

MMA Technical Standards Board/ AMEI MIDI Committee

Technical Note – April 2006

Title: Downloadable Sounds Level 2 Amendment 2 (DLS 2.2)

Abstract:

Contains corrections to DLS 2.1 to address errors and clarify issues. Includes specification of some new behaviors selected for Mobile DLS but also relevant to the broad MIDI market. The Scalable MIDI Working Group (SMWG) developed a Mobile DLS Specification, based in large part on DLS 2.1. During the review of DLS 2.1, SMWG identified a few errors and portions that needed clarification, and proposed some new behaviors for Mobile DLS which are also relevant to the broad MIDI market. All of these changes appear in the new 2.2 version of the DLS specification.

Details:

A. Note Exclusivity

DLS 2.1 text for Note Exclusivity, section 1.4.4, incorrectly states that the Non-Self-Exclusive flag is in the usKeyGroup field of the region header chunk, but it is actually in the fusOptions field. In addition, the text incorrectly describes the polarity of the bit as though it were a “Self-Exclusive” flag rather than a “Non-Self-Exclusive” flag.

[New Section 1.4.4 Text]

A DLS 2.2 Device has provisions for two forms of note exclusivity on a MIDI channel basis, shown below. The first form of exclusivity involves notes on a MIDI channel with the same MIDI note number. DLS-2 introduced the concept of a shutting down an oscillator, to give the sound developer more control over polyphony without degrading sound quality. When a note exclusive event occurs, the Volume envelope proceeds directly to the release phase, but using the time constant EG1_SHUTDOWNTIME, instead of EG1_RELEASETIME.

By default, if a MIDI Note-On event is received and there are oscillators previously assigned to the same MIDI note and MIDI channel that have not received a Note-Off event (or Note-On event with a velocity of 0), the control logic will immediately shutdown those oscillators. However, there is a Non-Self-Exclusive flag in the fusOptions field in the region header chunk that, when set, will defeat this logic. When the Non-Self-Exclusive flag is set, Note-On events for a particular MIDI note number on a MIDI channel do not cause a shutdown of oscillators assigned to the same MIDI note and MIDI channel. When the Non-Self-Exclusive flag is not set and the synthesis engine receives a second Note-On event of the same MIDI note number on the same MIDI channel, the second Note-On will cause a shutdown of the first note. The Non-Self-Exclusive flag is off by default.

The second form of note exclusivity is useful for drums and sound effects. Each region can be assigned a Key Group. If a Note-On event is received and there are oscillators that have been assigned to play a region that has the same Key Group number as the region for the new Note-On, those oscillators are shutdown. As an example, this can be used to create mutually exclusive Open, Closed and Pedal High Hat sounds for a drum group. A second example might be in a sound effects collection to stop a squeaking door sound when the sound of the door closing is triggered.

Description	DLS instrument data
Oscillator Shutdown with EG1_SHUTDOWNTIME	Yes
Channels capable for mutually exclusive mode	Any channel

Table <>: Supported Note Exclusivity Functionality

[END]

B. Bipolar Transforms

The description of bipolar was not precisely correct for all transforms, although it worked for many.

[New Section 1.6.5.2 Text]

The Bipolar flag maps the controller input signal to a range of -1 to +1. For non-linear input transforms, this introduces a slight complication, as the non-linearity must be mirrored in both the positive and negative quadrants.

For the concave and convex input transforms the following algorithm can be used to perform the bipolar mapping:

```

if (bipolarFlag)
{
    value = (2 * input) - MaxValue;
    output = sgn(value) * InputTransform(abs(value));
}
else
    output = InputTransform(input);

```

On the other hand, for linear and switch input transforms, the algorithm is:

```

if (bipolarFlag)
{
    value = InputTransform(input);
    output = (2 * value) - 1;
}
else
    output = InputTransform(input);

```

[END]

C. Convex and Concave Transforms

The equations for the concave and convex transforms were incorrect. The concave transform had problems with 14-bit controllers. When applied to a 14-bit controller, the curve did not match that of the corresponding 7-bit controller. The convex transform was simply incorrect, producing a curve that was neither concave nor convex.

