Downloadable Sounds Level 1

Version 1.1b
September 2004

Published By:
The MIDI Manufacturers Association
Los Angeles, CA

PREFACE

This document and the underlying specification is the result of extensive research and discussion by a group of hardware and software manufacturers to determine a base-line architecture which will support playback of "downloadable" sounds. It was the goal of this group to determine an architecture that would be similar enough to the majority of existing products to allow fast time to market, without sacrificing performance. The specification is particularly aimed at multimedia computing applications and platforms.

The discussion and drafting of this specification occurred in meetings and via email over a one year period under the auspices of the Interactive Audio Special Interest Group's Downloadable Sounds Working Group (IASIG DSWG). Members of this group also met at various times as the Downloadable Sounds Device Architecture Working Group (DSDAWG) and the Downloadable Sounds API and File Format Working Group (DSAFFWG). A complete list of DSWG members is on record with the IASIG.

In order to improve audio performance on multimedia platforms beyond today's capabilities, the specification includes guidelines for a Level 2 feature set, which can not be supported on much of the current installed base but is more in-line with what is commonly considered state of the art today. Development of the Level 2 specification will be an on-going process.

This document is in two parts. The first part describes the playback device architecture, and how device parameters will be defined. The second part describes a standard file format for distributing these samples. Recommendations are also planned for the protocol and messages to be used in creating standardized processes for initiating and managing downloading, so that applications, devices, and platforms can all be compatible.

DLS playback device must be able to accurately, within specific guidelines, play back files properly authored to this specification. **Certification of DLS compatibility is therefore required of all Manufacturers of DLS compatible devices and related products.** Companies should contact the MMA to receive information about testing procedures, and obtain test materials and certification.

The MMA, IASIG, and all members shall not be held liable to any person or entity for any reason related to the adoption, implementation or other use of this document or the specification herein.

Note: Version 1.1b contains only one change from 1.1.a, which is the removal of "Expression" from the list of controllers that are not reset when RAC = "0". This is an editorial correction, since all known implementations already worked this way.

Printed 2004

MMA
PO Box 3173
La Habra CA 90632-3173

Contents

Figures

Tables

Chapter 1: Device Architecture

Introduction

As wavetable (WT) synthesis architectures have become increasingly prevalent, the need for a standard format for defining musical instruments has grown in proportion.

Although the General MIDI (GM) specification for 128 instrument presets helps to a small degree, it lacks both the depth and breadth to deliver a truly consistent playback experience across a wide range of platforms.

There is a need for a musical instrument standard that allows composers to define exactly how each musical instrument sounds on a wide variety of playback devices.

The Problem

When authoring for MIDI, a composer faces two limitations:

1. General MIDI provides a very limited set of 128 instruments.
2. Even within the predefined GM set, there isn't enough consistency. This inconsistency ranges from subtle to outright ridiculous. A piano on one card may sound like an organ on another.

Other media do not suffer from these problems. An author of graphics or wave audio can rely on consistent results on multiple hardware solutions. Yet there is no such standard for MIDI. Because of this, many content providers have opted for digital audio (which delivers an exact digital recording of a performance) in order to resolve the common playback experience problem.

Unfortunately, digital audio is a very inflexible medium. It is poor at delivering interactive music solutions. Additionally, its storage requirements are orders of magnitude greater than that of MIDI. This latter point is of particular significance in places where data bandwidth is at a premium, for example on the Internet.

The Solution

The Downloadable Sounds Level 1 Architecture enables the author to completely define an instrument by combining a recorded waveform with articulation information. An instrument designed this way can be downloaded onto any hardware device that supports the standard and then played like any standard MIDI synthesizer.

Together with MIDI, it delivers the following benefits:

- A common playback experience, unlike GM.
- An unlimited sound palette for both instruments and sound effects, unlike GM.
- True audio interactivity, unlike digital audio.
- MIDI storage compression, unlike digital audio.

With the industry-wide adoption of this standard for downloadable sounds, MIDI will blossom as a superior solution for quality interactive music on the personal computer.

