



Column #128, December 2005 by Jon Williams:

Sounds of the Season

Wow, can you believe we're at the end of the year already? – another one gone by so quickly. But it's easy to tell; decorations are everywhere, the crowds in the malls are growing larger every day, and the sound of music is in the air. (Boy, that last bit was cheesy, wasn't it?) Have you ever noticed that? This time of year music seems to be playing everywhere. Seasonal music, of course – I guess there's just more of it for this particular season. So, let's go with the seasonal flow, shall we? Let's build a digital music player to complement the holiday decorating we've done around the house.

A few months ago one of my Texas prop building friends, an exuberant guy named Vern Graner, told me about a cool MP3 player module from Rogue Robotics called the uMP3 (micro MP3) that was BASIC Stamp compatible. It turns out that Vern is right: the uMP3 player is cool, it is BASIC Stamp compatible, and thanks to recent efforts at Rogue Robotics, it's even easier to use with the BASIC Stamp – including the Prop-1 controller.

Now, before we get into this let me start telling you that the program we're going to work with here requires the uMP3 to be running firmware 110.11 or later. You can get this from Rogue Robotics and update the uMP3 player through a serial port. Make sure that you do this before you connect it to your BASIC Stamp module.

Let There Be Music

The uMP3 player has a lot of interesting features, but for the sake of this month's project we're going to keep things simple. What we're going to do is randomly play one of the songs on the player, and won't repeat a song until all have been played. While the program is neat and straightforward, it has a couple tricks inside that will be useful in other projects.

uMP3 Setup

Before we connect the uMP3 player to the BASIC Stamp we need configure it with a terminal program. Once we've adjusted the settings to our liking they're non-volatile, and we don't have to worry about making them again unless we update the firmware (which clears all user settings to default values).

The uMP3 has TTL level serial connections so I used the Parallax USB2SER module to connect the uMP3 player to my PC. The photo in Figure 128.1 shows the USB2SER connected to the uMP3. Note the upside-down orientation of the USB2SER so that it matches the uMP3 connections. And just a comment on the photos: my unit was originally a 111.10 version (hence the label) and didn't have header pins; I added those myself. Later versions include header pins installed by Rogue, so they may look different than mine.

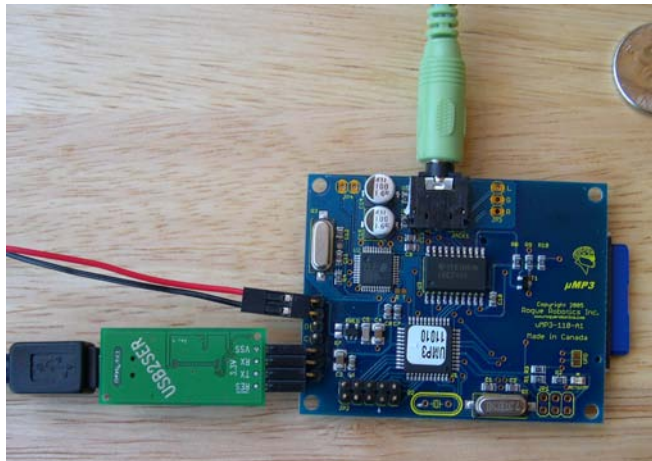
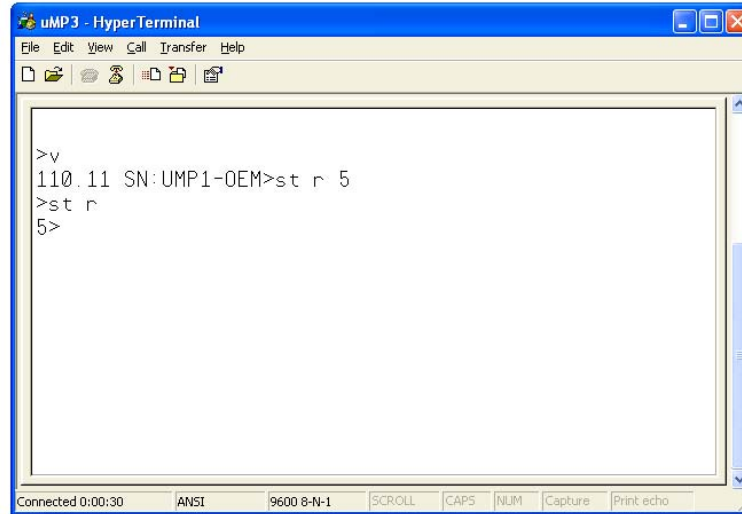


