This document describes the 'Sound Environment File'. This file is always called sounds.env and must be in the same directory as all the source files for the sound module. The environment file tells various programs including the SPL compiler, the audition program and Spectrum about the sound module you are trying to create. These tools only search the current directory for a sounds.env, so they must be invoked in the directory with the environment file and all the list files and sound source files.

A few notes about the philosophy and terminology. Sound for games generally comes in two flavors: Sound Effects (SFX), and Music. A sound module is a sort of 'Neapolitan', or mix of these two basic flavors plus some additional data and machine dependant stuff. The idea is to make getting sound out to the programmers as simple as possible, so as few files as possible are used (usually one object file, a documented sound list and possibly a header file). A need to add new SFX and Music files in some reasonably elegant way led to the separate SFX and Music list files. Although it is possible to have a single file of both music and sound effects for a module (except for the Handy version) this is not really a good idea: there are some parts of the sound development environment currently under construction that I intend to use this distinction in. You can mix 'em, but I'll be grouchy if you do. Anyway. The list files are simply that: the filenames of the sound effects to be included in the sound module are listed in the SFX list file; music files are listed in the Music List File. As more and more sound modules were generated, and ever higher levels of grooviness were sought thereby; the need to have all the various parameters of the sound module (such as SFX lists, Music lists, music rings, etc) specified in some simple way that was guaranteed to remain constant until specifically changed and always included or used or whatever no matter how quickly the sound module was needed became clear (quite a sentence, don't you think?) Of this need was born the Sound Environment File. This file is supposed to let you specify all the interesting things about the sound module you are working on *just once*. Once the sound environment has been set to create Apple II sound modules with the SFX listed in the file Frobisher and the music listed in the file Goo and create a module named Frank, it will merrily do this automatically each time you build the module. Neat, huh? So, on to the specification of sound modules!

Sound modules for all SPL drivers are specified by an 'environment' file called sounds.env. This file tells the various tools and programs what driver is being targeted and which files are being used in the module. What follows is a list of the various options available. Words in boldface are the ones used to select a particular option. Note that programs reading the sound environment file are case sensitive and so all options must have correct case as well as all files within the Sound List and Ring List files.

Machine: specifies the target machine.
- **c64** —— the Commodore 64 SPL driver
- **IBM** —— IBM one-voice SPL and Spectrum drivers[a]
- **Apple** —— Apple SPL and Spectrum drivers[b]
- **Handy** —— the SPL Handy driver[c]
- **AdLib** —— the IBM AdLib SPL driver.

Tempo: for drivers with a default tempo setting.
- **Tempo <n>** —— sets the default tempo to <n>.
- **DfltTempo <n>** —— same but overridden by Tempo.[d]

Music Rings: lists of phrases to play in sequence.[e]
- **RingFile <name>** —— use rings from file <name>.

Sound List Files: Specify lists of SFX and Music.
- **SFXFile <name>** —— include SFX listed in <name>.
- **MusicFile <name>** —— include Music from <name>.

Output file name: specify name of sound module.
- **ModuleName <name>** —— set module name to <name>.

Long name of game or event.
- **Name <name>** —— set name displayed by audition program to <name>.

There are a number of obsolete flags that are ignored by all programs but still appear in some older sound modules. For example there was a time when the number of rings in a module had to be specified in the environment file; there is an obsolete flag NumRings that was used to specify this.

---

[a] Support of the IBM 3-voice SPL, 3-voice Spectrum, EPYX DAC, Tandy DAC and AdLib drivers will be integrated in some clever way. Currently the data format is under revision to enable support of all these different formats.

[b] An Apple Sample Driver is underway. Including sounds for this driver would be easy.

[c] The Handy SPL driver is a separate compiler that will disappear into the New Products Division Real Soon Now. For all intents and purposes, it has...

[d] The DfltTempo is designed to be set automatically by converter programs. A Tempo statement will always override a DfltTempo statement.

[e] The Handy driver is an exception to this: rings are created as music files and the Music Rings feature is used to set the default voices in a sound module.

The SPL compiler, aud program and spectrum recognize some filename extensions. The extensions are:

.spl —— SPL compiler source code
.src —— XASM-65 source code for Macro SPL
.bsf —— Binary image from SPL compiler.
.dnl —— Binary image from *downfile* or *upl* utilities.
.bfx —— Binary image from Spectrum SFX Editor.

To be added for ***Spectrum-3v:***

.1fx – single voice of a three voice sound effect.
.3fx – three voice IBM sound effect.

The most useful extensions are .spl, .bsf and .bfx and these should be used for all new code. These are standard and graven in hard wood (not really stone — yet).