To achieve this end, a consortium of wavetable solution vendors has hammered out a recommendation for an industry standard for downloadable instruments. This specification is the end result of that effort.

This specification is intended to be a base level specification. It defines synthesis attributes that can be implemented today with the majority of off-the-shelf sound cards and chip sets. As such, we refer to this as the Level 1 specification, with the intention that a Level 2 specification will follow in the future. Level 2 will represent a richer set of capabilities that may not be attainable with many of today's products.

Objectives

The Level 1 Downloadable Sounds specification is designed to meet several objectives:

1. The specification must work for all or most current hardware solutions and computer platforms that support downloadable instruments.
2. The specification must be precise enough that it can guarantee a common playback experience. A specific combination of a MIDI sequence and downloadable instruments should sound extremely consistent from platform to platform.
3. The specification must be extensible so it can grow with time. Although the Level 1 specification must work with existing hardware, the Level 2 specification should anticipate and even help define silicon solutions currently in the design stage that represent the next generation of wavetable hardware.
4. The specification must be extensible to naturally support streaming of digital audio and real-time sound downloads. Even if these features can not be supported now, the design must not preclude their support in the future.
5. The specification must be open and non-proprietary.

The first objective is the most important. A standard that cannot be implemented immediately across the board is destined for failure. Amazingly, achieving this consensus has been surprisingly painless. Years of industry experience and legacy have already defined a base level synthesis architecture consisting of several key components:

- A sampled sound source with loop points.
- A low-frequency oscillator to control vibrato and tremolo.
- Two simple envelope generators to define volume and pitch envelopes.
- Standard behavior in response to MIDI events such as Pitch Bend and Mod Wheel.

The Downloadable Sounds specification starts with this bare bones functionality and delivers an implementation that works well on any synthesis hardware that supports these components. The specification also looks to the future and takes an extensible approach that allows the configuration to grow.

Design Overview

A musical instrument defined by a sound sample is much more than a simple wave file. In addition to the actual sample data and associated loop information, the instrument must indicate under what circumstances each sample should be used, and how to modulate, or articulate, the sample as it plays.

A generic sample-playback synthesis architecture can be broken down into three distinct subsystems:

1. Control logic
2. Digital audio engine
3. Articulation modules and connections

Control Logic

The control logic receives a MIDI note event and determines which instrument should play the note, and, within that instrument, which sample and articulation combination to use.

Choosing the instrument requires little more than observing the MIDI channel number in the event and selecting the proper instrument accordingly.

Choosing the sample and articulation to use is not as simple. Almost all sampled synthesis architectures employ some method of organizing samples by note range across the keyboard. In addition, multiple samples can be used to represent different velocity ranges and multiple samples can be played at once to create a layered, richer sound.

Terms such as *layers*, *splits*, *blocks*, and *regions* are commonly used in synthesizer jargon to refer to the management of multiple samples. For the purposes of this specification, we refer to them as *regions*.

The Level 1 specification implements a bare bones approach with no velocity cross-switching and no layering.

Digital Audio Engine

The digital audio engine is certainly the most obvious part of the synthesizer. It is composed of a playback engine, or digital oscillator, and a digitally controlled amplifier.

The digital oscillator plays a sampled sound waveform, managing loop points within the waveform so the sound can play continuously if needed. And, as the note plays, it responds to changes in pitch, allowing for real time expression such as vibrato and pitch bend.

The digitally controlled amplifier modulates the loudness of the instrument. Importantly, this is used to control the amplitude shape, or envelope, of the note. However, it is also used for other types of real-time expression, such as tremolo.

Pitch and volume control of the oscillator and amplifier are critical because they define the shape of the sound as it plays, and allow it to be dynamically responsive in real time, giving the sampled instrument much more expression than the simple digital audio playback could ever provide. Real-time control of these parameters comes from modules in the Articulation section which generate a constant stream of changing pitch and volume to which the digital audio engine responds.

The digital audio path represents the journey the sound takes from the oscillator to the amplifier to the digital-to-analog converter (DAC). This path can optionally include additional modules, such as filters and effects devices, that process the sound as it flows from oscillator to DAC.