[New Section 1.6.5.3 Text: Concave and Convex]

Concave: Input values are mapped in a concave fashion as defined by the following equation:

```

For Input > (1.0 - 10-12/5) * MaxValue,
    Output = 1.0

For Input ≤ (1.0 - 10-12/5) * MaxValue,
    Output = -(5/12) * log10(1.0 - Input/ MaxValue)

```

Convex: Input values are mapped in a convex fashion as defined by the following equation:

```

For Input < (10-12/5) * MaxValue,
    Output = 0.0

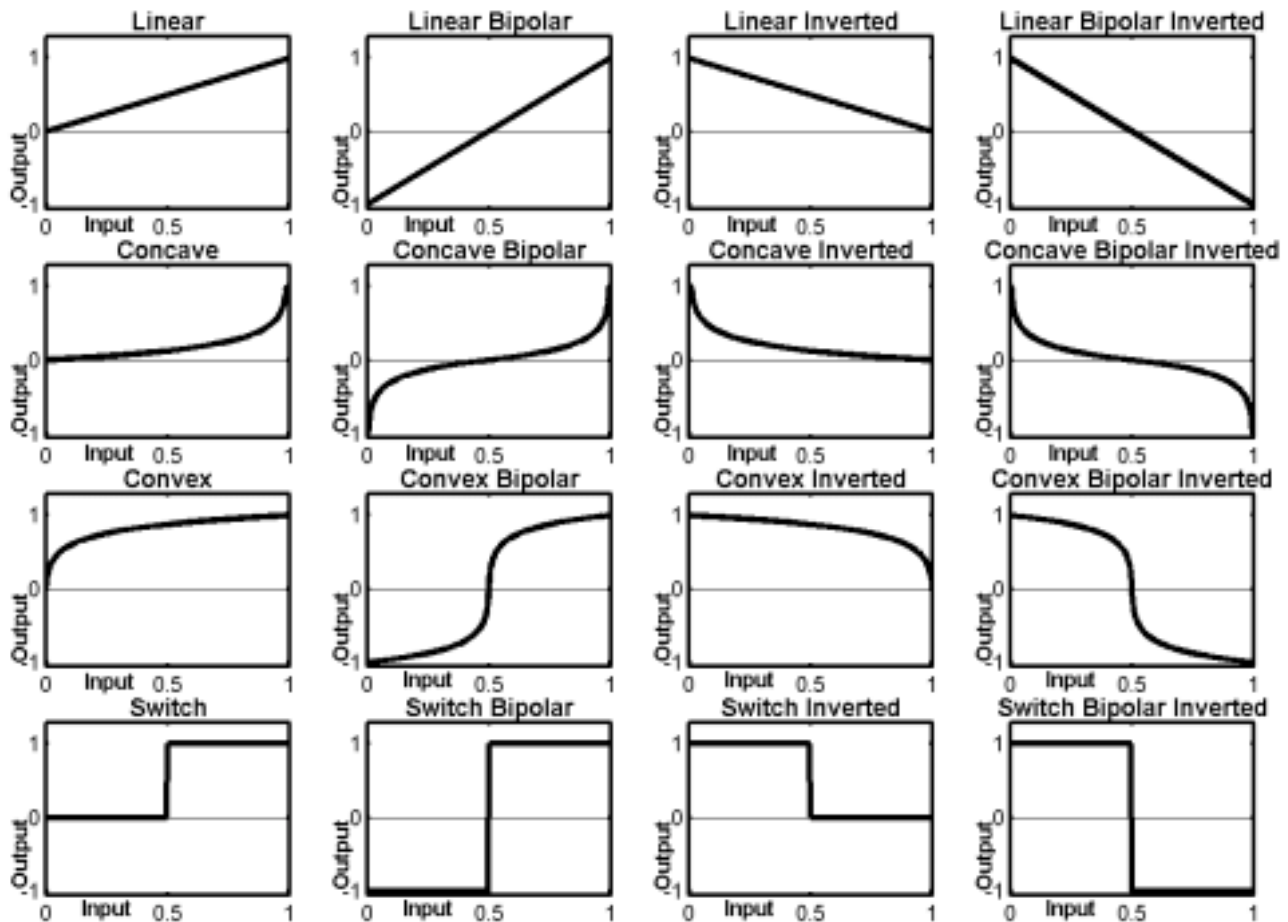
For Input ≥ (10-12/5) * MaxValue,
    Output = 1.0 + (5/12) * log10(Input/MaxValue)

