if the queue is not empty. If empty then queue empty status is returned
Return Values:	0 Success -1 No reminder requests, Reminder Queue (RQ) is empty
Notes:	1. If a reminder alert is obtained from the RQ, then the following variables are initialised from the ETR/RTR reminder alert data: R_status set to 0 if this alert is not the last alert, otherwise it is set to 1 (final_alert)[5] R_type set to the reminder type alerted (1 - 6) R_subtype set to the subtype the end user defined at reminder creation for the alerted reminder R_user1 set to the end user data defined when the ETR/RTR was created R_user2 ditto R_user3 ditto R_user4 ditto 2. The variable data established following a reminder alert are a very powerful aid to decision support, program flow and control. These data can be used to craft complex asynchronous solutions.

Examples

```
if (scan_RQ() != no_reminder_requests) {
  switch (R_subtype) {
    case heart_beat:
      analogWrite(heart_beat_pin, hb_intensity);
      // toggle heart beat output level for next pass
      if (hb_intensity == 255) {hb_intensity = 0;}
      else {hb_intensity = 255;}
      break;
    case midnight:
      // * midnight processing.
      today_day_number++; // day_number for today, a new day
      today_day_of_week = (today_day_of_week + 1) % 7;// next day of
                                                        week value
      break;
    default:
      Serial.print("!Spurious switch value=");
      Serial.println(R_subtype);
      Serial.flush();
      display_now_date_time();
      break;
    }
```

[5] It follows that for ETRs of type 1 (one off) and RTRs of type 4 (one off) R_status is always set to 1 (final_alert).

Function name:	resume_reminders

```
  } else {
    //  *******************************************************
    //  Reminder queue is currently empty, so do other things..
    //  *******************************************************

  }
} while (true);
```

Function name:	suspend_reminders
Function type:	void
Function parameters:	None
Purpose:	To temporarily suspend the processing of ETRs and RTRs
Return Values:	None
Notes:	1. The routine is analogous to noInterrupts() 2. Use sparingly and for as short a period as possible 3. ETRs and RTRs are not processed while suspended, so be aware of extended timings on ETRs and the possibility that RTRs may miss their schedule real-time alerting times 4. The opposite function is resume_reminders()
Examples	

```
suspend_reminders();
```

Supplementary Date/Time Functions & Declarations

To supplement the framework, a number of date/time functions, variables and definitions are provided that can be used to manipulate dates. Note that similar functionality may also be available from the configured RTC library.

Date/Time Functions

Function name:	check_date		
Function type:	`int`		
Function parameters:	`int day, int month, int year`		
Purpose:	Given the date as day, month, year the functions determines if it is a valid date, or otherwise		
Return Values:	#define 'tags'	Value	Notes
	`valid_date`	1	
	`invalid_day`	-1	
	`invalid_month`	-2	
	`invalid_year`	-3	Year is before origin year for date calculations
Notes:	1. Takes the given date as day, month, year and validates it. Takes into account leap years.		

Examples

```
result = check_date(d, m, y);
if (result) < 0){
   // date is in error, result defines what error is
  ...
}
```

Function name:	date_from_day_number
Function type:	`void`
Function parameters:	`int day_number, int &day, int &month, int &year`
Purpose:	Takes the given day_number value and converts it into a real date
Return Values:	None
Notes:	The date is returned in the day, month and year parameters which are declared by address

Examples

```
int dn, d, m, y;
dn = day_number(17,5,2020)+7; // one week on
date_from_day_number(dn,d,m,y);
Serial.print(d);Serial.print("/");
Serial.print(m);Serial.print("/");
Serial.println(y);
Serial.flush();
```

will produce: `24/5/2020`

Function name:	day_of_week
Function type:	int
Function parameters:	int day, int month, int year
Purpose:	Takes the given date (day, month, year) and returns a day of the week index value
Return Values:	The function returns the day of the week index value in the range 0 to 6, with 0 being Sunday
Notes:	Days of the week are referenced by the array char days_of_week[7][12] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

