Handy SPL Language Reference and Driver Tech Information

Chris Ebert & Chris Grigg
Epyx Sound Department

Rev. 21 March 1989

This document has two parts.

The first describes the Handy SPL language as it appears to the audio gnomes. This includes a description of the features of the language and some discussion of how to use them. *All Handy programmers should gain some amount of familiarity with the language* because some of the very useful Handy SPL features require intensive programmer-sound person interaction.

The second part describes driver internals, the music data format, and other technical details regarding the compiler-driver interface.

1. Handy SPL Language Reference
2. Handy Music Driver Tech Information

1. Handy SPL Language Reference

Handy SPL source code looks much like a generic assembly language, with one statement per line. The compiler is case sensitive. Conditional compilation, macros, included files and complex expression evaluation are all supported. The SPL language consists of Notes and Rests which express the score of a piece of music, and Commands which determine music voicing, support control structures, and allow access to other driver features. All event timing is in terms of music driver frames; at the time of this writing, 4 audio interrupts are planned per video frame, for a frame duration of 1/240 second.

Conditional Compilation

The spl compiler supports several conditional compilation directives. These may be nested to any level (so far as I know...).

#ifdef <name>

Example:

```
#ifdef foo
```

Causes subsequent code to be compiled if <name> has been defined.

#ifndef <name>

Example:

```
#ifndef bar
```

Causes subsequent code to be compiled if <name> has not been defined.

#if <expression>

Example:

```
#if 3*foo < 5
```

Subsequent code will be compiled if <expression> has a non-zero result.

#elseif <expression>

Example:

```
#elseif foo+bar
```

Subsequent code will be compiled if <expression> evaluates to non-zero and no previous #if, #ifdef, #ifndef or #elseif was taken.

#else

Subsequent code will be compiled if no previous conditional clause was taken.

#endif

Ends a set of conditional clauses. It is an error to encounter the end of a file before all conditionals have been terminated with an #endif statement.

Macros

The compiler supports macros with arguments. Arguments have the form ?0, ?1, ?2 ... ?n and must be declared when the macro is defined.

#macro <name> ?0 ?1 ...

Example:

```
#macro HiHat ?0 ?1
        AGD ?0
        LTR ?1
#endm
```

Define <name> as a macro with arguments ?0 ?1 ... ?n. A macro may have any SPL code in it. Macro definitions may not be nested. Macros may invoke other macros but any more than ten levels of such nesting will cause a compiler error. Macro arguments must start with 0 and use consecutive numbers in order. A macro may have any number of arguments.

#endm

End a macro definition.

Included Files

.in <filename>

Example:

```
.in macros.spl
```

Causes the named file to be opened and compiled. Any object code will appear at the point of inclusion. Files may include other files ad nauseum.

Complex Expression Evaluation

The compiler understands expressions of reasonable complexity made up of operators, numbers, and pre-defined constants. Constants are defined by assembler style equates or by the equals operator. Numbers are any hex or decimal number. Hex numbers are indicated by either the '$' or the '0x' prefix. The operators are C-style and have the same precedence. Operators are listed in precedence groups. Examples try to cover commonly used operators.

()	Parenthesis
-,!,~	unary minus, logical negation, bitwise negation
*,/,%	multiplication, division, modulo

+,-	addition, subtraction
<<,>>	shift left, shift right
<,<=,>,>=	less than, less than or equal, greater than, greater than or equal
==,!=	equal, not equal
&	bitwise and
|	bitwise or
&&	logical and
||	logical or
=	equals

Examples:

```
        2+3*2           ; evaluates to 8
        3 & 6           ; evaluates to 2
        3 && 6          ; evalueates to 1
        ~foo + 1        ; evaluates to -foo
        (foo << 2) / 2  ; evaluates to two times foo
bar     = $25           ; the label bar is set to hex 25
```

Notes

Note events are expressed as note pitch name and duration:

<note name> <duration in ticks>

For example:

```
g.2 4
bs3 16
```

Note names are three characters long: the letter of the note ("a" through "g"), a natural (".") or sharp ("s"), and the octave of the note (single digit); for example a.1 is the note A-natural in the first octave. Duration specifies the number of musical ticks (defined by the Tempo instruction; see below) for which the note will play.