```

[END]

D. Transform Graphs

The graphs in DLS 2.1 did not match the equations. The corrected graphs are shown below.



E. History of the Concave Transforms

The “History of the Concave Transform” section is greatly expanded to provide a clear picture of how we arrived at the current forms of the equations.

[New Section 1.6.5.4 Text]

The Concave Transform was created to maintain compatibility with existing General MIDI implementations. Its name comes from the shape of the curve when graphed as a linear volume output. In DLS-1, it was defined as a series of fixed connections from MIDI controllers 7 and 11, and velocity, to attenuation in dB using the following formula:

$$\text{atten}_{\text{dB}} = 20 * \log_{10}(127^2 / \text{Input}^2)$$

To map this fixed transform onto the flexible routing model found in DLS-2, it is necessary to adjust the output of the transform so that it produces the desired curve in the range 0 to 1, such that when scaled at 96 dB and applied to the DST_GAIN summing node, it produces the same output curve as that of the DLS-1 and GM devices.

Due to these changes, in this Mobile DLS specification the convex transform corresponds to the theoretical definition of the concave function and the concave transform corresponds to the theoretical definition of the convex function.

In DLS 2.1, the scaling factor of 5/12 was derived from the 96 dB scaling factor, as shown here:

The attenuation formula from DLS1.0:

$$20 \cdot \log_{10}((127/\text{Input})^2)$$

is equal to:

$$40 \cdot \log_{10}(127/\text{Input})$$

Dividing by 96 to scale back to a range of 0–1:

$$(40/96) \cdot \log_{10}(127/\text{Input})$$

reduces to the equivalent:

$$(5/12) \cdot \log_{10}(127/\text{Input})$$

Since this was attenuation, whereas DLS-2 uses gain, we invert the input, so it becomes:

$$(5/12) \cdot \log_{10}(127/(\text{MaxValue} - \text{Input}))$$

The IScale values for connections to DST_GAIN are intended to be negative numbers. For example, the default IScale value for velocity to gain is –96dB. And the input transform is defined to be inverted concave. So the default velocity to gain calculation is:

$$\text{gain} = -96 \cdot (5/12) \cdot \log_{10}(127/(127 - (127 - \text{velocity})))$$

Selecting a velocity of 127 produces 0dB of gain:

$$\text{gain} = -96 \cdot (5/12) \cdot \log_{10}(127/(127 - (127 - 127))) = 0\text{dB}$$

Selecting a velocity of 0 produces -96dB of gain – it forces the inverted concave transform to equal 1.0 because the input is equal to MaxValue:

$$\text{gain} = -96 \cdot (5/12) \cdot \log_{10}(127/(127 - (127 - 0)))$$

$$\text{gain} = -96 \cdot 1.0 = -96\text{dB}$$

It should be noted that 96dB is an approximation of 16-bit dynamic range. The true value is $20 \cdot \log_{10}(65536)$, or 96.3296dB. However, the approximation cancels out when re-linearizing the final output to gain, only producing a difference when the output is saturated to 1.0.

The concave transform in DLS 2.1 has some problems. First, it was defined using Range, whereas the intended implementation used MaxValue. Second, the output is greater than 1.0 before the input reaches MaxValue when applied to a 14-bit controller. Third, when the 7 least significant bits of a 14-bit controller are zero, the output values do not equal the output values using a 7-bit controller.

The first and third problems are addressed by using $(127/128) \cdot \text{Range}$ instead of Range. The second problem is addressed by the modified saturation threshold equation.