Sound List Files

Sound list files specified by SFXFile and MusicFile simply list the files to be included in the output module. An example IBM SFXFile:

```
up.bfx
down.bfx
laser1.bfx
squeak.bfx
```

The module will contain the sound effects up.bfx, down.bfx, laser1.bfx and squeak.bfx in the order listed. Multiple sound list files may be used; sound files from each list file will be included in the order they appear in the sounds.env file. For example, if the environment file has the following lines to include sound list files

```
SFXFile     oldsfx
SFXFile     newsfx
MusicFile   theme
MusicFile   muslist
```

and the list files contain the sound source files

file: oldsfx

```
in.spl
out.spl
wind.spl
waves.spl
stone.spl
```

file: newsfx

```
hitonhed.spl
babyseal.spl
whap.spl
```

nose.spl
fin.spl

file: theme

guitar.spl
drums.spl
bass.spl
intro_g.spl
intro_d.spl
intro_b.spl

file: muslist

fanfare.spl
entre_1.spl
entre_2.spl
entre_3.spl

the resulting data-module would have the sounds in the order -

in.spl
out.spl
wind.spl
waves.spl
stone.spl
hitonhed.spl
babyseal.spl
whap.spl
nose.spl
fin.spl
guitar.spl
drums.spl
bass.spl
intro_g.spl
intro_d.spl
intro_b.spl
fanfare.spl
entre_1.spl
entre_2.spl
entre_3.spl

The order of sounds in a module will be the order of sounds in the list files with the first list file first, the second second, and so on.

Rings

Ring files are similar, but deal with a different thing. A 'ring' is a list of 1-voice music phrases to be played on a single voice; a ring may end in either the RingEnd or the RingLoop token. RingEnd tells the driver to stop playing sound and release the voice; RingLoop tells the driver to go back to the start of the ring and play it again in an infinite loop.

For example a song with three voices could be built from phrases using the 'ring' construct in the following way: voice 0 plays parts lead_1.spl, lead_2.spl, lead_1.spl and finally coda_v1.spl; voice 1 plays parts fill.spl, arp.spl, fill.spl and coda_v2.spl; and voice 2 plays parts bass_1.spl, bass_2.spl, bass_3.spl and coda_v3.spl and it looks like this in the ring file:

```
Theme_voice0                    — a unique label indicating the start of a ring.
        lead_1 lead_2 lead_1 coda_v1 RingEnd
Thmv1                           — ring labels can look like just about anything
        fill arp fill coda_v2 RingEnd
Thmv2
        bass_1 bass_2 bass_3 coda_v3 RingEnd
```

The Handy uses the ring file differently. Rings in HSPL are formed by using the GSJ calls within normal spl source files. The ring file in HSPL is used to select the musical parts to play as the 'default song'; the song the module is set up to play when loaded into a game.

Examples

A C64 sound environment file might look like this:

```
c64                                  — specify the machine
SFXFile      sfxlist                 — include sound effects first
MusicFile    Title                   — music for title screen
MusicFile    levels                  — music for levels of play[f]
RingFile     ringlist                — music ring specification file
Tempo        9                       — default tempo of 9
ModuleName   blaster.dnl             — name of the module
```

An IBM sound environment file:

```
IBM                                  — machine
SFXFile      list.sfx                — include sound effects first
MusicFile    muslist                 — music for the event
RingFile     rings                   — music rings to play...
ModuleName   onevoice.dnl            — specify module name
Name         Whammo Zap              — name of the event or game
```

An Apple II sfx list file: — further examples are shown above under Sound List Files.

```
hop.bfx
blat.bfx
step.bfx
zappo.bfx
```

A C64 music list file: — further examples are shown above under Sound List Files.

```
lead_1.spl
intro_lea.spl
bass.spl
intro.bas.spl
drums_1.spl
intro_dru.spl
lead_2.spl
drums_2.spl
```

An IBM ring file: — for more detail see above section Rings.

```
Title                                — Ring label, Title music
        Intro_1 Main_Thm Main_Thm Coda RingEnd
Street_dancer                        —Ring label, incidental music for game
        Street RingLoop
Bar_dancer                           —Ring label, incidental music for game
```

---

[f] The music is put in two files only as an example: all music could be put in one file just as all sfx can be but sometimes it's nice to split them into several files.

FourArms RingLoop

Implementation Guide

The various programs that look at the sound environment file do so through a function called *read_env_file()*. A good example of this function's use is in the options.c file of the SPL compiler. In the SPL compiler most of the environment settings are important, so the function is fairly full. At the top of the file is an enumerated type that starts with mC64, mIBM,... that is used in *read_env_file()*'s central switch statement. Also towards the top of the file is the array *cmd_names* which is scanned by the function *match_env()* to recognize commands when the environment file is being parsed. If flags or commands are added to the environment file it is important to make sure they can be properly ignored by all programs that don't use them. Right now, when *read_env_file()* finds something it doesn't recognize it ignores it. Every program must recognize the 'name' command because I wanted names with spaces in them so for that particular command I read to the end of the line instead of reading a string. If a new program is being written just nab the *read_env_file()* from the SPL compiler and all should be well.