Figure 128.1: USB2SER Connected to the uMP3

Using a terminal program we can check the firmware and update the settings to make things operate smoothly with the BASIC Stamp. Figure 128.2 shows a HyperTerminal session with the uMP3. Out of the box, the uMP3 is configured for 9600 baud so that's what the terminal was set for. At power up the uMP3 scans the connected media (SD card) and when ready it will transmit a ">" to the terminal.



```

uMP3 - HyperTerminal
File Edit View Call Transfer Help
>v
110.11 SN:UMP1-OEM>st r 5
>st r
5>
Connected 0:00:30 ANSI 9600 8-N-1 SCROLL CAPS NUM Capture Print echo

```

Figure 128.2: uMP3 HyperTerminal

We can check the firmware version with the “v” (no quotes) command. Note that all uMP3 commands are terminated with a carriage return, and we especially have to remember this for our BASIC Stamp program. As you can see, I have the version (110.11) that supports a feature we need (prompt delay).

Before connecting to the BASIC Stamp we want to change the prompt delay to five milliseconds. This will let the SERIN instruction load and get ready after a command is sent to the uMP3 with SEROUT – that way we can catch any response and use it. Change the prompt delay with “st r 5” and then confirm the setting entering “st r” without a delay time. The current delay will be reported before the prompt character (it should be 5).

Okay, now we’re ready to get going. Connecting to the uMP3 player falls into the “no-brainer” category, just two wires for serial (TX and RX) and a common ground (I used a cable

hacked from a long-dead servo). Remember that the uMP3 player uses TTL level signals, so it's BASIC Stamp-compatible without any interfacing. Figure 128.3 show the electrical connections between the BASIC Stamp and the uMP3, and the photo in Figure 128.4 shows the player and everything plugged in. Since the output is low level (i.e., for headphones), I connected a set of low-cost amplified computer speakers (green plug). Of course, you'll have to provide (regulated) five volts to the uMP3 as well. If you're experimenting with a Board of Education or Professional Development Board, you can pull power from it.

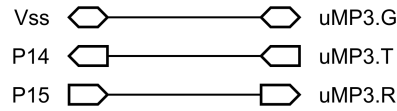


Figure 128.3: uMP3 Connections

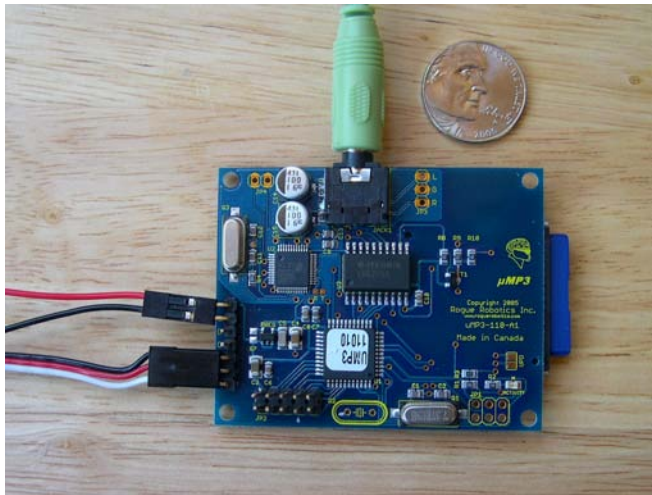


Figure 128.4: uMP3 Player

Play It Again ... But Not Until I've Heard 'Em All

The purpose of our program this month is to play MP3 files – perhaps background music as we're having holiday dinner with friends and family. Let's make it fun, though; let's make it behave like a CD player in "shuffle" mode and force it to play all songs on the list before repeating. Before we attempt to play any of the songs on the uMP3, we need to make sure we're actually connected to it. Here's the Reset section:

```
Reset:
  PAUSE 2000
  SEROUT TX, Baud, [CR]
  SERIN  RX, Baud, 250, Reset, [WAIT(">")]

  lottery = 1225
  GOSUB Reset_Markers
```

At start-up we wait a couple seconds before attempting to get a prompt from the uMP3; this lets the uMP3 player scan the connected media (Note that if you format your SD card for FAT16 it takes a long time to scan – stick to FAT32). After the delay we'll send CR until the prompt comes back. After that we seed the random number generator value (lottery) and reset the song-played markers.