[END]

F. Digital Filter

Figure 7 of the DLS 2.1 specification was not visible in the printed publication. It is shown below as Figure 2. Figure 6 of the DLS 2.1 specification is changed in DLS 2.2. It is shown below as Figure 1. In addition, the behavior of the filter when F_c was greater than $F_s/6$ is clarified as follows:

[New Section 1.15.2 7th bullet point]

- The cutoff frequency according to this specification is not well defined as the resonance frequency approaches one octave below the Nyquist frequency, corresponding to a pole angle of $\pi/2$ radians. In fact, at a pole angle of $\pi/3$ radians and a radius of 0.9, the resonance frequency calculated by this method is almost exactly equal to $Nyquist/2$, even though the resonant peak is at $Nyquist/3$. Also, using these parameters, the calculated cutoff frequency intersects the filter response curve at 0 dB. As the pole angle is increased, filters specified in this manner change from low-pass to a simple resonator with no attenuation, and finally to a high-pass characteristic (actually only providing high frequency gain). In view of these facts, and that second order filters of this type will not have a dramatic effect on the sound in this case, cutoff frequencies when calculated according to this specification are only required up to $1/6$ th of the sample rate F_s . Higher cutoff frequencies may be specified and devices may optionally support them. However, support is not guaranteed by all platforms. Device requirements support output sample rates as low as 22.05kHz in DLS-2, although many implementations have a higher output sample rate such as 48kHz. Due to this, DLS synthesizers operating at lower output sampling rates may encounter instruments where the filter cutoff frequency is within the range from $F_s/6$ to $F_s/2$. Therefore, it is required that the filter implementation may not introduce disturbing processing artifacts when the filter cutoff is sweeping beyond $F_s/6$. When the specified cutoff frequency is greater than that supported by a DLS device, it will be limited to the maximum supported cutoff frequency. The one exception to this rule is that when the specified cutoff frequency is greater than the Nyquist frequency, the filter will have a flat passband characteristic, regardless of the specified resonance. Content developers are advised to specify the maximum value (0x7FFFFFFF) for the filter cutoff when no low pass filter is desired. However, devices will respond as defined in this text.

[END]

Also, based on a suggestion by Tom Savell, the MMA Tech Board recommended adding clarifying language that the filter nominal cutoff freq calculation need not directly drive the implementation's filter coefficient calculation. In other words, the DLS spec shouldn't mandate a particular filter topology, nor a particular trajectory for the pole radius/angle vector. This may make it clearer that 'more musical' filter implementations are OK. The following text is proposed:

[New Section 1.5.2 Text to be inserted at end of section]

The equations provided for r and θ are not intended to mandate a method of calculating coefficients for use by the DLS device implementer. Rather, the device implementer should rely on the equations for F_c and resonance to measure their own filter implementation relative to the specified F_c and resonance in the DLS file. There are a number of well known, musically useful methods of calculating filter coefficients, especially when the parameters such as F_c vary with time, i.e. a filter sweep. The implementer may find more desirable or efficient methods of calculating time-varying filter coefficients. The method will most likely vary with different implementations. For example, hardware implementations may use different methods than software. Such implementations may result in actual F_c and resonance values that differ from those specified, and in some cases the differences may provide a more desirable result. The purpose of providing the equations for r and θ was to provide a prototype filter for measurement purposes only, and may not be of interest to all implementers of DLS devices. Implementers are advised to evaluate their criteria for filter sound quality and act accordingly.

