C99 Sound List Instructions.

.

This product was made in response to a request from Vern Jensen, of Pasadena, CA. He asked whether it were possible to use the "sound list" process that's available for Assembly programmers from his C99 code. What we've done in this little Assembly module is to make a very elegant way for C99 programs to incorporate and use Sound Lists. This product is Public Domain software, written by Bruce Harrison. It may be shared freely with anyone without compensation to the author..

.

Including Sound Lists.

.

The simplest way to include sound lists is just to create them as D/V 80 files, putting in a line that says #asm at the start of the file, and a line that says #endasm at the end of the file. At the end of your C99 source file, [after the closing bracket of main()] put in a line that says #include "DSKx.SOUNDLST", where x is the drive number where the list file will be found. You may include as many such lists as you like. Each such list should have a unique 6-character or less name, which does not duplicate any label names used in your C99 code. There are several sample lists on the disk, including two versions of the "Jeopardy" theme, several chime sounds, a crash sound, and part of one movement of a Vivaldi Sonata..

.

The second way to include sound lists is to DEF the labels in the list file, then Assemble the list file separately into an object file, include the names of the lists as extern statements in your C99 code, then load the lists' object files along with the object file made from your C99 code. In this method, you don't put in the #asm and #endasm lines in the list source file. If you use this method, be sure that you write the extern as if the list were a function. [e.g. extern jeothm();] If you omit the (), the C99 compiler will not handle the references correctly..

.

The Sound List Concept.

.

The idea of Sound Lists is quite simple. A sound list is a bunch of one-byte instructions that are used by the Interrupt Handler in the TI's ROM. These instructions cause bytes to be sent to the sound chip, and the code in the ROM then provides timing of the duration for each note. For the sound to keep working, the computer must be executing LIMI 2 and LIMI 0 instructions now and then. That usually is the case when your C99 program is executing a "key loop" of some kind, or awaiting input from a joystick, etc. The data from the sound list has to be loaded into VDP Ram memory or GROM. Since C99 doesn't have any capability to write things into GROM, that's not an option. Once the sound list data is written to VDP Ram, it can just stay there and be activated selectively by a little Assembly code. Under normal conditions, a properly constructed Sound List in VDP Ram will be played through one time and then the sound will stop. For example, the BEEP and HONK sounds issued by the Basic languages use short sound lists that are permanently stored in GROM. If there's room, any number of sound lists can be loaded into VDP Ram, left in place there, and activated whenever needed..

Structure of Sound Lists.

.

A sound list is made of Assembly source statements. Usually each line of such a list is a single "note" statement, which may specify notes and volumes for each of the four sound generators in the TI sound chip. Our normal practice in a sound list is to start with a "silence" line, to cancel out anything that the generators were doing when our sound list was activated. That first line looks like this:.

.

```
LISTNM BYTE 4,>9F,>BF,>DF,>FF,1.
```

.

The label LISTNM is of course the name of this particular sound list. It must start in the leftmost column. There are three parts to this line, and each must be correctly done. The word BYTE must have at least one space to its left and one space to its right. The number after BYTE is the number of bytes that will be sent to the sound generator itself. In this case, there are four such bytes, each expressed in hex notation.

The >9F means silence for generator 1. >BF means silence for generator 2, >DF silences gen. 3, and >FF silences the noise generator, number 4. The final number, 1 in this case, is NOT sent to the sound chip, but indicates the duration of this particular "note" in 60ths of a second. Thus our sound list LISTNM will begin with 1/60th second of silence. In the lines after that, we send frequencies to the three main generators, noise numbers to the noise generator, and volume bytes to all four generators, and durations for each note group to be played..

.

Notes or frequencies for the three main generators require two bytes. The left nybble of the first "note" byte specifies which generator is being addressed. If that left nybble is 8, the note affects generator 1. If it's A, that note goes to generator 2, and if C, generator 3. Specification of a noise "note" requires only one byte, and its left nybble must be E..

.

Unlike the CALL SOUND in Basic, a note once started in a sound list, given a non-silent volume, will continue to play unless turned off by sending a silence volume byte. Volume bytes always start with an odd Hex number in the left nybble, (9,B,D,or F) and the second nybble ranging from 0 through F. If that right nybble is 0, the sound on that generator will be at its maximum volume. Each number above that, up through E, will reduce the volume by 2 decibels. Thus a volume byte of >9E will set the volume of generator 1 to 28 decibels below maximum, or about 1/630th of the maximum sound power..

.

For more about sound lists, look in the E/A Manual's Sound chapter. You amy also find it helpful to look at the structure of our sound lists that are included on this disk..

.

IMPORTANT CAUTION.

.