To keep everything manageable I decided to limit the play list to 16 songs. What this lets us do is use a single Word variable to track the songs that have played in the current cycle. Since we'll want to reset the play list at the beginning of the program and when all songs are played, we'll put that function into a subroutine:

```
Reset_Markers:
  markers = $FFFF
  FOR idx = 1 TO NumSongs
    markers = markers << 1
  NEXT
  RETURN
```

This is pretty simple, really, and yet a very handy trick. We start by setting all of the bits in markers to one, and then looping through the number of bits (starting at the LSB) that match our song count, clearing them to zero by shifting the other bits left. If, for example, we had only six songs in our play list, markers would be set to:

```
%1111111111000000
```

Column #128: Sounds of the Season

...by this routine. What if we want to extend the play list past 16 songs? That's not a big challenge, we just use two (or more) variables. Here's how we could reconfigure for 24 songs (one Word plus one Byte required to store the markers):

```
Reset_Markers:
  IF (NumSongs < 17) THEN
    markersHi = $FF
    markersLo = $FFFF
    FOR idx = 1 TO NumSongs
      markersLo = markersLo << 1
    NEXT
  ELSE
    markersHi = $FF
    markersLo = $0000
    FOR idx = 17 TO NumSongs
      markersHi = markersHi << 1
    NEXT
  ENDIF
  RETURN
```

Okay, now we get to the meat of things. The core of the program starts by tumbling the random number generator and then uses that value to select a song from the embedded play list. If the song has already played in this cycle then we will go back to Main and try again. If the current song selection hasn't played yet, and it wasn't the last song in the previous cycle, we'll mark it, pass its number to Play_MP3, and on returning check to see if all of the songs on our list have played. If that's the case, the list gets reset. Here's the core code:

```
Main:
  RANDOM lottery
  theSong = lottery // NumSongs
  IF (markers.LOWBIT(theSong) = 1) THEN Main
  IF (theSong = lastSong) THEN Main

Play_The_Song:
  markers.LOWBIT(theSong) = 1
  lastSong = theSong
  GOSUB Play_MP3
  IF (markers = $FFFF) THEN
    GOSUB Reset_Markers
  ENDIF
  PAUSE SongDelay
  GOTO Main
```

Okay, now for the hard work and, truthfully, it's not that hard. With the song number in hand, what we have to do now is tell the uMP3 to play it. Here's where we need to make a choice: We could choose to make the code very simple by forcing the file names into a non-friendly mode (e.g. "F001.MP3") or, we could work a bit with our program so that we can use the file names as-is.. My choice is to spend a little effort writing code so that we don't have to do anything special with the file names on the uMP3 player module.

Let's say we had "Jingle Bells.MP3" in the root folder of our uMP3 and wanted to play it using a terminal program. We'd enter this command:

```
>PC F /Jingle Bells.MP3 [Enter]
```

If we keep all of our files in the root of the SD card, and know that all files have the ".MP3" extension, all we have to do is store the song title; we can do that like this:

```
MP3s    DATA    "Jingle Bells", 0
```

Using the z-string approach will allow us to scan through the memory to find the selected song, we just need to know where the list starts and what song number to play. Let's have a look at the code that sends the play command and song name to the uMP3.

```
Play_MP3:
  eePntr = MP3s
  IF (theSong > 0) THEN
    zCount = 0
    DO
      READ eePntr, char
      eePntr = eePntr + 1
      IF (char = 0) THEN
        zCount = zCount + 1
      ENDIF
    LOOP UNTIL (zCount = theSong)
  ENDIF

  SEROUT TX, Baud, ["PC F /"]
  DO
    READ eePntr, char
    eePntr = eePntr + 1
    IF (char = 0) THEN EXIT
    SEROUT TX, Baud, [char]
  LOOP
  SEROUT TX, Baud, [".MP3", CR]
  PAUSE 100
```

Column #128: Sounds of the Season

```
DO
  SEROUT TX, Baud, ["PC Z", CR]
  SERIN  RX, Baud, [char, DEC pos, DEC loopNum]
LOOP UNTIL (char = "S")
RETURN
```

The routine starts by setting the variable eePntr to the beginning of the songs list (in DATA statements). If the song number is greater than zero what we have to do is fast-forward through the list to the selected song title. This is a pretty simple matter: we simply read characters from EEPROM and count the number of zeros encountered. Notice that the loop that reads the characters from the DATA statements always increments the pointer after the READ instruction. This positions the pointer properly when the final zero is located.