[END]

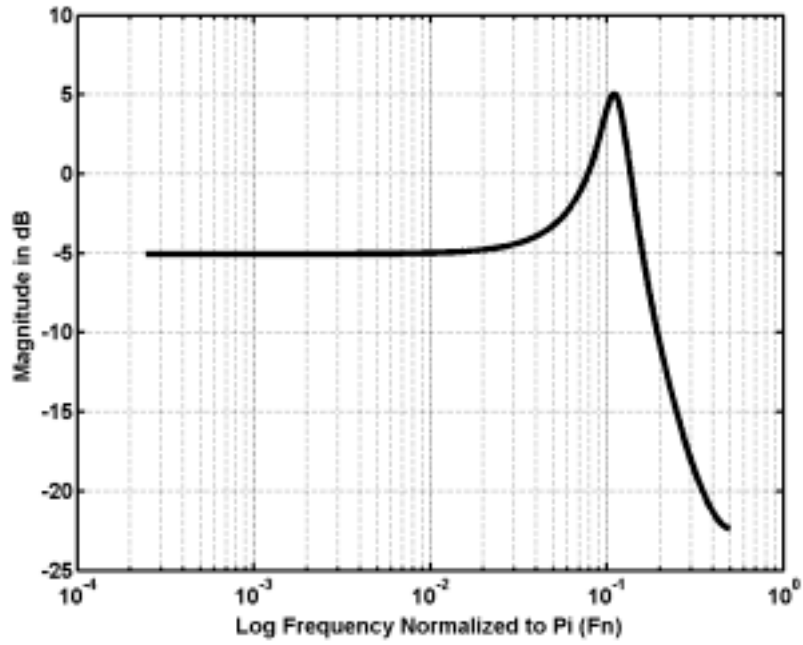


Figure 1: "Figure 6 – Resonant Filter"

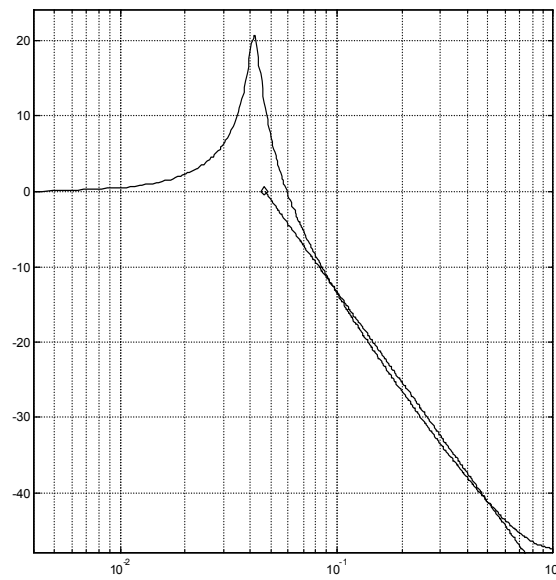


Figure 2: "Figure 7 – Filter Cutoff Frequency"

G. Mod LFO to Gain

DLS 2.1 contains a correction for the ModLFO->gain connections so that they are bipolar, non-inverted (DLS 2.0 was unipolar, inverted). However, the table entry for Mod LFO Channel Pressure to Gain was not updated during the DLS 2.1 revision process.

[New Table Entry]

Articulator Name	UsSource (*)	B	I	Transform	UsControl (*)	B	I	Transform	UsDestination (*)
Gain									
Mod LFO Channel Pressure to Gain	SRC_LFO	T	F	Linear	SRC_CHANNEL PRESSURE	F	F	Linear	DST_GAIN

[END]

H. Mod EG to Pitch

The Mod EG to Pitch connection is listed as bipolar, non-inverted. It should be unipolar, non-inverted.

[New Table Entry]

Articulator Name	UsSource (*)	B	I	Transform	UsControl (*)	B	I	Transform	UsDestination (*)
Pitch									
Mod EG to Pitch	SRC_EG2	F	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH

[END]

I. Bank Select

Bank select is a behavioral difference proposed for the DLS 2.2 specification.

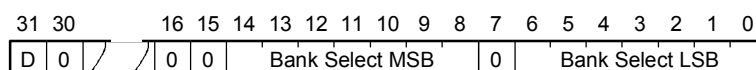
The DLS 2.1 specification divided bank select addresses into two separate spaces, one for drums and the other for melodic instruments. The address space was split using bit 31 of *ulBank* to indicate a drum instrument. The MIDI channel number was used to determine which space to search for matching bank MSB and LSB values. This worked fine for a General MIDI-like implementation, where drums are always on channel 10 and melodic instruments are on every channel except 10. This however, does not work for SP-MIDI, and perhaps more important to the broad market, it does not work for GM2 instruments.