The Level 1 specification implements a simple digital audio engine composed of an oscillator, amplifier, and DAC. The oscillator supports looped as well as one-shot samples.

Articulation Modules and Connections

The *articulation modules* are a set of devices that provide additional control over the pitch and volume of the sample as it plays.

The articulation modules include low frequency oscillators (LFOs) to contribute vibrato and tremolo, envelope generators to define an overall volume and pitch shape to the sample, and MIDI real-time controllers, such as Pitch Bend and Mod Wheel, to give the music real-time expression.

Generally, these modules can be linked in different ways to provide different results. For example, an LFO might generate a sine wave which modulates the pitch of the sample for vibrato or the volume of the sample for tremolo. Modules can also receive as well as send control signals. An envelope generator might use key velocity to influence the attack time of the envelope.

Articulation modules can be configured in different ways, and this configuration is an important part of the instrument definition. In fact, the term *patch* as used for an instrument preset refers to the early days when the hardware modules in an analog synthesizer were “patched” together with cables, which routed signals from module to module.

The ability to configure the modules is important because it yields a flexible approach to synthesizer design. At the same time, it is important to define a base level configuration that can be supported by all hardware.

For these reasons, the Level 1 specification is relatively rigid. It defines a preset routing arrangement that is simple enough to be supportable by existing hardware. Fortunately, it is also a musically logical and useful configuration.

Significantly, the specification maps the routing onto a flexible system, so it can grow into Level 2 and beyond.

It is important to understand that this is purely a symbolic system that can be mapped on many different hardware designs. Do not think of it as the recipe for a specific method for building synthesis architectures. It should be flexible enough to provide a common language between different hardware implementations.

Now that we have a reasonable overview of the three primary components of the synthesis architecture, we can investigate each in much greater detail, with attention to the Level 1 specifications.

Control Logic

The control logic implementation in the Level 1 specification is relatively simple. The control logic receives bank select and program change commands to select the instrument on a particular channel, and MIDI note on and off events to play notes on the same channel.

The control logic uses the combination of instrument choice (bank select and program change) and note to select a specific configuration of the articulation modules and digital audio engine to perform the note.

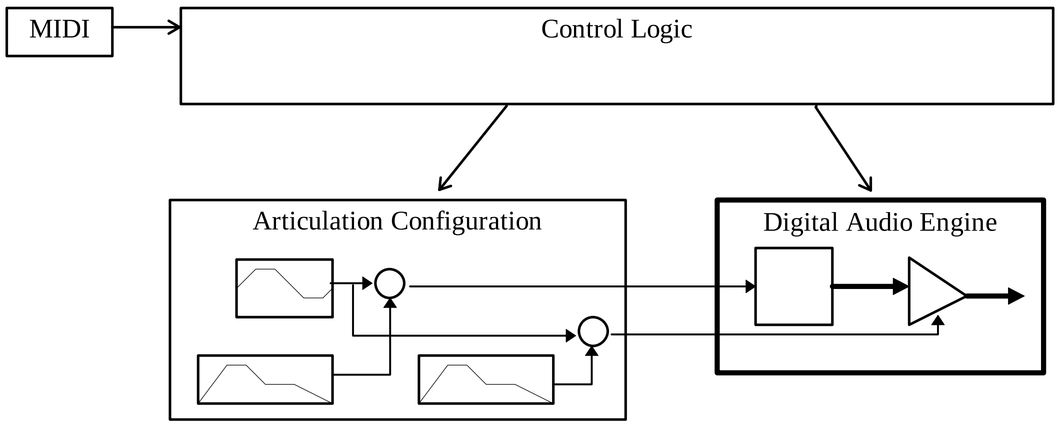


Figure 1. **Level 1 Device—Block Diagram**

For the digital audio engine, the control logic must select a specific sample to play. The sample is indirectly accessed through a *region*. The region defines the key range and velocity range used by the control logic to select the sample. It also determines a preset value for the overall amplitude of the sample and a preset tuning. The sample, in turn, defines the actual chunk of digitized sound along with control parameters such as loop points, sample rate, and so on.