Okay, now that our EEPROM pointer is set we can start the command by sending "PC F /" to the uMP3. We'll follow that with a loop that reads and sends the characters in the song title. Finally, we append ".MP3" and activate the uMP3 with a CR. Pretty simple, isn't it? And remember that if we use a member of the BS2p family, we could put the songs list in another program slot and use the STORE instruction to point to that slot – this would let us have a very long list of songs. Using this strategy is great for multilingual applications where each slot represents a different language – the program just needs to point to the slot that has the language of choice.

One of the features of the uMP3 that's useful for this application is its ability to report status while a song is playing. We can get status from the uMP3 by sending "PC Z." This will give us a status character ("P" for playing, "S" for stopped, "D" for paused), the current position (in seconds), and the loop counter for the file. Note that the position and loop counter are returned as text, so we use the DEC modifier to convert the values on-the-fly. This is another reason we left the uMP3 player at 9600 baud – using numeric conversion modifiers (DEC, BIN, HEX) in SERIN doesn't work well above 9600 baud (because of the inter-byte processing required by the modifier).

A DO-LOOP structure is used to send the "PC Z" command and then wait for the status character, position, and loop count. In other applications we might use the position value for prop synchronization, but we'll ignore that in this program. Once the status character changes from "P" to "S" we can return to the main part of the program and play another song.

And that's about that – a fairly simple program that lets us play full-blown MP3 files with a cool little player module. For those that are into prop building (and the faux candles program in October proved there are a bunch of you) this opens up a lot of neat possibilities. The program we just worked with plays continuously, but we would easily add a trigger input,

perhaps playing a random tune when someone rings the doorbell. That would certainly be more cheerful than the ubiquitous “*ding-dong*,” wouldn’t it?

MP3s Made Easy

If you’re new to audio projects and don’t have any tools for creating MP3s, you’re in luck as a great one is available at no charge: Audacity. This cool bit of freeware lets you edit audio and export to a variety of standard formats. Audacity does not have the ability to create MP3s built in, but it does support an MP3 encoder DLL (called LAME), and that’s available on the Audacity site as well. I’ve used Audacity and LAME to create MP3s for my uMP3 player, and even ring tones for my cell phone.

A particularly useful feature of Audacity is the ability to record from your system’s audio mixer. I’ve used this feature to convert MIDI files (available by the truckload on the Internet) to MP3s that I can load into the uMP3. To do this you set Audacity’s record input to Mixer, and then load the MIDI file into your normal media player (Audacity can open MIDI files, but not play or convert them directly). Once the MIDI file is loaded into your media player, click the Record button in Audacity, and then Start on the media player. Audacity will capture the audio stream and record it as a new track. When the MIDI file is done playing, click on Stop in Audacity. It’s a pretty simple matter to trim the leading and trailing blank space before exporting the audio as an MP3. I’ve used this same technique to pull songs from a CD and sound FX from a DVD.

Finally, let me remind you that just because you can “rip” audio from a CD or DVD that you purchased, that doesn’t mean you can play the file publicly – to do so without getting clearance from the publisher is illegal, so please don’t do it. If you do want to build a prop or display that uses commercial audio, contact the publisher and do the right thing (i.e., pay the license fee). The artists of the world – starving or otherwise (and including me, an actor) – will thank you for your support.

Well, that’s it for this month – and this year. Please accept my very best wishes for you and yours this holiday season, and let’s hope that we all have a very a happy new year. Until 2006, Happy Stamping!