It is proposed that DLS 2.2 incorporate the Mobile DLS bank select behavior, which allows drums and melodic instruments on any channel, with power-on defaults to be General MIDI compatible. Any built-in GM or GM2 bank can be selected using the GM2-compatible pre-determined bank select MSB values of 0x78 and 0x79. The only potential conflict with existing content is if there is a drum instrument and a melodic instrument for which *ulBank* differ only by bit 31. Thus, they will both have the same bank select MSB and LSB.

[New Section 1.4.6 Text]

The DLS device must support MIDI Bank Select and Program Change messages as the method of selecting the instrument to be played on a MIDI channel. The Bank Select address space consists of 16,384 banks, represented by the MIDI Controller Change MSB and LSB Messages (controllers 0 and 32, respectively), with each bank supporting up to 128 instruments, for a total address space of over 2 million instruments.

The device does not take any direct action upon receipt of Bank Select messages, but saves the state for future reference. Upon receipt of a Program Change message, the device should refer back to the previously received Bank Select values to decode the correct instrument to be played. The Bank Select LSB and MSB are concatenated as 8-bit bytes to form *ulBank* as shown:



Content authors must send Bank Select MSB, Bank Select LSB, and Program Change messages in that order to insure that the correct instrument is selected.

Bit 31 is informative and indicates a drum instrument.

A MIDI channel that references *ulBank* with MSB 0x79, via a Bank Select and Program Change, shall be treated as a melodic channel using the default melodic bank on the device. A MIDI channel that references *ulBank* with MSB 0x78, via a Bank Select and Program Change, shall be treated as a drum channel using the default drum bank on the device.

DLS 2.2 devices that support General MIDI or General MIDI 2 should provide dedicated locations for instrument and percussion banks. The GM or GM2 percussion programs shall appear at bank select MSB=0x78 LSB=0x00, starting with Program 0x00. If a GM instrument set is provided, the melodic programs shall appear at bank select MSB=0x79 LSB=0x00. If a GM2 instrument set is used, the melodic programs shall appear at bank select MSB=0x79, LSB=0x00 through 0x09, as specified in RP-024 General MIDI 2 Specification.

Custom program(s) from DLS content files may use the same bank and program number(s) as any of the DLS device's built-in GM/GM2 programs; if so, they will temporarily override the corresponding built-in programs until they are unloaded.

[END]

J. Dependence on *mmreg.h*

The original DLS specification is dependent on a 'C' header file owned and maintained by Microsoft. This file, *mmreg.h*, is not under control of the MMA, and dependence on it can be a potential issue, especially for companies that are not implementing products for the Microsoft platform. A new header file is proposed that extracts the relevant definitions from *mmreg.h* as of a date to be chosen by the MMA, and allows further definitions to be made within the MMA.

(Adoption of the header file by AMEI/MMA will precede actual publication of the 2.2 Specification).

K. Wave CODEC Extensions

DLS 2.1 specification only supports 8-bit and 16-bit PCM waveforms. It is proposed to support additional CODEC's using the WAVE_FORMAT_EXTENSIBLE mechanism as used in the Microsoft Wave file format.

[New Text]

2.16.1 Format Chunk <fmt-ck>

The WAVE format chunk <fmt-ck> specifies the format of the <wave-data>. The <fmt-ck> is defined as follows:

```

<fmt-ck>          →   fmt ( <common-fields>
                        <format-specific-fields>
                        )
<common-fields>  →   struct {
                        WORD  wFormatTag;
                        WORD  wChannels;
                        DWORD dwSamplesPerSec;
                        DWORD      dwAvgBytesPerSec;
                        WORD  wBlockAlign;
                    
```

```

    }

    <format-specific-fields>  →    <PCM-format-specific>

                                   OR

                                   <WAVE-format-extensible-specific>

    <PCM-format-specific>      →    struct {

                                   WORD    wBitsPerSample;

                                   }

    <WAVE-format-extensible-specific> → struct {

                                   WORD    wBitsPerSample;

                                   WORD    cbSize;

                                   union {

                                   WORD    wValidBitsPerSample;

                                   WORD    wSamplesPerBlock;

                                   WORD    wReserved;

                                   }

                                   DWORD    dwChannelMask;

                                   GUID     SubFormat;

                                   }
    
```