Octaves change numbers at C and start with octave 0. On Handy notes above octave 5 will have tuning problems. Note that the Articulation commands (AGD and ASD) allow for space between consecutive notes without requiring a seperate rest event.

Rests

Rest events are expressed by the word "rest" and a duration in ticks

rest <duration in ticks>

For example:

rest 4
rest 16

The duration value for a rest is treated exactly the same as for a note.

Commands

Commands affect only the current voice, except for the Tempo and GlobalTranspose commands which affect both the current voice and all subsequently compiled voices.

Most Handy SPL Command statements have both a long, descriptive name and a short, fast-to-type name. Some of us hacks like short names.

ArticGateDur <gate-on duration in frames>
AGD <gate-on duration in frames>

Example:

AGD 6

Determines voice articulation. In every subsequent note played on this voice, the envelope generator gate bit will be turned on at the beginning of the note, will stay on for the indicated number of frames, and then will stay off for the remainder of the note's duration. This gives staccato effects. Stays in effect until another AGD or an ASD is executed.

ArticSepDur <gate-off duration in frames>
ASD <gate-off duration in frames>

Example:

ASD 2

Determines voice articulation. In every subsequent note played on this voice, the envelope generator gate bit will be turned on at the beginning of the note, will stay on until the indicated number of frames before the end of the note's duration, and then will turn off and stay off for the remainder of the note's duration. This gives legato effect.s Stays in effect until another AGD or an ASD is executed.

DefaultVoice
DFV

No parameters. Example:

DFV

Makes the compiler emit a set of simple-sounding default voicing parameters. DFV has no parameters.

If you simply want to hear something (say fresh from a MIDI file converter) you can use this instruction to audition the notes in the track before you start voicing.

EndOfMusic
EMS
END_SFX
EndSfx

No parameters. Example:

```
...
EMS
```

Indicates the end of a voice of music. Required. No parameters.

The END_SFX and EndSfx tokens are used in other machines' SPL versions, and are left in Handy SPL only to allow music in some of these other SPL's to be no-brain ported. Use EMS or EndOfMusic in all new SPL code.

Feedback <feedback register image>
FB <feedback register image>

Example:

```
FB $0b0b
```

Sets the voice's hardware feedback register to the indicated pattern. What it will sound like? Well, that's another question...

GlobalTranspose <pitch offset in semitones>
GTR <pitch offset in semitones>

Example:

```
GTR -12
```

Causes all subsequent scored pitches in all voices to be offset by the indicated number of semitones before the note pitch is emitted. Stays in effect until another GTR supercedes it. The pitch offset number may be positive (transpose up) or negative (transpose down).

At the start of compilation of a piece, GTR is set to 0.

LocalTranspose <pitch offset in semitones>

LTR <pitch offset in semitones>

Example:

```
LTR 7
```

Much like the GlobalTranspose function, but LTR only applies to the current voice.

At the start of compilation of a voice, the local transpose for the voice is cleared to zero.

StartLoop
SetLoop
LoopTop
SLP

No parameters. Example:

```
SLP
```

Mark the start of a range of SPL code to be looped; the next LOP command marks the end of the looped range. No parameters.

There are two levels of looping in the driver. SetLoop's may be nested and the compiler will keep track of which loop is which. For example, the following structure is a nested loop:

```
SLP
        (0 or more SPL statements)
        SLP
                (0 or more SPL statements)
        LOP <number of times to play inner loop>
        (0 or more SPL statements)
LOP <number of times to play outer loop>
```

In this example, the inner SLP and LOP could have been a SLP2/LOP2 pair.

Loop <number of times to play loop>
LOP <number of times to play loop>

Example:

```
Loop 4
```

Marks the end of a range of SPL code to be looped and indicates the number of times to play the loop. Note that "Loop 1", not "Loop 0", plays the looped region once. See SLP for a thorough discussion of loops.

StartLoop2
SetLoop2
LoopTop2
SLP2

Loop2 <number of times to play inner loop>
LOP2 <number of times to play inner loop>

These instructions are explicit names for the inner level of looping. If inadvertantly used as the outer level, the compiler translates them to SetLoop and Loop instructions. These statements operate analogously to the outer-level looping statements.

These alternate names are preserved from other machines' SPLs and, although not required for nested loops in Handy SPL, both simplify some music ports and may be used for structural clarity in original Handy music.

