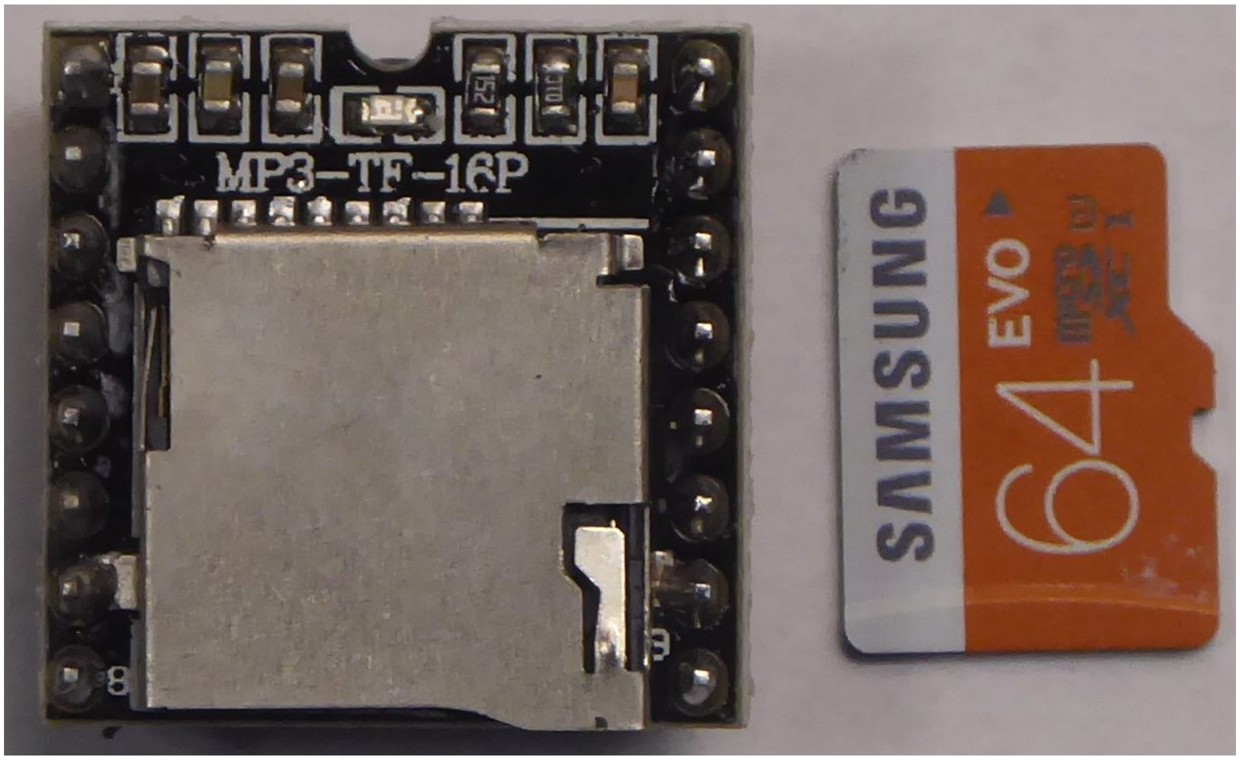
Over View:
One of the main features of the PRO race controller (PRO) is that it is relatively easy to configure or program. There are two very different but basic programmable components to the system. Either or both of these systems can use user authored or selected from library.

- The first is the audio sound files.
- The second is the actual race start sequence from a spread sheet

Programming the sound modules:

How it works:
There are two identical audio systems, one for competitors via VHF radio broadcast and the other for the signal boat volunteers via a speaker mounted inside the controller box. Both are programmed the same way.

The sound modules used by the PRO hardware are common DFPlayer modules. They play MP3 files that have been copied on to a micro SD card commonly found in cell phones. While the DFPlayer is a very convenient solution for this and other applications, it is poorly and somewhat inaccurately documented. The procedures outlined here are as a result of finding what works and should be followed.