Examples

```
// day of the week Christmas falls in 2022
Serial.print(days_of_week[day_of_week(25, 12, 2022)][0]);
```

Function name:	day_number
Function type:	int
Function parameters:	int day, int month, int year
Purpose:	Takes the given date (day, month, year) and returns the number of elapsed days since the origin date (1/1/2020)
Return Values:	The number of elapsed days of given date from the origin date
Notes:	1. If manipulating dates then this function is essential. It returns the number of elapsed days since the origin date and takes into account all leap years 2. Use this function (and its reverse function) when doing arithmetic with dates 3. The reverse process is to determine a date from a day number - see date_from_day_number()

Examples

```
int dn, d, m, y;
dn = day_number(17,5,2020);
Serial.println(dn);
Serial.flush();
```

Will produce: `138` (the number of days elapsed since the origin date 1/1/2020)

Function name:	display_now_date_time
Function type:	void
Function parameters:	None
Purpose:	Displays the current time and date of the RTC to the serial port
Return Values:	None
Notes:	1. The call will be ignored if the RTC is not configured 2. Reads the RTC to assemble and print the current date and time

Examples

```
display_now_date_time();
```

Function name:	display_now_date_time
will produce date/time, for example: 17/5/2020, Sunday, 14:35:57	
Function name:	leap_year
Function type:	`int`
Function parameters:	`int year`
Purpose:	Determines if the given year is a leap year or not
Return Values:	`true` if a leap year `false` if not a leap year
Notes:	1. Takes the given year and returns **true** if a leap year, otherwise **false**. 2. The function utilises full leap year checking: • if divisible by 4 then a leap year, unless • divisible by 100 then not, unless • divisible by 400 then a leap year. 3. Remember Y2K?

Examples

```
year++;
if (leap_year(year)){
    // this is a leap year
    ...
}
```

Function name:	seconds_since_midnight
Function type:	`long unsigned int`
Function parameters:	None
Purpose:	Calculates the number of seconds from last midnight of the current time
Return Values:	The number of seconds since the last midnight, or 0 if the RTC is not configured.
Notes:	1. This function is used by the RTR scanning process to determine if RTR times have been reached 2. It can be used by end user code if needed 3. Note that this function will return a value of 0 if the RTC is not configured.

Examples

```
Serial.print("number seconds since midnight = ");
Serial.print(seconds_since_midnight());
Serial.println(" secs.");
Serial.flush();
```

#define declarations

The framework has definitions configured for end user reference and use to aid readability. Use these when testing for days of the week, months of the year, etc.

check_date() Function Return Values	
Definition	Ascribed Value
#define valid_date	1
#define invalid_day	-1
#define invalid_month	-2
#define invalid_year	-3

Day of Week Definitions	
Definition	Ascribed Value
#define Sunday	0
#define Monday	1
#define Tuesday	2
#define Wednesday	3
#define Thursday	4
#define Friday	5
#define Saturday	6

Create ETR/RTR Reminder Errors	
Definition	Ascribed Value
#define invalid_R_type	-1
#define invalid_R_subtype	-2
#define invalid_R_start_in	-3
#define invalid_start_in_mins	-4
#define invalid_start_in_secs	-5
#define invalid_start_in_subsecs	-6
#define invalid_duration_mins	-7
#define invalid_duration_secs	-8
#define invalid_duration_subsecs	-9
#define invalid_freq_mins	-10
#define invalid_freq_secs	-11
#define invalid_freq_subsecs	-12
#define invalid_freq	-13
#define invalid_RT_hrs	-14
#define invalid_RT_mins	-15
#define invalid_RT_secs	-16
#define reminder_list_full	-99
#define RTC_not_configured	-100