Tempo <number of frames per tick>
TPO <number of frames per tick>

Set the number of audio frames per musical tick to <n>. All note and rest durations are expressed as multiples of this value. The Tempo command applies to all notes and rests in the current voice from this point on, and to all subsequent voices compiled until another Tempo command is encountered.

Priority <priority number>
PRI <priority number>

Set the HSFX Driver priority for this voice to <priority number>. This feature is intended to allow the sound artist to allow important parts of a piece of music to be heard rather than always lose priority battles to sfx. *Audio artists note:* It's important to discuss music and sfx priorities with the programmer; don't use this command without discussing the question with the programmer first. See below s suggested priority ranges for an idea of how things should work.

Proposed Standard Priority List:

Priority Range	Use
00 - 31	Reserved for music driver
32 - 63	Standard Music range
64 - 191	Standard SFX range
192 - 223	High Priority Music
224 - 254	High Priority SFX
255	Reserved for reserved channels.

If this or some similar scheme is adopted the whole process of integrating music and SFX into a game should be much smoother: less fiddling around specifying and respecifying priority ranges and less opportunity for miscommunication.

ADSR <attack> <peak> <decay> <sustain> <release>

Example:

ADSR 2, 127, 5, 80, 20

Sets the volume envelope shape for subsequent notes played on this voice. When a note is started, the voice's volume ramps up from zero to the <peak> value over a span of <attack> audio frames, and then ramps from the <peak> to the <sustain> over a span of <decay> frames. The volume holds at <sustain> for as long as the envelope generator's gate bit is held on (see AGD and ASD for a discussion of the gate bit). When the gate bit is turned off, the volume ramps from <sustain> down to zero over <release> frames. All ramps are linear. Stays in effect until another ADSR is executed by this voice. The <attack>, <decay> and <release> times are in audio frames and have a range of sixteen bits. The <peak> and <sustain> values are volume levels and have a range of seven bits.

VoiceFreqMod <start> <atk> <dcy> <sus> <rel>
VFM <start> <atk> <dcy> <sus> <rel>

Example:

VFM 0,-55,-35,-10,20

Sets the frequency modulator (pitch envelope) shape for subsequent notes played on this voice. Stays in effect until another VFM is executed for the current voice. Useful for drums and other sfx-like sounds.

Frequency modulation allows a note's pitch to change over the life of the note, in a way tied to the current voice's envelope generator (ADSR) timing. The pitch envelope linearly ramps at the <atk>, <dcy>, <sus>, and <rel> rates respectively during the ADSR's <attack>, <decay>, <sustain>, and <release> phases, with continuity at the transitions between phases. The rates can be positive or negative.

When a VFM is in effect, a note's frequency starts at the usual frequency for the scored note (after taking into account the GTR and LTR) *plus* <start>, then ramps at the rate of <atk> per audio frame for the ADSR's <attack> duration. Any non-zero <start> will offset the notes from musically correct pitch (but that's not always bad). During the ADSR's <release> phase the VFM ramps from the value it had at the end of the <atk> phase at the rate of <dcy> until the ADSR hits its <sustain> level. Then the VFM ramps from its value at the end of teh <atk> phase at the rate of <sus> until the ADSR gate bit goes off. When the gate bit goes off, the VFM ramps from its value at the end of the <sus> phase at the rate of <rel> until the note ends.

To turn off the effect of a VFM, do a VFM with all-zero parameters:

VFM 0,0,0,0,0

Changes in ADSR will affect any VFM in effect at the time. All VFM parameters are sixteen bit signed values.

Shifter <shifter register image>
SHF <shifter register image>

Example:

Shifter $1b1c

Loads the hardware shifter register for the current voice with the indicated pattern.

Note that the exact shifter load time is non-deterministic, subject to other activity in the machine at the time.

VoiceFBMod <start> <atk> <dcy> <sus> <rel>
VFBM <start> <atk> <dcy> <sus> <rel>

Example:

VFBM $1121,1,-45,2376,-2

This sets up a feedback modulator in much the same way that VFM sets up a frequency modulator. In this case, a <start> value is loaded into the hardware feedback register at the start of a new note, and is ramped and rewritten to hardware every audio frame during the ADSR's <attack>, <decay>, <sustain>, and <release> phases at the <atk>, <dcy>, <sus> and <rel> rates. All parameters are sixteen bit signed values.