Additional Resources:

Rogue Robotics

www.roguerobotics.com

Audacity Audio Editor

audacity.sourceforge.net

Vern Graner

www.spiderspreyground.com/howto

```

' SerialuMP3-looped-bs2.bs2
' {$STAMP BS2}

MP3R   CON 5      ' uMP3: set the pin where the uMP3 "R" pin is connected
MP3T   CON 4      ' uMP3: set the pin where the uMP3 "T" pin is connected

N2400T CON 396
N9600T CON 240    ' uMP3: Set the 9600 8 bit no parity TRUE

serdata VAR Byte
playcount VAR Word
serstr VAR Byte(6)
loop VAR Byte

statusstr VAR Byte(5)
statusstr(0) = "S"
statusstr(1) = " "
statusstr(2) = "0"
statusstr(3) = " "
statusstr(4) = "0"

playcount = 0

top_loop:
playcount = playcount + 1

SEROUT MP3R,N2400T,[CR]

SERIN MP3T, N2400T, 2000, no_resp1, [serdata]
DEBUG serdata,CR
GOTO output_data

no_resp1:
DEBUG "No Response from initial CR",CR
DEBUG "Playcount: ",DEC5 playcount,CR
STOP

output_data:
DEBUG "Sending Play command",CR
SEROUT MP3R,N2400T, ["PC F /B0000.MP3",CR]

SERIN MP3T, N2400T, 2000, no_resp2, [serdata]
DEBUG serdata,CR
DEBUG "Playcount: ",DEC5 playcount,CR

PAUSE 200

check_status:
SEROUT MP3R,N2400T, ["PC Z",CR]
'SERIN MP3T,N9600T, [STR serstr\6]

```

Column #128: Sounds of the Season

```
HIGH 15

SERIN MP3T, N2400T, 2000, no_status, [STR serstr\6]

LOW 15

DEBUG "GOT: ", STR serstr\4, CR
DEBUG "rx:"
FOR loop=0 TO 5
  DEBUG " ", HEX2 serstr(loop)
NEXT
DEBUG CR
'DEBUG "rx: ",STR serstr,CR

IF serstr(0) = "S" THEN top_loop

PAUSE 100

GOTO check_status

no_status:
DEBUG "Not Ready",CR
PAUSE 1000
GOTO check_status

'STOP

no_resp2:
DEBUG "Timeout error"
DEBUG "Playcount: ",DEC5 playcount,CR
STOP
```

```

'{$STAMP BS2}
'{$PBASIC 2.5}

' *****
' *          Rogue Robotics uMP3 Basic Stamp Test System          *
' *          By Vern Graner SSE, Texas Information Services        *
' *****
' * Code demonstrates the uMP3 audio playback feature set        *
' * Pin Configuration:                                           *
' * Pin 0- No Connection                                          *
' * Pin 1- N.O. pushbutton to V+ (pulled low)                    *
' * Pin 2- N.O. pushbutton to GND (pulled high)                  *
' * Pin 3- N.O. pushbutton to GND (pulled high)                  *
' * Pin 4- uMP3 T                                                *
' * Pin 5- uMP3 R                                                *
' *****
' *Revision Info: V1.0 (VLG) 7-6-2005                             *
' *****

' *****
' * Constant definitions *
' *****
MP3R   CON 5      ' uMP3: set the pin where the uMP3 "R" pin is connected
MP3T   CON 4      ' uMP3: set the pin where the uMP3 "T" pin is connected
N2400T CON 396
N9600T CON 84    ' uMP3: Set the 9600 8 bit no parity TRUE
BUT1   CON 1      ' N.O. Momentary Push Button on PIN1
BUT2   CON 2      ' N.O. Momentary Push Button on PIN2
BUT3   CON 3      ' N.O. Momentary Push Button on PIN3
BUT1ON CON 0
BUT2ON CON 0
BUT3ON CON 0

UMP3BPS CON N2400T

' *****
' * Variable definitions *
' *****
serdata   VAR Byte      ' 255 characters set aside for input from the uMP3
btnWrk    VAR Byte(3)   ' Workspace for BUTTON
Volume    VAR Byte      ' Volume control variable 0-255, 0= loudest
TriggerValue VAR Byte   ' sensors results stored here

' *****
' * Main Program Begin *
' *****

SEROUT MP3R,UMP3BPS,[CR] 'Send an INIT CR to the uMP3