DFPlayer module and microSD memory card

The player reads both FAT 16 and FAT 32 directories but contrary to publications, the player does not appear to understand file names or volume names. The player appears to be “slot” oriented. The slots are assigned in the order in which the files are copied onto the Micro SD card. The file in slot 1 is the first file copied onto an empty card. Therefore *0001_file name.mp3* is in slot 1 if it was copied first. If *0004_different file name.mp3* was copied first onto a blank card, it would be file 1 since names are meaning less.

The numbered file name structure shown is a good format because it makes sense to humans and can be used to help keep files in order including sorting in the PC file window. The sorting by the PC file tool appears to be for its display only and does not correlate any order. This includes viewing the SD Micro card with the PC..

Copy files to SD Micro using a PC:

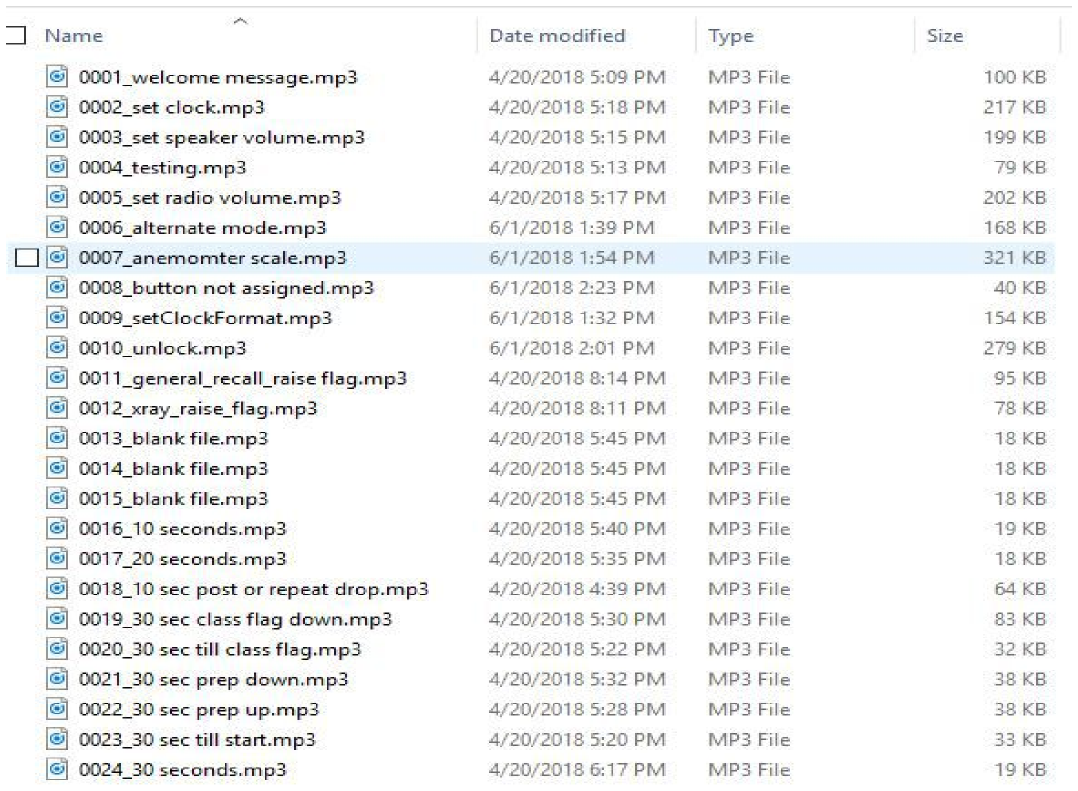
- Mount and open card on PC
- If the entire file system is to be copied, make sure no files exist. Directories are meaningless so files are copied to the root directory. It is recommended to do a fast format on the SD card. This is done by opening the properties of the card and selecting fast format.
- Open the source directory and sort. This is where the 4 digit prefix on the file name is handy. First file or low number is on top.
- Files can be dragged across singly in proper order, low number first. -or- for a block copy, select the bottom file and shift select the top file. Make sure this select order is followed, it dictates the copy order. Drag the selection to the SD window
- The file directory on the SD card can be viewed but since it is sorted, this is no indication of the order copied or "slots"
- File order can be viewed by using the DOS window. Mount SD card. Windows "run cmd" to open DOS window. In DOS window: select SD card i.e. "*G: enter*". Prompt becomes*"G:\*>" . From there type "*dir enter*". The files will be listed in order