Hook <token>
User <token>
USR <token>

Example:

User CarGoBoom

Once the address of a User Routine has been placed in a special location in the music driver (see the Music Driver documentation and example files for this address), execution of this SPL command makes the music driver call the User Routine once every audio frame, starting with the current audio frame. The <token> parameter is an 8-bit value that is passed to the User Routine in the accumulator at every call. This can be used for any purpose but was originally designed to enable synchronous starting of music. This feature requires a great deal of programmer-sound artist interaction for best effect (Kids! Don't try this at home!).

SPL programs have no way to turn off the User Routine call in each audio frame; the main program and/or User Routine have to clear a flag to disble the call (again, see Music Driver documentation and example files).

CallSFX <name> <duration in ticks>
SFX <name> <duration in ticks>

Example:

CallSFX DogBark.HSFX qrtrnote

Start the HSFX sound effect <name> on the current channel and let it play for a musical duration of <duration> ticks. The <name> parameter is the full filename of the sound effect to play.

The note, rest, or command immediately following the CallSFX is executed at the musically correct time. If the sfx is longer than <duration> it is killed at that time; if shorter, the time interval is filled with silence. This feature allows musicians to include sound effects in music without bothering the programmer too much (see the User instruction for an alternate method).

GosubSoundJob <name>
Ring <name>
GSJ <name>

Example:

GSJ LeadIntro1.SPL

SPL language gosub call, where <name> is the full file name of an SPL track/program you want to call as a subroutine. Plays the indicated file once, and once it ends (via an EndSfx or any of its alias commands) continues on to the following command. Used to create larger pieces out of smaller phrases.

2. Handy Music Driver Tech Information

Theory Of Operation

This section deals with some details of the Handy music driver. The music driver exists as a 'shell' around the HSFX driver. Notes are simply 5-keyframe HSFX events whose command parameters are filled in by the music driver (according to the information in the music data) before the music driver tells the sfx driver to play the sfx. Playing the sfx plays the note. These note sfx have all the habits and properties of 'normal' sound effects. Somewhere in the Handy Music Driver is a group of four keyframe sets, one set per voice, with each set five keyframes in length. These keyframe sets are used by the drivers to define the notes to be played. These frames have the following form; each location is of word size (16 bits).

Format Of The Sfx Definitions That The Music Driver Modifies

Frame 0: Attack Phase

Priority	set by SPL Priority statement
Start Time	Always zero
$0fb0	Command Flag Word (CFW)
Frequency Accumulator	set by VFM statement's start parameter and by note pitches
Shifter Accumulator	set by Shifter statement
Feedback Accumulator	set by Feedback statement or VFBM statement
Volume Accumulator	set by ADSR **??** initial volume is always zero
Frequency Interpolation	set by VFM statement's attack parameter
Feedback Interpolation	set by VFBM statement's attack parameter
Volume Interpolation	set by ADSR statement's attack and peak parameters

Frame 1: Decay Phase

Start Time	Set by ADSR statement's attack time parameter
$00b0	CFW
Frequency Interpolation	set by VFM statement's decay rate parameter
Feedback Interpolation	set by VFBM statement's decay rate parameter
Volume Interpolation	set by ADSR statement's peak, sustain, and decay parameters

Frame 2: Sustain Phase

Start Time	Set by ADSR's attack and decay parameters
$00b0	CFW
Frequency Interpolation	set by VFM statement's sustain rate parameter
Feedback Interpolation	set by VFBM statement's sustain rate parameter
Volume Interpolation	Usually 0, unless music driver modifies for short notes

Frame 3: Release Phase

Start Time	Set by note duration and AGD/ASD and parameters
$00b0	CFW
Frequency Interpolation	set by VFM's relase rate parameter
Feedback Interpolation	set by VFBM's release rate parameter
Volume Interpolation	set by ADSR's release rate parameter

Frame 4: End of Sfx/Note

Start Time	end of note. Set by AGD/ASD and note duration
$0001	CFW signifying end of sfx

Music Data Module Format

Each music data module consists of a header with offsets to the individual voices' data, voices' actual music data, and if the music calls any sfx they appear too. All "offsets" in this document are offsets measured from the start of the music data module.

