Macintosh Technical Notes

#19: How To Produce Continuous Sound Without "Clicking"

See also:	The Sound Driver
	The File Manager
	The Vertical Retrace Manager

Written by: Ginger Jernigan 4/18/85

This technote describes how to use the Sound Driver to produce continuous sound without the "click" that the driver produces when it starts a sound.

If you have tried using the Sound Driver to produce sound you probably noticed that one StartSound call produces sound for a comparatively short time. So, to produce longer sounds the obvious thing to do is to make consecutive calls to StartSound. This, however, produces an annoying "click" everytime StartSound is executed. This technote describes how to use the Sound Driver to produce continuous sound without the "click", and provides a little more background information about how the Sound Driver works.

How Does The Sound Driver *REALLY* Work

In order to better understand how to modify the output of the Sound Driver, we need to know more about how sound is produced. For this discussion we will use the free form synthesizer as our example. The process used for the other synthesizers is very similar.

To produce sound, the first thing we do is set up the sound buffer. For the free form synthesizer this consists of the the **FFSynthRec**, which includes the following fields:

Mode: The mode for the free form synthesizer is FFMode.
Count: Count is a fixed point number used for sampling the wave form.
Waveform: This is the array of values that define the wave form.

To set up the buffer, use the example in the Sound Driver section of Inside Macintosh, page 11. Once you set up your buffer you then call StartSound. StartSound will in turn set up a parameter block where **iobuffer** contains your buffer (in our case FFSynthRec); the **iorefnum** is -4; **iocompletion** points to your completion routine; and **ioreqcount** is equal to the size of your buffer. Once the parameter block is set up it then calls the Write trap. (If you are using assembly language or a third party development system that doesn't support StartSound, these are the steps you would take to start the sound yourself.)

When the Sound Driver gets the parameter block it reads the mode from the buffer to find out which synthesizer to use to interpret the buffer. It then installs itself into the Vertical Retrace queue, samples the waveform according to the **count** given, and then loads the specified bytes into the high (even) bytes of the sound/disk speed buffer. As it indexes through the waveform array using **count** it increments the **ioactcount** field in the parameter block and checks it against the **ioreqcount** to make sure it doesn't exceed the wave form array. If the

ioactcount and **ioreqcount** aren't equal on the next vertical retrace, then it fills the buffer again. If they are equal and the buffer isn't filled, the rest of the buffer is filled with zeros.

How To Avoid The Embarassing "Click"

The click happens after the _Write call occurs, while the driver is setting up for the production of sound. Therefore, the trick is to make only one call to start the sound, then change the information in the paramater block while the Sound Driver is still producing sound.

The first step is to take care of the parameter block setup and the PBWrite (_Write) call yourself, instead of letting StartSound do it for you. The reason is that you need to have easy access to the parameter block in order to change the **ioactcount** field. The Sound Driver uses **ioactcount** to keep track of where it is in the sound buffer. So, when we want the sound to start over again, instead of calling _Write again we fool the Sound Driver into thinking it is actually starting at the beginning of the buffer by setting **ioactcount** to 0.

To set up the sound buffer and the parameter block here's an example. Again, we are using free form sound.

```
{$U-}
{$R-}
{$M+}
{$L+}
Program Soundtest;

{$L-}
uses
 {$U Obj/MemTypes  } MemTypes,
 {$U Obj/QuickDraw } QuickDraw,
 {$U Obj/OSIntf           } OSIntf,
 {$U Obj/ToolIntf  } ToolIntf,
 {$U Obj/PackIntf  } PackIntf,
 {$U Obj/SaneLib          } SANE;     { Just in case we want to do any SANE stuff }
{$L+}

VAR myHandle :     handle;
    MyFFPtr :      FFSynthPtr;
    Buffsize :     integer;
    i,j :          integer;
    Parmblock :    parmblkptr;
    err :          oserr;

BEGIN
 FlushEvents(everyevent, 0);

 { First we set up the sound buffer information }
 Buffsize := 30000;                   { The total size of the buffer is 30000 }
 myHandle := NewHandle(buffsize); { Make a new handle of that size }
```

```
   HLock(myhandle);                 { Lock it. We don't want it to move on us }
   MyFFPtr := FFSynthPtr( MyHandle^ );  { Force a type coercion on the contents of the
                                      handle to an FFSynthPtr }
   MyFFPtr^.mode := FFMode;         { Set the mode to free form synthesizer }
   MyFFPtr^.Count := $00010000;         { Set the count to 1 - remember this is in
                                      fixed-point notation }
     j := 0;                            { The waveform is zero based }
   WHILE j<=buffsize-7 DO BEGIN         { The -6 is for mode and count, and -1 is because
                                          we're zero based }
      FOR i := 0 to 255 DO BEGIN
         MyFFPtr^.wavebytes[j] := i;     { Stuff the waveform with values }
         j := j+1;                       { Increment wave index }
      END;
   END;
   SetSoundVol(6);                      { Set the sound volume }

   { Then we allocate the parameter block }
   ParmBlock := ParmBlkPtr( NewPtr( SizeOf( ParamBlockRec )));
   WITH Parmblock^ DO BEGIN         { And set up the parameter block with ... }
      iocompletion := nil;              { No completion routine }
      iorefnum := -4;                   { Refnum is -4 as per Inside Macintosh }
      iobuffer := ptr( MyFFPtr );       { The buffer is set to point to my sound buffer }
      ioreqcount := buffsize;           {  The  required  count  is  set  to  the  size  of  the
                                          buffer }
   END;
   err := PBWrite(Parmblock,true);      { OK - now write it asynchronously}
```

After this point the Sound Driver starts filling the buffer and sound is produced as discussed above. The sound is produced asynchronously in order to allow our program to keep on running during the sound generation. To restart the sound you want to change the **ioactcount** after the buffer has been filled (and after the Vertical Retrace task has incremented **ioactcount**) and before the last pass at filling the sound/disk speed buffer.

The trick is to make sure the change to **ioactcount** happens at the right time. If you change it too early, then the Vertical Retrace task will increment it after you have changed it, thus resetting it to where it was before. If you change it too late it will finish the waveform and stop the sound whether you want it too or not. The best way to take care of this timing problem is to install your own Vertical Retrace task in the queue before or after the sound task. The task would look something like this:

```
      IF ParmBlock^.ioactcount >= 29960 THEN BEGIN
       {the 29960 is from the formula (count*370)*Trunc(waveform/(count*370)). This should
           be before the last pass through the waveform. }
         ParmBlock^.ioactcount := 0;
         MyFFPtr^.count := MyFFPtr^.count + $00000500;   { Let's increment the count for
                                                          fun}
      END;
```

If the routine is in the vertical retrace queue then you avoid the "critical section". This is when the sound code and your code are trying to change the same value at the same time. You want to avoid this. Really.

Changing The Sound Information

Notice that we also changed the count parameter in the buffer. Since **ioactcount** is now 0 it will begin at the *beginning* of the waveform the next time is needs to stuff the sound/disk speed buffer. Since we have changed the count(sampling rate), it will use the new value to sample the waveform on the next pass.

If you need to change the values in the waveform, there are two methods. The first is to change the current waveform. To do this, you would, for example, check to see if **ioactcount** is half of **ioreqcount**, and start stuffing at the beginning of the buffer. Be careful, though, because you can stuff the waveform faster than the values are sampled. Then, before the last pass at filling the sound/disk speed buffer change the **ioactcount** so it will start at the beginning of your new wave form.

The second method is to fill a *different* waveform while you are producing sound and when you change **ioactcount**, you assign the second buffer into **iobuffer**. Make sure the mode and count are correct in the second buffer.