How it's tied together:

The spread sheet generated start sequence (discussed later) selects file slots for the message to be played. Therefore there is an implied relationship between the sound slots on the card and the file referenced by the spread sheet that can not be deviated from.

The system reserves the first slots on both of the sound systems for it's use. The radio directory reserves slots 1-9 and the speaker directory reserves 1-15. The speaker has more reserved slots because it contains the audio user interface for system setup. Empty slots have a 1 second silent recording that holds the slot. If you are using a new voice or language, the new recording must contain the same messages for the slot to maintain the relationship with the firmware. If you are building on existing systems, the messages can be re-recorded with additional files appended.

It is important that some sort of overall file convention or order be maintained. A good start would be to use the supplied samples. Do not deviate from the slot order..Record them to a different voice or language but keep the message intent the same. Keep the message timing the same. Horn sounds should coincide with the boat horn. Sailors expect or it can be explained that sound takes time to travel but not when the competitor is at the boat end of the line. Append more files as needed but never delete or change the slot/message relationship going forward. The PRO controller will support many start sequences but they all reference the same audio system.



Name	Date modified	Type	Size
0001_welcome message.mp3	4/20/2018 5:09 PM	MP3 File	100 KB
0002_set clock.mp3	4/20/2018 5:18 PM	MP3 File	217 KB
0003_set speaker volume.mp3	4/20/2018 5:15 PM	MP3 File	199 KB
0004_testing.mp3	4/20/2018 5:13 PM	MP3 File	79 KB
0005_set radio volume.mp3	4/20/2018 5:17 PM	MP3 File	202 KB
0006_alternate mode.mp3	6/1/2018 1:39 PM	MP3 File	168 KB
0007_anemomter scale.mp3	6/1/2018 1:54 PM	MP3 File	321 KB
0008_button not assigned.mp3	6/1/2018 2:23 PM	MP3 File	40 KB
0009_setClockFormat.mp3	6/1/2018 1:32 PM	MP3 File	154 KB
0010_unlock.mp3	6/1/2018 2:01 PM	MP3 File	279 KB
0011_general_recall_raise flag.mp3	4/20/2018 8:14 PM	MP3 File	95 KB
0012_xray_raise_flag.mp3	4/20/2018 8:11 PM	MP3 File	78 KB
0013_blank file.mp3	4/20/2018 5:45 PM	MP3 File	18 KB
0014_blank file.mp3	4/20/2018 5:45 PM	MP3 File	18 KB
0015_blank file.mp3	4/20/2018 5:45 PM	MP3 File	18 KB
0016_10 seconds.mp3	4/20/2018 5:40 PM	MP3 File	19 KB
0017_20 seconds.mp3	4/20/2018 5:35 PM	MP3 File	18 KB
0018_10 sec post or repeat drop.mp3	4/20/2018 4:39 PM	MP3 File	64 KB
0019_30 sec class flag down.mp3	4/20/2018 5:30 PM	MP3 File	83 KB
0020_30 sec till class flag.mp3	4/20/2018 5:22 PM	MP3 File	32 KB
0021_30 sec prep down.mp3	4/20/2018 5:32 PM	MP3 File	38 KB
0022_30 sec prep up.mp3	4/20/2018 5:28 PM	MP3 File	38 KB
0023_30 sec till start.mp3	4/20/2018 5:20 PM	MP3 File	33 KB
0024_30 seconds.mp3	4/20/2018 6:17 PM	MP3 File	19 KB

Screen shot of speaker file directory

Preparation is extremely important !