Each sound list that you wish to use can have instructions for any or all of the four generators, but the LAST line of the sound list MUST be like this:.

.

```
LSTSND BYTE 4,>9F,>BF,>DF,>FF,0.
```

.

This very important line need not have a label, as we've shown here, but MUST have that 0 byte at the end. That tells our sound list loading code where the end of your list is, and also tells the code in ROM when to stop playing. Those four bytes in the middle shut off all four sound generators by setting them to "infinite" attenuation. The 4 in the first byte indicates how many bytes are to be sent to the sound generator chip, and the last byte, which normally indicates the duration for a sound in 60ths of a second, signals by being zero that this sound list is at an end. If you make this line without the label, then the word BYTE must be indented at least one column from the left edge..

.

Our Own Method.

.

The object file C99SOUND/O provides all the Assembly code to load sound lists into VDP Ram, to activate them when needed, and to perform some tricks you didn't think were possible, but which we've made possible. Assuming you've #included one or more sound lists, you use our setsou routine to load them into VDP Ram. A call to setsou looks like this:.

.

```
setsou(0,fuldat,jeothm,onedat,vivdat);.
```

.

The first parameter MUST be zero (0). The other parameters are the label names of the sound lists to be loaded, and there may be up to 10 such names included. The routine will put all of the data from those lists into VDP Ram. None of them will be playing, but they'll all be available for play from that point on in the program..

.

The next order of business is to start some sound list playing. To do so, you first jot down the order in which you included these lists in your setsou call. In this example, the list fuldat is number 1, jeothm is number 2, onedat is number 3, and vivdat is number 4. Let's suppose that you want vivdat to start playing and keep playing over and over until you tell the sound processor to do something else. For this you use the

stsou function, which has two parameters. The first parameter is the number of the sound list to play. (4 in this case) The second parameter indicates which optional way of playing you choose. The four options are:.

.

1. Play continuously with Da Capo repeat..
2. Play once only, then stop..
3. Interrupt the continuous, play once, then return..
4. Play in "foreground", just once through..

.

Except for number 4, each of these options operates the sound "on background", which means that your C99 code can go on and do other things while the music continues to play, provided only that there's a cycle of LIMI instructions someplace in what you're doing (like for example a GetKey) that turns on the interrupt handler now and then. Option 4 causes the sound list to "take over" the computer, so that nothing else will happen until the chosen sound is finished. To start our vivdat sound list playing continuously, we do this:.

.

```
stsou(4,1);.
```

.

The Vivaldi sonata will play. When it finishes its sixteen bars, a User Interrupt included in our Assembly part will cause it to start playing "Da Capo", or "from the top". In other words, it will start over after it's finished. (Da Capo is Italian for "from the head", and is used in musical terminology to mean, "go back to the beginning of the piece.) So long as you don't interfere with it, the vivdat sound list will literally play over and over forever. Now suppose that some event happens in your program, such as for example the Y key is pressed, and you want a single chime to sound, interrupting the Vivaldi just for a brief interval, but then you want the Vivaldi to continue after the chime. That's what Option 3 is designed for. You'd do this from C99:.

.

```
stsou(3,3);.
```

.

When that executes, the Vivaldi will stop instantly, one chime will sound, then the Vivaldi will resume at whatever is the next note in that sound list. It will then be back to its original state, doing a Da Capo every time it finishes..

.

Now suppose that you wanted the full set of chimes to interrupt the Vivaldi, and then stop the Vivaldi. That's what Option 2 does. The list fuldat is number 1, so to use that to play and terminate the music, you'd do setsou(1,2);. The chimes will sound, then all sound will be shut down..

.

In all of these cases, the computer continues to function pretty much as if the sound were not happening. That is, you can accept keystrokes, write to the screen, move sprites around with a joystick, and so on while the music plays. Let's suppose, though, that you don't want the program to do anything until a particular sound is over. Easy! Use Option 4. Let's say we want the chimes to sound, and all other things stop until the chimes finish. Simply use stsou(1,4);. Sprites that are in motion will continue moving, but nothing else will happen until the sound stops. Unlike options 1 through 3, using option 4 allows the sound to take over control of the computer. We're not sure whether anybody will want that to happen, but the option is there if you need it..

.

But What If I ... .

.

Our dear friend Vern Jensen always wants something else, and the sound list capability is no exception. Vern wanted to "have his cake and eat it" by having the ability to let the sound list control the pace of actions in the C99 program, while playing on background and not interfering with him unless he permits it. It's hard to describe what this means in words, but we'll try. Let's say you have a character on the screen, and you want that character to move over one spot for each note in the music, but at the same time you want to be reading the keyboard or joysticks, and having the program respond to those inputs as well. When Vern first mentioned this, the word "impossible" crossed our lips. But then, maybe if we ....