General #definition tags		
Definition	Ascribed Value	Comments
#define no_reminder_requests	-1	Value returned by the scan_RQ() function if there are no outstanding reminder alerts in the queue
#define success	1	Used generally and widely
#define fail	-1	Used generally and widely
#define inactive	0	Used by remind list scanners to determine if entry is active or not
#define final_alert	1	Used by remind list scanners to determine if entry is a final alert. If so, it is passed on to end user via scan_RQ()
#define heart_beat_pin	13	digital pin number for visible heart beat
#define heart_beat	254	ETR subtype for heart beat LED monitor
#define midnight	255	RTR subtype for midnight processing

Month Definitions	
Definition	Ascribed Value
#define Jan	1
#define Feb	2
#define Mar	3
#define Apr	4
#define May	5
#define Jun	6
#define Jul	7
#define Aug	8
#define Sep	9
#define Oct	10
#define Nov	11
#define Dec	12

Reminder Types	
Definition	Ascribed Value
#define ET_oneoff_type	1
#define ET_recurring_type	2
#define ET_repeat_duration_type	3
#define RT_oneoff_type	4
#define RT_recurring_type	5
#define RT_repeat_duration_type	6

Useful Variable Declarations

The framework includes several useful declarations (variables) that can be helpful to the end user designer. These are summaries below:

Declaration	Purpose/Description
`byte days_in_month[13] =`	The array is preset with: `{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};` To note - entry 0 not used, referencing is from 1 to 12
`char days_of_week[7][12] =`	The array is preset with: `{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};` And can be used as needed for day of week conversion

`int origin_day = 1`
`int origin_month = 1`
`int origin_year = 2020`
	These three variables work as one to define the start date from which date functions, such as day_number(), date_from_day_number(), etc, calculate the number of the index value for a specified date.

`volatile int R_status`
`volatile int R_type`
`volatile int R_subtype`
`volatile int R_user1`
`volatile int R_user2`
`volatile int R_user3`
`volatile int R_user4`
	These variables are set up as a set by scan_RQ so that they can be referenced from the main end user code processing following a timed reminder alert being taken off the queue.
	
	Note that if an alert is not available, then each of these variables is set to 0.
	
	A fuller description can be found by reference to the scan_RQ() function description.

`int today_day_number`	If the RTC is enabled then this variable is used to record the day number of today's date. It is initialised during setup() for the current date and kept up to date via the midnight RTR processor, so can be relied on throughout any code to represent the current day number. It is a fundamental value if there is a need for date arithmetic processing.
`int today_day_of_week`	If the RTC is enabled then this variable is used to record the day of the week index value of today's date. It is initialised during setup() for the current date and kept up to date via the midnight RTR processor, so can be relied on throughout any code to represent the current day of the week index value. The range of values it reflects is 0, ..., 6 with 0 being Sunday. It is a fundamental value if there is a need for date arithmetic processing.

Example:

```
// check to see if we need to alter the RTC dow to daylight saving changes
int month;
```

```
month = now.month();  // read current month from the RTC
If (month) == Oct | month == Mar){
   If ( today_day_of_week == Sunday){
      // check for daylight saving changes
      // calculate if this is the last Sunday in the month.
      // if so, change the RTC time
      ...
   }
}
```

Example:

```
If (R_status == final_alert){
    // no further alerts for this subtype, so finish up the process
    ...
} else{
    // not a final alert, so continue this subtype process
    ...
}
```

Appendix B

A Note About Time (hmss) Parameters

For elapsed time (ET) reminder creation (the create_ET_reminder(...) function), all times are specified in hours, minutes, seconds and subsecond format (hmss). This is applicable to all ET reminder types (1, 2, and 3). However, it should be understood that these times are NOT real-time, but periods of time. Real-time (RT) reminders are supported by the sister function create_RT_reminder(). The discussion below refers exclusively to periods of elapsed time.

Parameters - the h(our), m(inute) and s(econd) parameters are all straight forward with the m and s values conforming to general validity constraints for time. The h value can be any practical value, even greater than 24. However, this is not so in respect of s(ubsecond) parameters. These parameters behave in a non-intuitive manner and it is important that this feature is thoroughly understood before designing ETRs.