Recording the files: The system was developed using *Audacity* . It is free (contributions recommended) and works well. This document will not discuss recording techniques or how to use a recording tool. Most of the defaults in Audacity work well. Use monaural. Experiment with your microphone levels. Keep the level constant from file to file and make the levels high without clipping. Be aware of background noise. Practice, practice, practice.

The files need to be timed to coincide with events in the start sequence. Sound snippets like whistles or horns can be found on the internet or edited out the existing samples. Once again, it is important that sounds, like an air horns, be timed accurately with the boat signaling device. It may be helpful to put notes like file lengths in the meta-data portion of the mp3 file.

Programming the start sequence:

The start sequence is a spread sheet that is formatted to be read and compiled into a down loadable program written in the programming language C. It is important that none of the formatting should changed and the rules outlined here be explicitly followed.

Refer to sample spread sheet scheduleONE.xls on http://www.windwhisper.org/PROcontroller/ or the copy at the end of this document. This actually is the spread sheet used for two different buttons in the first controller prototype. It contains two start sequences, the six minute repeat/postpone flag drop and the five minute start. The six minute actually rolls into the five minute so one file can be used for both sequences. The two buttons are just assigned different entry points into the same sequence. Entry points are discussed in the section on assigning buttons.

Lines 1 thru 11 contain definitions. Each command has a number assigned to it. The C compiler substitutes the number for the command. Look at lines 1 and 11. They open the sequence with " ***/**** " and close with " ****/*** ". Everything in between is treated as a comment by the compiler and therefore ignored. Therefore the definitions here are a copy of definitions that are defined else where. The command names can be be cut and pasted into the spread sheet to make things easier and reduce typo's. Remember, even

Line 14 begins the array *scheduleONE* . That name can be edited to something else, like *scheduleTWO*. The name is the **only** thing it that row that can be changed and that cell must contain everything else already in it.
Line 17 column A opens the array with *"* ***{*** ".
Line 18 the first instruction in the array. All the lines in the array have the same format. It opens in column B with a " ***{*** "

- Column C is blank just for formatting.
- Column D contains the time left in the start sequence. The time format is minutes and seconds with the colon implied..that is not written. Writing the colon will cause a compiler error.
- Column E contains a " , " a delimiter used by the C language
- Column F contains the command. Cut and paste the command from the comments above. Anything other than an exact copy, including case, will cause a compile error.
- Column G contains another comma, a delimiter for the command argument
- Column H has a number or operand that is used by the command. Times in this field are ½ second increment so a one second boat horn would be the number 2.
- Column I contains " ***},*** " and that is the syntax for C to close that row in the array

Line 18 executes as at 6 minutes 10 seconds before the race starts (count down) turn on the radio transmitter for 5 seconds. (½ second resolution)
Line 19 also at 6:10 send message 13 to the radio
Line 20 similarly at 6:10 plays message 18 on the speaker.

Note: remember the relationship between the message number and then file structure on the SD cards

Line 119 column A closes the array with " ***};*** "

Additional tips:

The compiler needs to know the number of rows in the array. That is entered automatically by the spread sheet in cell 14 E. The spread sheet counts the number of filled cells in column B including comments. It does not care what is in the cell so there can not be anything other than the opening "{" for each line in the array. Anything else in that column, including comments, will cause an erroneous count.

The C compiler treats everything after the double slash // til the end of the line as a comment. Note that this does not mean embedded in the line. the sequence /* ….. */ treats every thing in the middle, including multiple lines, as comments. Remember: nothing in column B...never.

I like to insert a blank row with all the mandatory syntax. I then cut and paste that row into additional rows. I then fill in the time, command and operands into each row. Unused rows can be deleted later. This method seems to cut down on the typo's.

If the command Reload_Count is used, it must be the last instruction executed in the schedule. Since it changes count down counter, commands after it might not get executed. This command provides the rolling start. If Reload_Count is not used, the sequence terminates after the last command and the system wait for the next button press.

Assigning Buttons:

Each button must be bound or assigned to a schedule. The button also needs an assignment for an initial or starting time for the count down timer. That assignment is the entry point into the schedule. The buttons may share the same schedule but have different entry points. An example is the six and five minute buttons, the five minute being a later entry into the six minute.