```

Column #128: Sounds of the Season

```
GOSUB ReadMP3           'Check for response from uMP3
DEBUG ">CR",CR         'Echo the same command to DEBUG

GOSUB uVolReset        'Reset the uMP3 volume to FULL

GOSUB USTOP           'Tell the unit to STOP in case it's playing

LoopPoint:

' *****
' * Wait for START button *
' *****
DEBUG "Ready to play",CR
waitbut1:
  BUTTON BUT1, BUT1ON, 255, 250, btnWrk, 0, waitbut1
But1Lift:
  IF IN1 = 0 THEN But1Lift

SEROUT MP3R,UMP3BPS,["PC F /B0000.mp3",CR] 'Send a PLAY command
DEBUG ">PC F /B0000.mp3",CR             'Echo the command to DEBUG
GOSUB READMP3           'Check for response from uMP3

DEBUG CLS               'Clear the DEBUG screen

VOLLOOP:
  DEBUG HOME            'Move the DEBUG position to the top/left

  BUTTON BUT2, BUT2ON, 255, 250, btnWrk, 0, noVolUp ' Check if BUT2 pressed
  GOSUB uVolUp          ' if so, increment the volume
But2Wait:
  IF IN2 = 0 THEN But2Wait ' then wait for button release
noVolUp:
  BUTTON BUT3, BUT3ON, 255, 250, btnWrk, 0, noVolDown ' Check BUT3 pressed?
  GOSUB uVolDown      ' decrement volume
But3Wait:
  IF IN3 = 0 THEN But3Wait ' then wait until released
noVolDown:
  BUTTON BUT1, BUT1ON, 255, 250, btnWrk, 0, noPause ' Check BUT1 pressed?
  GOSUB UPAUSE      ' pause the player
But1Wait:
  IF IN1 = 0 THEN But1Wait ' then wait for release
noPause:
  SEROUT MP3R,UMP3BPS,["PC Z",CR] ' Check status of uMP3
  SERIN MP3T, UMP3BPS, 1000, noData, [serdata] ' Get the response
  DEBUG serdata,CR
  IF serdata = "S" THEN LoopPoint ' If finished playing file, jump to start
noData:
  PAUSE 100
  GOTO VOLLOOP ' otherwise, kill some time
```

```

' and check the buttons again

' *****
' * Subroutines *
' *****
uStop:                                'Stop the player
  DEBUG "PC S",CR
  SEROUT MP3R,UMP3BPS,["PC S",CR]
  GOSUB READMP3
  RETURN

uPause:                                'Toggle player pause
  DEBUG "PC P",CR
  SEROUT MP3R,UMP3BPS,["PC P",CR]
  GOSUB READMP3
  RETURN

uVolUp:                                'Increment the volume
  IF VOLUME > 7 THEN VOLUME=VOLUME-8
  DEBUG "PC V",DEC VOLUME,CR
  SEROUT MP3R,UMP3BPS,["PC V ",DEC Volume,CR]
  GOSUB READMP3
  RETURN

uVolDown:                              'Decrement the volume
  IF VOLUME < 248 THEN VOLUME=VOLUME+8
  DEBUG "PC V",DEC VOLUME,CR
  SEROUT MP3R,UMP3BPS,["PC V ",DEC Volume,CR]
  GOSUB READMP3
  RETURN

uVolReset:                              'Reset the volume to "full"
VOLUME=0
  DEBUG "PC V",DEC VOLUME, CR
  SEROUT MP3R,UMP3BPS,["PC V ",DEC Volume,CR]
  GOSUB READMP3
  RETURN

ReadMP3:                                'Read command response from uMP3 unit
  SERIN MP3T, UMP3BPS, 2000, No_Resp1, [serdata]
  DEBUG serdata,CR
  RETURN

No_Resp1:                               'Tell user no reply from uMP3
  DEBUG "Timeout: No response from uMP3!",CR
  RETURN