The <common-fields> struct:

Field	Description
wFormatTag	A number indicating which format specific field to use (either <PCM-format-specific> or <WAVE-format-extensible-specific>). If the number is WAVE_FORMAT_EXTENSIBLE (0xFFFFE), the data following the <common-fields> is in the form detailed in <WAVE-format-extensible-specific>, below; if it is any other number, the data following the <common-fields> is in the form detailed in <PCM-format-specific>, below. In the case of <PCM-format-specific>, the number indicates the encoding of the waveform data. The format WAVE_FORMAT_PCM (0x0001) is defined to be Microsoft Pulse Code Modulation (PCM) format. The MMA will maintain a registry of supported wFormatTags.
wChannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo. DLS-1 supports only mono data (value = "1").
dwSamplesPerSec	The sampling rate (in samples per second) at which each channel should be played.
dwAvgBytesPerSec	The average number of bytes per second at which the waveform data should transferred. Playback software can estimate the buffer size using this value.
wBlockAlign	The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of wBlockAlign bytes of data at a time, so the value of wBlockAlign can be used for buffer alignment.

The <PCM-format-specific> struct:

Field	Description
wBitsPerSample	Specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel. Mobile DLS supports only 8 or 16 bit samples. If this field is not needed, then it should be set to zero.

The <WAVE-format-extensible-specific> struct:

Field	Description
wBitsPerSample	Specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel. Mobile DLS supports only 8 or 16 bit samples. If this field is not needed, then it should be set to zero.
cbSize	Specifies the size in bytes of the extra information in the <WAVE-format-extensible-specific> structure not including the size of wBitsPerSample and cbSize . cbSize must always be set to at least 22.
wValidBitsPerSample	Specifies how many bits are used per sample. If the wValidBitsPerSample is less than wBitsPerSample, then the actual PCM data is aligned left and all extra bits are at the least significant part of the word.
wSamplesPerBlock	Specifies how many samples are stored in one compressed block. wSamplesPerBlock is used for formats with fixed number of samples per block. If wSamplesPerBlock is zero, a variable amount of samples is contained in each block of compressed audio data.
wReserved	If neither wValidBitsPerSample or wSamplesPerBlock apply to the audio data being described by the <WAVE-format-extensible-specific> structure, set the wReserved field to zero.
dwChannelMask	The field dwChannelMask indicates which channels are present in the multi-channel stream.
SubFormat	The SubFormat field is set to the GUID that specifies the type of encoded waveform data described by the <WAVE-format-extensible-specific> structure. The MMA will maintain a registry of supported GUIDs.

2.16.2 Data Chunk <data-ck>

The <data-ck> contains the waveform data. It is defined as follows:

```
<data-ck>    →    data( <wave-data> )
```

The <data-ck> chunk:

Field	Description
<wave-data>	This is the waveform data encoded in the form described above in <common-fields>, wFormatTag and <WAVE-format-extensible-specific>, SubFormat. For PCM encoded waveform data, see Data Packing for WAVE_FORMAT_PCM Files and Data Format of the WAVE_FORMAT_PCM Samples sections.

[END]

Comments:

The RIFF structure of the file has not changed. Existing (pre-DLS 2.2) parsers can read DLS 2.2 files, as well as files written for Mobile DLS. Small differences in behavior resulting from format incompatibilities should only be observed in rare circumstances.

Date of issue: April 26, 2006

RP#: 025/Amd2

Related item(s): RP-025 (DLS 2.0); RP-025/Amd1 (DLS 2.1); RP-016 (DLS 1.0)