Unused buttons still need a sequence and entry point. The same dummy file can be used for all the unused buttons. My dummy file gives a long sound on the beeper and plays an button unassigned messag.

The assignments are in a file called button_assignment.h. This file is compiled with the rest of the source files including the schedule files . A copy of the assignment file, used in the prototype, with explanations follows below.

```
/***********************************************
There are 4 buttons in the system that are programmable
for use when the system is in standby waiting to enter a start
sequence. The buttons may be used for other things when the
system is not ready to begin a start sequence.

The system may have one or  multiple start schedules. For our purposes
we have four that have been prepared elsewhere with a spread sheet.
Each schedule is an array that has a case sensitive name. That name
must be entered into the appropriate button i.e. but3sched is attached to
ScheduleONE in the first line below.

The schedule must have an starting value for the count down counter...time
before race starts. the format is minutes and seconds. The colon in the time
is implied and can not be used.

In our example below we are using a six minute sequence. but3 begins with the horn
for the dropping of the postpone or repeater. Since the 5 minute sequence is just
a subset of the 6 minute sequence, but1 uses also uses ScheduleONE but enters
at 5 minutes 10 seconds, which is the 5 horn alert.

Every button must have an entry even if it is the same as another button.

Case and spelling is critical so cut and paste is recommended.

 ScheduleZERO  ScheduleONE ScheduleTWO  ScheduleTHREE

***************************************/

#define but0alt_sched ScheduleTWO
#define but0alt_entry 2            // Button not assigned
#define but1alt_sched ScheduleTWO
#define but1alt_entry 2            // Button not assigned
#define but2alt_sched ScheduleTWO
#define but2alt_entry 2            // Button not assigned
#define but3alt_sched ScheduleTWO
#define but3alt_entry 2           // Button not assigned


#define but0_sched ScheduleZERO  // 3 minute sequence
#define but0_entry 305
#define but1_sched ScheduleTWO  // Button not assigned
#define but1_entry 2
#define but2_sched ScheduleONE   // 5 minutes 10 seconds
#define but2_entry 511
#define but3_sched ScheduleONE   // 6 minutes 0 seconds
#define but3_entry 611
```

Compiling and downloading:

The actual compiling and downloading the spreadsheet generated schedules involve the firmware source files and are beyond the scope of the end user.

Spread sheet schedules are outputted as csv files with blank delimiters. For C language convention sake the .h delimiter is used so the csv outputted file becomes ScheduleONE.h

Presently all the files including the source files for the firmware are complied and linked into one Intel hex format file. The processor used is an Atmel AT89c51rd2, a vintage 8051, but current and more than adequate for the job. The SDCC (small device C compiler) compiler and linker generates the hex file. The download is done through the uart using an Atmel PC utility called FLIP. Flip erases and programs the hex file into the flash memory on the processor. A USB to asynchronous converter (FTDI) dongle connects the PC running FLIP to the programming/debugging port on PRO race controller board.

Opening the box:

Unless the box needs to be opened, such as installing new audio micro SD memory files, do not open the box especially if it is installed in a wet environment.

A lot of care went into the design to make it water resistant. The enclosure selected is a Bud industries IP65 rated poly-carbonate box PN-1325-C. The only penetrations are an IP-67 rated bulkhead connector and a small penetration in the cover for the switches. The cover penetration has a circuit board for domed switches epoxied over it. Over the top of the switch board and domes is the switch membrane with a waterproof adhesive and additional adhesive around the edges.