```

Column #128: Sounds of the Season

```
' =====
'
' File..... uMP3-Music-Player.BS1
' Purpose.... Random song player
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started.... 17 OCT 2005
' Updated.... 24 OCT 2005
'
'   {$STAMP BS1}
'   {$PBASIC 1.0}
' =====

' -----[ Program Description ]-----
'
' This program requires uMP3 firmware version 110.11.
' -- set baud to 2400 (ST D)
' -- set prompt delay to 5 milliseconds (ST R)

' -----[ Revision History ]-----

' -----[ I/O Definitions ]-----

SYMBOL Trigger          = PIN6
SYMBOL TX                = 5
SYMBOL RX                = 4

' -----[ Constants ]-----

SYMBOL Yes              = 0
SYMBOL No               = 1

SYMBOL Baud             = OT2400

SYMBOL Mp3Count         = 6           ' files on uMP3

' -----[ Variables ]-----

SYMBOL markers          = B0           ' song has played
SYMBOL tag              = B1
SYMBOL char             = B2           ' character to/from uMP3
SYMBOL theMP3           = B3           ' file# to play
SYMBOL eePntr           = B4           ' eeprom pointer
SYMBOL lottery          = W3           ' random number
```

```

' -----[ EEPROM Data ]-----
' Song list contains name of file only, and must be followed by zero
MP3s:
  EEPROM ("Frosty the Snowman", 0)
  EEPROM ("Jingle Bells", 0)
  EEPROM ("Joy to the World", 0)
  EEPROM ("Let It Snow", 0)
  EEPROM ("Little Drummer Boy", 0)
  EEPROM ("Silent Night", 0)

' -----[ Initialization ]-----
Reset:
  PAUSE 2000                                ' let uMP3 start

Prod_uMP3:
  SEROUT TX, Baud, (13)                     ' send CR
  SERIN  RX, Baud, char                      ' get response
  IF char <> ">" THEN Prod_uMP3              ' wait for ">"

  lottery = 1025                             ' seed random value
  markers = %11000000                        ' reset markers

' -----[ Program Code ]-----
Main:
  RANDOM lottery                             ' tumble random value
  IF Trigger = No THEN Main                  ' wait for trigger

  theMP3 = lottery // Mp3Count               ' create file#, 0..N
  LOOKUP theMP3, ($01, $02, $04, $08), tag  ' create tag bit
  char = markers & tag                       ' check markers fo played
  IF char > 0 THEN Main                      ' if played, try again
  markers = markers | tag                    ' mark played
  GOSUB Play_MP3                             ' play it
  IF markers < %11111111 THEN Main          ' all played?
  markers = %11000000                       ' yes, reset markers
  GOTO Main

' -----[ Subroutines ]-----
' Put file # to play in "theMP3"
' -- "theMP3" will be destroyed by subroutine
Play_MP3:

```

Column #128: Sounds of the Season

```
eePntr = 0                                ' reset pointer
IF theMP3 = 0 THEN Send_Cmd                ' at position, play it

Fast_Fwd:                                 ' move to selected title
  READ eePntr, char                        ' retrieve a character
  eePntr = eePntr + 1                      ' advance pointer
  IF char <> 0 THEN Fast_Fwd                ' if not 0, try again
  theMP3 = theMP3 - 1                      ' else, decrement file#
  IF theMP3 > 0 THEN Fast_Fwd              ' if > 0, not at titel yet

Send_Cmd:                                  ' start play command
  SEROUT TX, Baud, ("PC F /")

Send_Name:                                 ' send file title
  READ eePntr, char
  eePntr = eePntr + 1
  IF char = 0 THEN Finish_Cmd
  SEROUT TX, Baud, (char)
  GOTO Send_Name

Finish_Cmd:                                ' send extention + CR
  SEROUT TX, Baud, (".MP3", 13)

Wait_For_Stop:                             ' let song finish
  PAUSE 100
  SEROUT TX, Baud, ("PC Z", 13)
  SERIN RX, Baud, char
  IF char <> "S" THEN Wait_For_Stop
  RETURN
```

```

' =====
'
' File..... uMP3-Music-Player.BS2
' Purpose.... Random song player
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started.... 17 OCT 2005
' Updated.... 21 OCT 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
' =====

' ----[ Program Description ]-----
'
' Randomly plays files from a Rogue Robotics uMP3 player. All songs on
' list will be played before any repeats occur.
'
' Note: Requires uMP3 firmware 110.11 or higher

' ----[ Revision History ]-----

' ----[ I/O Definitions ]-----

TX          PIN    15          ' to uMP3.RX pin
RX          PIN    14          ' to uMP3.TX pin

' ----[ Constants ]-----

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON    84
#CASE BS2SX, BS2P
  T9600      CON    240
#CASE BS2PX
  T9600      CON    396
#ENDSELECT

SevenBit    CON    $2000
Inverted    CON    $4000
Open        CON    $8000
Baud        CON    Open | T9600

NumSongs    CON    16          ' songs on uMP3 player
SongDelay   CON    0000       ' 2 secs between songs