.

Yes, Vern, it can be done. We've included a little feature called PACEIT, by which you can have other things going on, and still have something happening only when a new note starts in that sound list. To use PACEIT, you'd first start a sound list playing with stsou, using option 1. Next, you'd set up for the pacing operation by setpac() (no parameters). After this, you enter a loop that may include checking keyboard and/or joysticks, writing to the screen, crunching numbers, or whatever, but somewhere in that loop you'd include this:.

.

```
status=paceit();.
if (status == 1) [do something] .
```

.

The return variable from paceit will be zero if the same note that was playing on the last call to paceit is still playing. It will be 1 if the sound list is on a new note since the last call. If the sound has ended, paceit will return 0. For purposes of paceit, a rest in the music counts the same as a note. Try out the demo PACED from E/A Option 5, and perhaps you'll see more clearly what this means. (see description below) There's one more handy function included in our C99SOUND object file. It's called testso. This function returns a 1 if a sound list is playing, 0 if not. You could use this with a return variable or just by itself in an if statement..

.

The Demos.

.

As always with our disks, we've included demo programs to illustrate the use of this software in actual C99 programs. Here, there are two such demos, called SOUND and PACED. Both are E/A Option-5 Program Files, so you can quickly and easily run them. The original C99 source for these is included as SOUND/C and PACED/C. To be complete, we've also included the compiled forms, with their /S ending, and the object files, with their /O ending. Also on the disk are the original sound lists used in these demos..

.

The first demo is called SOUND. Run it from either E/A Option 5 or from Funelweb. When this starts, the "Jeopardy" theme will be playing on background, while the computer awaits a keystroke from you. This is a long version of the theme, with three repeats in different keys. It's been set up with playing option 1, so it will continue all day. When you press a key on the keyboard, two chimes will sound activated with option 3, interrupting the theme for a short interval, then the theme will pick up at its next note. Left to itself, it will continue to repeat endlessly. Meanwhile, the computer is again cycling through a "key loop", and will do so until you press a key. When you do, the full chimes sound will start with option 2. Thus the chimes will play through once, then sound will stop. Pressing a key during or after the chimes will start the Vivaldi music playing. This uses option 1, so it will continue all day. Pressing a key while the Vivaldi plays will start another round of full chimes, but this time through option 4. Thus all other actions will stop until the chimes finish. After the chimes, a prompt will appear, accompanied by a crash sound effect. Pressing a key during or after the crash will get you to the exit/rerun prompt..

.

The second demo is called PACED. When this runs, there will be a "T" on the screen, starting at row 3, column 1. The "Jeopardy" theme will be playing continuously, and as it plays, the "T" will move over by one column each time a new note (or rest) plays. Thus we say that the movement of the "T" is paced by the music. It will proceed on across, moving down a row when it reaches the right side, and so on across and down the screen. If it reaches the bottom, it will return to the top of the screen at row 3, col 1, and continue from there. Just to prove that this is not tying up the computer, we have a call to GetKey within the loop. If you press a key other than <ENTER>, the program will display that key's ASCII value at the top of the screen, and will keep displaying it there until you release the key. This just proves that we can not only keep pace with the music, but also perform other functions while the "T" is moved along. To exit from this, press ENTER. The number 13 will appear at the top of the screen, and just under that the exit/rerun prompt. Take your choice..

.

The All Important END.

.

In any program where this sound list stuff is used, it's important that before exiting you include a call to the endsnd() routine. That needs no parameters, and returns no variables, but it shuts off all the sound generators and restores the original address number to the word in Ram Pad at >8370. Unless that gets done, certain other things won't work when you return to E/A control..

.

Overall Rules for Use.

.

Any number of sound lists up to and including ten can be loaded at one whack with setsou(). The first parameter in that call must be a 0. This allows our Assembly routine to figure out how many lists are included. Loading of the lists into VDP Ram is very quick, and should be done very early in the program. There should be room in VDP memory for a total of 10,199 bytes of sound lists. If characters above 127 are not being used in your program, then this number can be extended to 11,223 bytes..

.

The sound lists, once loaded, will remain in VDP and available for use until endsnd() is called. You should, if at all possible, load all the lists you'll need at the very beginning of your C99 program. If your program uses file access, you MUST call endsnd() before trying to open or access files. If sounds are to be used again after file access, you can repeat the setsnd() action after files are closed..

.

We hope you'll find this disk useful. Should you need help, contact the author either by mail or phone at:.

.

Bruce Harrison.
5705 40th Place.
Hyattsville MD 20781.
U.S.A..
Phone (301) 277-3467

.....