All that said, care and patience must be taken putting the cover back on. The silicone gasket in the cover must be intact in good shape with no deformities or aged out. The raised lip on the box that mates with the gasket and must have no nicks or other deformities. The screws need to be retracted so they do not engage or falsely locate the cover. The lip of the box must fully engage the gasket in its slot all the way around the box. I like to squeeze it and make sure that it fully engages the entire gasket. The Bud people mention a click. Then start engaging the screws. First in opposite corners then the center. Very lightly. Then make sure the gasket is still compressible by squeezing again around the box. The compression is slight but should be there. Tighten again following the same pattern. Without a torque screw driver it has to be by feel. When you feel that it is starting snug up you are done. There should be a crack all the way around at the seam. The goal is not to pull cover down so tight that it introduces distortions.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
61			// 5 second count down till prep												
62		{		405	,	Boat_beeper	,	1	},	//begin 5 sec count down					
63		{		404	,	Boat_beeper	,	1	},						
64		{		403	,	Boat_beeper	,	1	},						
65		{		402	,	Boat_beeper	,	1	},						
66		{		401	,	Boat_beeper	,	1	},						
67		{		401	,	Transmitter	,	5	},						
68		{		400	,	Radio_Message	,	20	},	// 1 horn second blast					
69		{		400	,	Horn	,	2	},	// 1 horn second blast					
70			// 30 sec to prep flag down												
71		{		133	,	Transmitter	,	7	},						
72		{		132	,	Radio_Message	,	16	},	// "30 seconds to prep flag down"					
73		{		132	,	Speaker_Message	,	21	},	// "30 seconds to prep flag down"					
74		{		130	,	Boat_beeper	,	1	},						
75		{		129	,	Boat_beeper	,	1	},						
76		{		128	,	Boat_beeper	,	1	},						
77			// 20 second 2 beeps												
78		{		121	,	Boat_beeper	,	1	},						
79		{		120	,	Boat_beeper	,	1	},						
80			// 10 sec 1 beep												
81		{		110	,	Boat_beeper	,	1	},						
82			// 5 sec beeper count down for prep drop												
83		{		105	,	Boat_beeper	,	1	},						
84		{		104	,	Boat_beeper	,	1	},						
85		{		103	,	Boat_beeper	,	1	},						
86		{		102	,	Boat_beeper	,	1	},						
87		{		101	,	Boat_beeper	,	1	},						
88		{		101	,	Transmitter	,	6	},						
89		{		100	,	Radio_Message	,	21	},	// 2 second air horn					
90		{		100	,	Horn	,	2	},	// long (2 second) air horn					
91			// 30 seconds till start												
92		{		33	,	Transmitter	,	7	},						
93		{		32	,	Boat_beeper	,	1	},						
94		{		32	,	Radio_Message	,	18	},	// "30 seconds until the start"					
95		{		32	,	Speaker_Message	,	19	},	// "30 seconds until the class flag"					
96		{		31	,	Boat_beeper	,	1	},						
97		{		30	,	Boat_beeper	,	1	},						
98			// 20 seconds till start 2 beeps												
99		{		22	,	Transmitter	,	5	},						
100		{		21	,	Radio_Message	,	15	},	// "20 seconds"					
101		{		21	,	Boat_beeper	,	1	},						
102		{		20	,	Boat_beeper	,	1	},						
103			// 10 seconds till start 1 beep												
104		{		12	,	Transmitter	,	5	},						
105		{		11	,	Radio_Message	,	14	},	// "10 seconds"					
106		{		11	,	Boat_beeper	,	1	},						
107			// 5 seconds till start												
108		{		6	,	Transmitter	,	16	},						
109		{		5	,	Radio_Message	,	12	},	// 5 second voice count down message with horn					
110		{		5	,	Boat_beeper	,	1	},						
111		{		4	,	Boat_beeper	,	1	},						
112		{		3	,	Boat_beeper	,	1	},						
113		{		2	,	Boat_beeper	,	1	},						
114		{		1	,	Boat_beeper	,	1	},						
115		{		0	,	Horn	,	2	},						
116		{		0	,	Clock_Pause	,	30	},	// pause rtc update for scorer to copy start time					
117		{		0	,	LoadOcsCount	,	400	},	// start the OCS counter going					
118		{		0	,	Reload_Count	,	500	},	// rolling start at 5 minutes..this must be the last command for any current second					
119	};														

ScheduleONE