There is a balance to be struck between a). providing a high resolution for ETR alerting and b). processing capacity for the end user code. It is clearly not helpful if most of the processor time is taken up dealing with ETR management and processing leaving little for end user code. It is this balance of needs that the framework has been designed around.

The framework cannot function without relying on a fundamental timing source. The timing source selected for framework base timing is sourced from microcontroller timer0 (or timer2), established at configuration time, depending on end user needs for use of such functions as delay(), millis(), micros(), etc). The configured timer source is initialised to generate timer interrupts every millisecond (msec), or every 1 KHz.

However, whilst the framework would happily operate at such a frequency, it would not be a sensible design feature. Furthermore, ETR processing at such a high frequency may not be necessary for the vast majority of end user application designs and needs.

To provide a degree of granularity and flexibility, the frequency (cycle rate) for the processing of ETRs can be configured to be an integral number of timer0/2 cycles, with some conditions applying. That is, the ETR scan rate can be set to between 1 Hz and 1 KHz through the setting of a scan rate parameter ('ETR_R_list_scan_freq).

The scan rate parameter ETR_R_list_scan_freq controls how often ETRs in the reminder list are scanned and processed. For example,

- A setting of ETR_R_list_scan_freq = 1, will cause the timed reminder list is scanned and processed every 1 msec (i.e. 1,000 times per second) .
- A setting of ETR_R_list_scan_freq = 10 will cause the timed reminder list to be scanned and processed every 10 msecs (i.e. 100 times per second)
- A setting of ETR_R_list_scan_freq = 100 will cause the timed reminder list to be scanned and processed every 100 msecs (i.e. 10 times per second)
- A setting of ETR_R_list_scan_freq = 250 will cause the timed reminder list to be scanned and processed every 250 msecs (i.e. 4 times per second)
- Etc.

The OOTB preset value for ETR_R_list_scan_freq is 100, or 10 scans per second (10 Hz). Choosing a suitable scan rate for ETR scanning/processing (ET_R_list_scan_rate) is a fundamental design requirement. As part of the end user design stage, it is essential to understand what frequency (resolution) is appropriate to meet needs. The value of the ET_R_list_scan_rate parameter should

be chosen to be as large as possible to meet design needs as this will give more processor time to end user code.

It is only once the value of the parameter ET_R_list_scan_rate is selected that s(ubsecond) parameters can be understood. Recall that the value set for ET_R_list_scan_rate determines how many times the timed reminder list is scanned and processed each second. It is therefore the value defined for ETR_R_list_scan_freq that dictates how s(ubsecond) parameters are also defined and validated – the validity range is not 0, ..., 59, but in the range 0, ..., (ETR scans per second - 1).

For example:

- A setting of ETR_R_list_scan_freq = 1 (1,000 scans per sec), would yield a s(ubsecond)
- validity range of 0, ... , 999
- A setting of ETR_R_list_scan_freq = 10 (100 scans per sec), would yield a s(ubsecond)
- validity range of 0, ... , 99
- A setting of ETR_R_list_scan_freq = 100 (10 scans per sec), would yield a s(ubsecond)
- validity range of 0, ... , 9
- A setting of ETR_R_list_scan_freq = 250 (4 scans per sec), would yield a s(ubsecond)
- validity range of 0, ... , 3
- Etc.

All possible values for ETR_R_list_scan_freq are shown in the table below:

ETR_R_list_scan_freq Settings (msecs, MHz)	Scans per sec (Hz)	Valid ranges for create_ET_reminder 'subsecs' parameters
1	1,000	0 - 999
2	500	0 - 499
4	250	0 - 249
5	200	0 - 199
8	125	0 - 124
10	100	0 - 99
20	50	0 - 49
25	40	0 - 39
40	25	0 - 24
100	10	0 - 9
200	5	0 - 4
250	4	0 - 3
500	2	0 - 1
1000[6]	1	0 - 0 (alert frequency is effectively every second, so set the seconds parameters instead and leave the subsecs parameters at 0)

Table – ETR_R_list_scan_freq Settings

It should be no surprise that the above values are a). an exact divisor of the timer0/2 frequency and b). asymmetric – their product must exactly total 1,000 Hz (timer0/2 interrupt frequency).