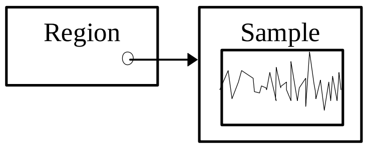


Figure 2. **Region**

The articulation is a complete configuration of articulation modules and their connections, including the envelope generators and LFO. These define how the note should be articulated as it plays.

To facilitate compatibility with existing synthesizers, Level 1 imposes the following limitations:

- There are two distinct instrument types: Melodic and Drum Kit.
- There can be only one Drum Kit and its use is restricted to MIDI channel 10.
- The Level 1 synthesizer is not required to support the MIDI Bank Select messages CC0 and CC32. Therefore, on some synthesizers, the minimum number of instruments could be restricted to 128 melodic instruments (including GM instruments for those devices that also support GM). If the Level 1 synthesizer does not support Bank Select, the Level 1 Device Driver must virtualize the Bank Select messages by mapping Bank Select messages sent by the application to the device's address space.

The primary difference between Melodic and Drum instruments is how they pair up articulations and regions. Melodic instruments usually require anywhere from one to a dozen or more samples to define the same sound over the range of the keyboard. Since it is all one instrument, only one articulation configuration is required. On the other hand, a drum kit by nature defines a complete cast of different instruments, so each drum must be treated individually.

Melodic Instrument

A Level 1 melodic instrument is composed of one global set of articulation parameters and up to 16 regions.

Each region defines a specific range of keyboard notes that it can play. Therefore, a melodic instrument uses up to sixteen keyboard splits to cover the pitch range of the sound.

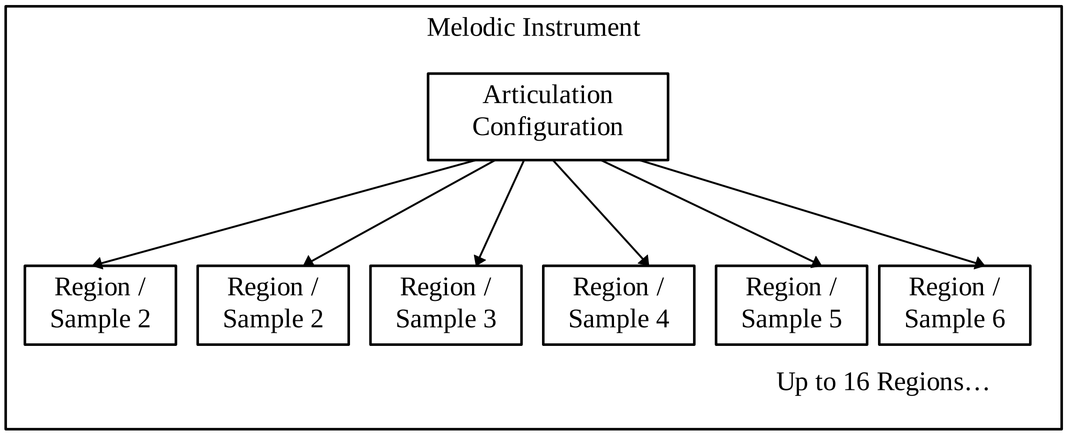


Figure 3. **Melodic Instrument Hierarchy**

Although the region structure can also define a velocity range, the Level 1 specification does not support velocity cross-switching.

Because there is only one articulation configuration per instrument, all of the samples (up to 16) are played against one global set of articulation data (envelopes, LFOs, etc.).

Drum Kit

Drum kits comprise a complete set of different sounds. Each drum requires a unique sample and requires its own articulation configuration.

A Level 1 drum kit contains up to 128 regions, one for each drum note. It also includes up to 128 articulation configurations, also one for each drum note.