These first four offset words are used as program counter variables by the music driver at runtime, i.e. their content is volatile:

word: offset for current voice 0
word: offset for current voice 1
word: offset for current voice 2
word: offset for current voice 3

This block constitutes a catalog of the sound data resources available in the module. By convention, music driver voice definitions come before sfx definitions.

byte: number of music voices plus number of included SFX
word: offset to music voice 0
word: offset to music voice 1
word: offset to music voice 2
.
.
.
word: offset to last music voice
word: offset to sound effect 0 (if any)
word: offset to sound effect 1
word: offset to sound effect 2
.
.
.
word: offset to last sound effect

The rest of the module contains the data actually used by the drivers to play notes and other sounds:

Duration and Pitch tables

Music voice data

SFX data for sfx called from music voices within this module, if any are called

Any sound effects in a music data module are for use by the music driver, called by an Sfx statement in the music SPL source. Most data modules won't have any sfx at all. As the offsets to a piece of music can change any time a part is modified it is recommended that this table be indexed to select musical voices to play. Sound people are responsible for providing a document (suggested here because people forget things if you just tell them) describing the parts a programmer will use to play music.

Format Of Pitch And Duration Tables

Pitch Tables consist of sixteen frequency words. A frequency word is made by shifting the multiplier value for the Handy hardware to the left by the value of the bit representing base period plus two. For example, a multiplier value of $fd with a base period of 64 (bit 6) appears as $fd00. Pitch values of zero indicate a rest.

Duration tables are eight words long. Each word is simply the number of audio frames in the given duration. Duration values of zero are 4.5 minutes long (aieee!).

Pitch:

<multiplier> << <bit of period + 2>

Duration:

<musical duration> * <current tempo setting>

There is some optimization of table use: when it's time to emit a new pitch or duration table, the compiler compares the table it wants to emit now to all previously-emitted pitch and duration tables. If a match is made the note data for the range in question are re-translated to use the older table and the new table isn't emitted. This gives a lot of compression in some files, and not too much in others. C'est la Vie. The last tables in a data module may be partially empty, as the compiler only generates full tables.

Data Format For Music Voices (Tracks)

The data for the music driver consists of commands and notes (a rest is just one kind of note). Notes are bytes with the sign bit off; the lower four bits are the index into the current pitch table and bits 6 to 4 index into the current duration table. Commands are heralded by a Command Flag Word (CFW) which has the sign bit of the low-order byte set (6502's are little-endian). CFW's may be followed by one or more parameter data, and these data may be words or bytes.

Notes:

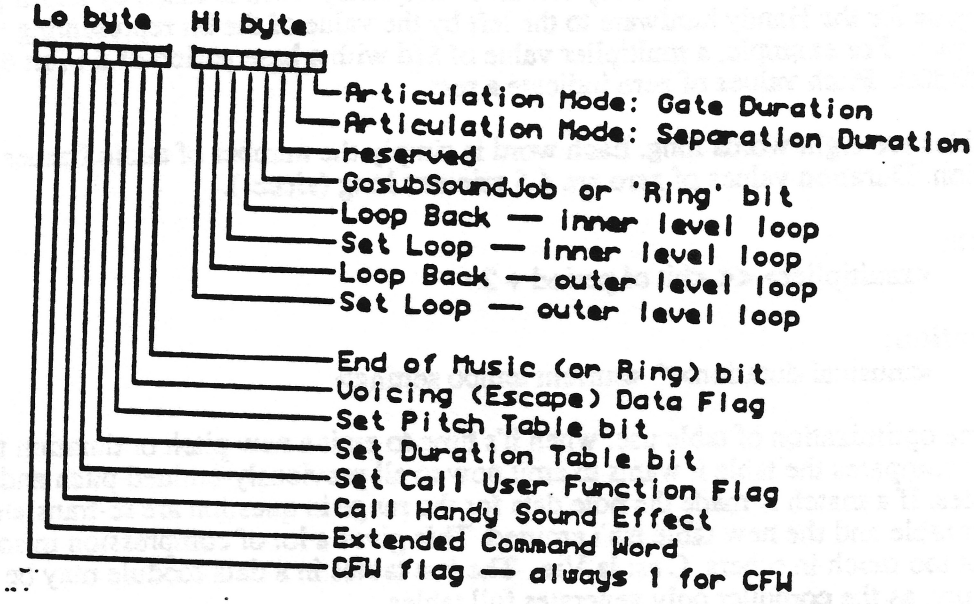
Notes are 1 byte long each. The note's pitch and duration are encoded thus:

`0dddpppp` where ddd is the 3-bit duration index and pppp is the 4-bit pitch index

When a note is executed, the music driver gets the note frequency by using the pitch index to index into the current pitch table, and the note duration by using the duration index to index into the current duration table. These tables are generated by the SPL compiler and stored within the music data module, and the compiler emits commands as needed within each voice's data stream to pick which pitch table and which duration table are "current".

Commands:

Here, as in the 6502, the CFW appears in Lo-Hi order.



Wherever possible commands are accumulated into a single CFW, so often many of these bits will be set together. Each command bit, if set, tells the music driver to look for that command's parameter data following the CFW; if a command's bit is clear, the command's data are omitted. These data appear in the following order. In addition to being the order of parameters, this is the order of execution...with the exception of the looping. Inner loop ends occur before outer loop ends; outer loop starts occur before inner loop starts; and both levels of loop ends occur before either level of start. In the voice data, the inner level of looping must always appear inside the outer level; the current SPL compiler ensures this.

1. Extended Command Word. 1 word: the command word. No extended commands have been defined yet, but data for them will appear before any data for the normal command word. This mechanism has been provided to allow for future expansion of the music driver command set.

2. Gosub Sound Job. 1 byte: the number of the music voice (within this data module) to call. The byte is used as an index into the table of offsets at the beginning of the data module, and the music driver data at that location is started as a subroutine. At its end, control proceeds from this command forward. Maximum gosub depth is **??** 3.

3. Outer Set Loop. 1 byte: times to play loop. The loop count specifies the number of times to play the looped phrase, so a loop count of 1 means *play* it once, not *repeat* it once!

4. Outer Loopback. No Parameters. This makes the driver decrement the outer loopback counter and, if it's greater than 1, loop back to the last-encountered Set Outer Loop.

5. Inner Set Loop. 1 byte: loop count. Analogous to Outer Set Loop.

6. Inner Loopback. No Parameters. Analogous to Inner Loopback.

7. Reserved: No parameters have been defined, but they would appear here.

8. Set Articulation mode: Gate Duration: and Set Articulation mode: Separation Duration. 1 word: the articulation duration; for AGD this is the number of audio frames

from the start of the note until the start of the release phase, and for ASD this is the number of audio frames from the start of the release phase to the start of the following note. These commands' parameters are shown at the same position because the SPL compiler guarantees that even if both modes are used without any notes or Ring commands between them, only the last articulation mode statement used will be compiled.

9. **Call Handy Sound Effect.** 1 byte: the number of the sound effect to call. This number will be used as an index into the table of offsets at the beginning of the data module, and the resulting address will be passed to the HSFX driver to start the sfx.

10. **Set Call User Function Flag.** 1 byte: parameter to be passed to the user routine when first called. This sets a music driver flag that causes the user function routine to be called on the current and all subsequent audio frames. This per-frame call continues until the flag is cleared by the next CFW with the Set Call User Function bit clear, or the host program clears the flag.

11. **Duration Table.** 1 word: the offset of a new duration table to use when turning note events' duration indexes into actual audio frame counts.

12. **Pitch Table.** 1 word: the offset of a new pitch table to use when turning note events' pitch indexes into actual frequency register images.

13. **Voicing Data.** 1 or more blocks of three bytes each. In each of these blocks, the first byte is an index into the keyframe set (the music driver variables that are passed to HSFX as a sfx definition) for the current voice; the following word is data to be indexed-stored at that location. The offsets have the sign bit turned on to identify them. An initial voicing data byte of $ff may appear: this tells the music driver to clear a number of key frame locations. If a CFW follows the current command(s) (with no intervening notes or rests), the Voicing Data bit in this CFW must be set and after the last 3-byte voicing data block a single terminating $ff byte must appear .

14. **End Of Music.** No Parameters. This signals the end of the current voice of music. If this voice/track was started via a Gosub Sound Job, control is returned to the caller; otherwise the current voice is freed.

..end..