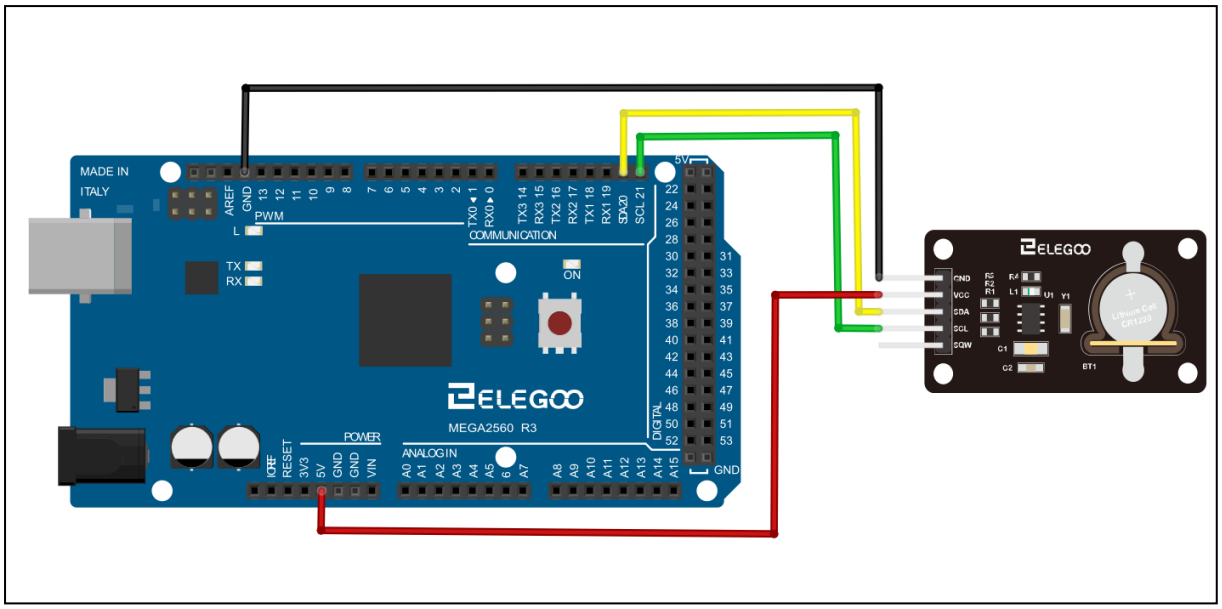
Validation of the s(ubsecond) parameter is therefore a function of the ETR_R_list_scan_freq setting which defines how many scans per second ETRs are scanned/processed.

[6] Note that the heart beat monitor LED will cycle at 1/2 Hz at this value of frequency.

Appendix C

Elegoo DS1307 RTC Module / Mega 2560 Wiring Plan

(Other microcontrollers and RTC modules available)



Wiring Plan	
Microcontroller Digital Pin	DS1307 Connection
20, SDA	SDA
21, SCL	SCL
+5v	VCC
GRN	GND

Appendix D

Timed Reminder Definition Table

This table can be used to define and catalogue the ETRs/RTRs used in an application. It should be used during the design stage and following to understand the details of all timed reminders implemented.

Application name/reference:																			
Author:															**Date:**				
ETR_R_list_scan_freq:				**subsec range:**										**End User Parameters**				**Comments**	
Name / ID	**ETR/RTR Control Parameters**														R_user1	R_user2	R_user3	R_user4	
	Type	Subtype	Start				Frequency				Duration								
			H	M	S	s	H	M	S	s	H	M	S	s					