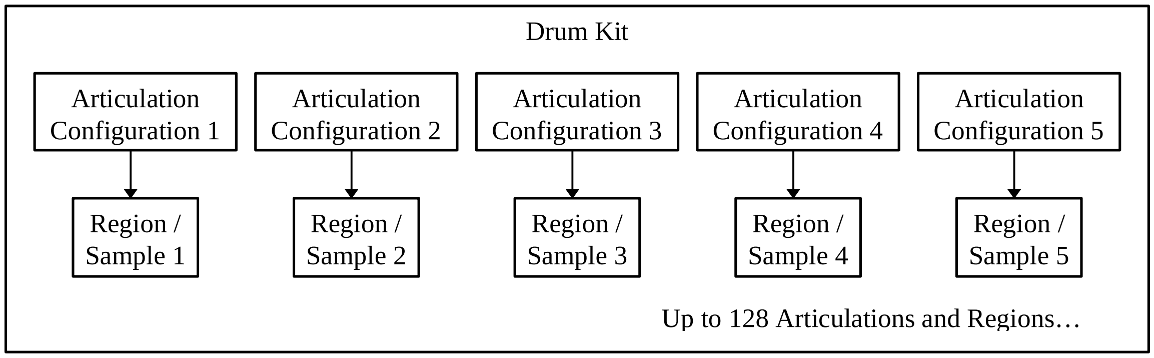

Drum Kit
Articulation Configuration 1
Articulation Configuration 2
Articulation Configuration 3
Articulation Configuration 4
Articulation Configuration 5
Region / Sample 1
Region / Sample 2
Region / Sample 3
Region / Sample 4
Region / Sample 5
Up to 128 Articulations and Regions…

Figure 4. **Drum Kit Hierarchy**

Each articulation configuration maps to a unique region, which in turn points to a specific sample. The region can define a single key or a contiguous range of keys. The sample does not necessarily have to be unique—several regions can point to the same sample.

Only one drum kit is allowed at a time. When a device supports both Level 1 and General MIDI, the device may share MIDI channels between Level 1 and General MIDI. In such a device, it will not be possible to use the General MIDI drum kit and the Level 1 drum kit simultaneously.

Level 2 should remove the distinction between drum kits and melodic instruments. All instruments should support 128 regions, etc. However, the wide range of existing hardware implementations cannot support such a generalized requirement.

Note Exclusivity

The Level 1 Device has provisions for two forms of note exclusivity on a MIDI channel basis. The first form of exclusivity involves notes on a MIDI channel with the same MIDI note number. By default, if a MIDI Note-On event is received and there are oscillators previously assigned to the same MIDI note and MIDI channel which have not received a Note-Off event, the control logic will issue a Note-Off to those oscillators. However, there is a flag in the usKeyGroup 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 trigger Note-Off events for oscillators assigned to the same MIDI note and MIDI channel. When the Non-Self-Exclusive flag is not set and a second Note-On event of the same MIDI note number on the same MIDI channel is received by the synthesis engine, the second Note-On will be played as well as the first. 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, a Note-Off event is issued to the other oscillators. 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 bank to stop a squeaking door sound when the door finally closes.

For Level 1, this is implemented as a usKeyGroup with a range of 0 to 15. usKeyGroup equal to 0 indicates a non exclusive key group. For Level 1, this mutually exclusive mode is limited to Channel 10.

Voice Allocation

Voice Allocation is the means by which digitally controlled oscillators are allocated to play samples as dictated by the instrument parameters and the MIDI data stream. Ideally, there should be enough to play every note and sound effect in the score. However, this may not always be the case. Some Level 1 devices may have more oscillators than others, and the developer may choose to develop a score that uses more than the minimum amount of polyphony to take advantage of this. This leads to the case of what to do when the score calls for more oscillators than the device is capable of producing. We would prefer to see a graceful degradation where the developer can accurately predict the end result.

To satisfy this need, the Level 1 device defaults to static MIDI channel priority. In this scheme, each MIDI channel is assigned a priority, with the drum channel (MIDI channel 10) given the highest priority, followed by the remaining channels in ascending numeric order (10,1-9,11-16).