```

Column #128: Sounds of the Season

```

#DEFINE TestMode = 0                                ' 1 = song title to DEBUG

' -----[ Variables ]-----
idx          VAR      Byte      ' loop counter
theSong      VAR      Byte      ' song number to play
lastSong     VAR      Byte      ' last played
zCount      VAR      Byte      ' for fast-forward routine
eePtr       VAR      Word       ' eeprom pointer
char         VAR      Byte
pos          VAR      Byte      ' song position
loopNum      VAR      Byte      ' song loop #
markers     VAR      Word       ' played markers
lottery     VAR      Word       ' random value

' -----[ EEPROM Data ]-----
MP3s        DATA    "Away in a Manger", 0
             DATA    "Choir of Bells", 0
             DATA    "Dreidel", 0
             DATA    "Frosty the Snowman", 0
             DATA    "Grandma Got Ran Over by a Reindeer", 0
             DATA    "Hark the Harold Angels Sing", 0
             DATA    "Jingle Bells", 0
             DATA    "Joy to the World", 0
             DATA    "Little Drummer Boy", 0
             DATA    "Let It Snow", 0
             DATA    "The First Noel", 0
             DATA    "O Holy Night", 0
             DATA    "Rudolf the Red Nose Reindeer", 0
             DATA    "Santa Claus Is Coming to Town", 0
             DATA    "Silent Night", 0
             DATA    "Winter Wonderland", 0

' -----[ Initialization ]-----
Reset:
  PAUSE 2000                                        ' let uMP3 start-up

  #IF TestMode = 0 #THEN
    SEROUT TX, Baud, [CR]                          ' prod
    SERIN  RX, Baud, 250, Reset, [WAIT(">")]      ' verify presence
  #ENDIF

  lottery = 1225                                    ' seed random generator
  GOSUB Reset_Markers                              ' clear available songs

```

```

' -----[ Program Code ]-----
Main:
  RANDOM lottery                                ' toss random generator
  theSong = lottery // NumSongs                 ' get song #
  IF (markers.LOWBIT(theSong) = 1) THEN Main    ' retry if played
  IF (theSong = lastSong) THEN Main            ' no repeat on next cycle

Play_The_Song:
  markers.LOWBIT(theSong) = 1                   ' mark current song
  lastSong = theSong                           ' mark as last played
  GOSUB Play_MP3                               ' play it
  IF (markers = $FFFF) THEN                    ' all songs played?
    GOSUB Reset_Markers                        ' yes, reset
  ENDIF
  PAUSE SongDelay                             ' break between songs
  GOTO Main

' -----[ Subroutines ]-----

Play_MP3:
  eePntr = MP3s                                ' start at beginnig
  IF (theSong > 0) THEN                         ' fast-fwd to song
    zCount = 0                                  ' reset counter
    DO
      READ eePntr, char                         ' read char
      eePntr = eePntr + 1                       ' update pointer
      IF (char = 0) THEN                        ' 0?
        zCount = zCount + 1                    ' yes, update count
      ENDIF
    LOOP UNTIL (zCount = theSong)
  ENDIF

#IF (TestMode = 0) #THEN
  SEROUT TX, Baud, ["PC F /"]                  ' play file in root
  DO                                            ' send file name
    READ eePntr, char
    eePntr = eePntr + 1
    IF (char = 0) THEN EXIT
    SEROUT TX, Baud, [char]
  LOOP
  SEROUT TX, Baud, [".MP3", CR]                ' send extension
  PAUSE 500
  DO
    SEROUT TX, Baud, ["PC Z", CR]              ' get status
    SERIN RX, Baud, [char, DEC pos, DEC loopNum]
  LOOP UNTIL (char = "S")
#ELSE

```

Column #128: Sounds of the Season

```
DEBUG DEC2 theSong, ": "  
DO  
  READ eePtr, char  
  eePtr = eePtr + 1  
  IF (char = 0) THEN EXIT  
  DEBUG char  
LOOP  
  DEBUG CR  
#ENDIF  
RETURN  
  
' -----  
  
' Resets the markers variable based on the number of declared songs  
' -- up to 16  
  
Reset_Markers:  
  markers = $FFFF          ' mark all played  
  FOR idx = 1 TO NumSongs  
    markers = markers << 1  ' clear available songs  
  NEXT  
RETURN
```