When a Note-On event is received, the Control Logic must first attempt to satisfy the request using a free oscillator. If all oscillators have been previously allocated, then an oscillator must be “stolen”; that is to say, the previous note is silenced, and a new note is begun. Using static MIDI channel priority, an oscillator assigned to a note on channel 16 would be stolen before one on channel 15, and so on. Notes on lower priority channels cannot steal oscillators assigned to notes on higher priority channels under any circumstances. For example, if a Note-On is received on channel 11, and all oscillators are currently assigned to notes on channels 10 and above, then the new note is not played. *This logic applies even to notes in the Release phase, i.e. after a Note-Off has been received.*

The designer of a Level 1 device is free to use any other voice allocation heuristic, provided that the channel priority heuristic is satisfied first. This flexibility allows for further prioritization within the channel, such as released notes vs. unreleased notes, “inside” versus “outside” voices, or any other scheme that the developer cares to implement.

The DLS device will be in static MIDI channel priority by default (or upon entering DLS mode if equipped for multiple modes of operation), and when it receives the DLS System On message. This static MIDI channel priority mode can also be enabled or disabled through a DLS System Exclusive message. To turn off static MIDI channel priority, a programmer would send the DLS Voice Allocation Off System Exclusive message (F0h 7Eh <device ID> 0Ah 03h F7h). As is common for this type of MIDI Message, <device ID> = 7Fh means the message is intended for all DLS devices in the system. To restore static MIDI channel priority, a programmer can send the DLS Level 1 Voice Allocation On System Exclusive message (F0h 7Eh <device ID> 0Ah 04h F7h), or reset the device with the DLS System On message. DLS Voice Allocation System Exclusive messages have no effect on a DLS device which is not currently in DLS mode. Please see the MIDI Specification for more details on System Exclusive Messages.

When static MIDI channel priority is turned off, device manufacturers may use any voice allocation heuristic, but consistent playback on any DLS Level 1 device can not be guaranteed except in static MIDI channel priority mode. Applications which modify the voice allocation mode should always return it to the default state upon exiting.

While Level 1 does not allow for layering of oscillators for instruments, a Level 1 device that also supports General MIDI may use layering to implement its GM instruments. The GM instruments should use the voice allocation heuristics described here when the device is operating in Level 1 mode. However, it is left to the discretion of the device designer to determine the heuristic for the second layer of a General MIDI instrument. For example, the device designer may elect to steal the second layer of a GM instrument to satisfy a Note-On event, regardless of the priority of the channels.

Level 2 should add a dynamic prioritization scheme that allows the developer to assign priority to individual notes as they are played.

Bank Select and Program Change

The Level 1 device will support Bank Select and Program Change MIDI messages as the method of selecting the instrument to be played on a MIDI channel. Because some Level 1 devices may not support Bank Select internally, the designer has the option of providing this support in the device driver by virtualizing the Bank Select messages. In this case, a DLS Level 1 is required to support only a single melodic instrument bank, and each downloaded instrument may go into this single bank. This mechanism is defined in the DLS Level 1 Protocol and Messaging Guidelines using the Download with Lock command (see Reference).

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.

DLS Level 1 specifies that a downloadable sound can be placed in any bank/instrument location in MIDI space. To avoid conflicts with Roland's GS, Yamaha's XG and any other manufacturer's MIDI bank mapping scheme, programmers are required to send the DLS System On (F0h 7Eh

<device ID> 0Ah 01h F7h) message prior to using the DLS device. This message will enable a downloaded instrument to be loaded into any bank/instrument location in MIDI space and enable the device to operate as a DLS device. If a download is to a manufacturer's proprietary MIDI bank/instrument location, the device is required to play back the downloaded sound and not the manufacturer's proprietary sound. When the downloaded sound is removed from the MIDI bank/instrument location, the device should again play back the manufacturer's proprietary sound. All manufacturers of DLS Level 1 devices are ***required*** to support this message. To turn DLS functionality off, send the DLS System Off message (F0h 7Eh <device ID> 0Ah 02h F7h). Sending a <device ID> = 7Fh will be a broadcast message to all DLS devices.

To detect whether a given instrument is a drum or melodic instrument, a program should check the instrument header's ulBank field bit 31 in the DLS Level 1 file format. If ulBank bit 31 is equal to 1, the instrument is a drum instrument. If ulBank bit 31 is equal to 0, the instrument is a melodic instrument. The distinction between drum and melodic instruments will cease to exist in a future revision of the DLS specification.

Digital Audio Engine

The Level 1 digital audio engine is relatively simple. Other than the digital oscillator and digitally controlled amplifier, there are no other modules on the digital signal path.

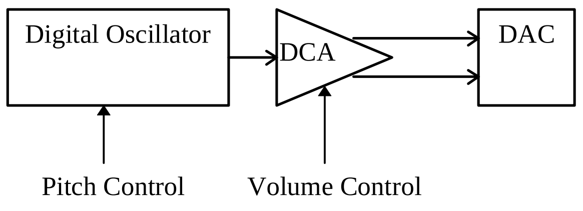


Figure 5. **Digital Audio Engine—Block Diagram**

Digital Oscillator

The digital oscillator is handed a sample to play by the control logic. It performs the sample, all the while modulating the pitch of the sound with the control stream from the articulation set.

The sample data can be 16-bit two's complement or 8-bit offset two's complement (often referred to as 8-bit unsigned data, with zero represented as 0x80). This complies with the both the RIFF WAVE PCM format and common practice on the Macintosh. To convert between offset two's and standard two's complement (signed 8 bit, -128 to +127), XOR each byte of the source sample value with 0x80.

The sample defines whether it is to be played once through or looped. If looped, the sample supports one looped region, as defined by a pair of fixed sample data point indices. The loop start specifies the first sample data point in the loop, while the loop end specifies the first sample data point beyond the end of the loop. There is no fractional component to the loop points.

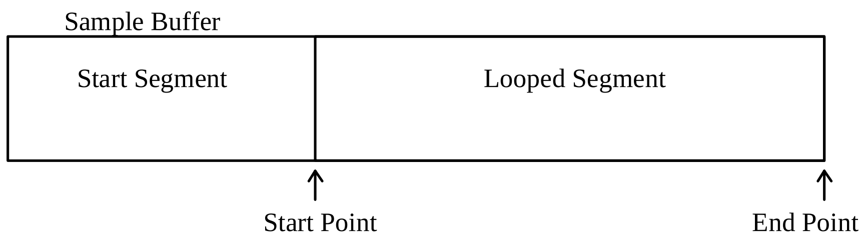


Figure 6. **Sample**

The sample structure also defines the MIDI note number and fine tune for the recorded instrument. For example, the sample might be a recording of a C# octave 4 piano key that's just a little flat.

It also defines the sample rate of the recording. Between this and the tuning information, the digital oscillator has everything it needs to pitch shift the sample appropriately for the intended pitch.

The Level 1 digital oscillator must be capable of pitch shifting the sample with a range of up two octaves and down four octaves. This pitch shift is the sum total of all tuning parameters, including sample fine tune and MIDI pitch bend. And the Level 1 digital oscillator is required to operate at a minimum playback rate of 22.05 kHz and is required to support a minimum of 24 simultaneous voices.

Level 2 should address issues such as extensions for additional loop types, fractional loop indices, data compression, higher maximum output sampling rate, and polyphony greater than 24 voices.

Digitally Controlled Amplifier

The second module in the signal path is the digitally controlled amplifier (DCA). It takes the output of the oscillator and modulates the volume, responding to controls from the articulation section, of which the amplitude envelope is probably the most important.

The MIDI pan signal is fed through a transform that converts to left and right amplitude signals. These signals are fed separately into the DCA to generate stereo output from the monophonic signal entering the DCA.

Level 2 should address additional capabilities such as low-pass filtering and digital effects (reverb, chorus, etc.) which can be represented by modules in the digital signal path.

Articulation Modules and Connections

The Level 1 set of Articulation modules include an LFO, two envelope generators, and several MIDI controller inputs. A configuration is defined by connecting this base set of modules and setting the controls for each one. The Level 1 set specifies the controls available on each module and a required set of connections to link the modules together.

Low Frequency Oscillator

The LFO generates a periodic waveform that is used to modulate the pitch or volume of the instrument. Level 1 requires just one shape —a sine wave—although a triangle wave may be substituted.

NOTE: When applied to pitch, the LFO will vary the pitch of the oscillator in a bipolar fashion. For example, if the *lScale* value for the LFO to Pitch connection is set to 50 cents, then the oscillator pitch will vary from +50 cents to –50 cents.

When applied to attenuation, the LFO will vary the attenuation of the DCA in a unipolar fashion. For example, if the *lScale* value for the LFO to Attenuation connection is set to 3dB, then the DCA output will vary from 0dB to –3dB.

Frequency

The frequency control sets the wave frequency between 0.1 Hz and 10 Hz.

Start Delay Time

The LFO starts after a preset delay time. If it cannot synchronize the wave cycle to start with a 0 bias, it should ramp up to full volume over a one cycle period.

The delay time control sets the initial delay within the 10 ms to 10 second range.

Level 2 should consider multiple wave shapes, greater frequency ranges, and definable ramps.

Envelope Generator

The envelope generator creates a four-section segmented envelope. The segments correspond to the attack, decay, sustain, and release sections of the traditional analog ADSR (Attack, Decay, Sustain, Release) design.

When the note starts, the envelope enters the attack segment, increasing until it reaches the peak level. It then enters the decay segment and decreases to a steady level—the sustain level—which holds until a MIDI note off is received. When the note ends, the envelope enters the fourth segment, the release section, and decays out.

The Level 1 scenario accommodates one-shot sample files that require no envelope. This is accomplished by setting the attack duration to 0 and sustain and release values to maximum.

For design consistency, the signal levels from the envelope generator are represented in an exponential format, that is, dB for amplitude. A linear change on an exponential scale generates an exponential curve. As a result, a convex curve translates into a linear curve and a linear curve translates into an exponential curve. The connection graph manages the transform which converts from linear to exponential (more on that later).

Decay and Release Time Constants are actually rates defined as the time for the signal to decay from full scale to -96 dB, at which point it is assumed that most 16-bit hardware makes an abrupt transition to infinite attenuation.

The following graph shows the output of a typical envelope generator plotted in linear amplitude. The dashed line shows the method used to calculate actual decay time. Release times use an identical method, calculated as if the Sustain Level were set to 100%.

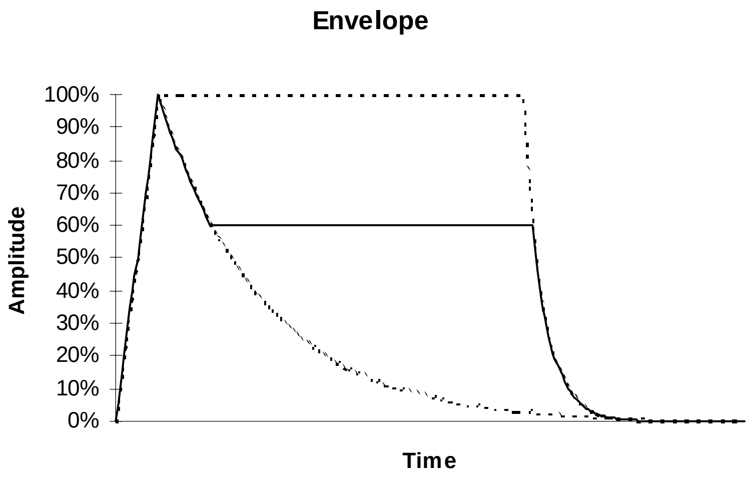


Figure 7. **Amplitude Envelope Overview**

There is no industry consensus as to the correct curve shape of the attack segment of the pitch envelope. Some use the same convex (logarithmic) shape as the amplitude envelope, resulting in a linear attack (in hertz) when used to control the phase increment (an exponential transform). Others use a linear shape resulting in an exponential attack when used to control the phase increment.

Experiments have not found any perceptible difference within the one-octave bounds of the pitch envelope attack when playing one or the other method. However, in order to provide a clear guideline for future development of DLS level 1 products, a single pitch envelope curve should be specified. Therefore, the attack segment for new products should be exponential, though existing products which provide a linear curve shape will be acceptable for DLS level 1 certification.

The graphs depicting various portions of the envelope generator on the following pages do not accurately depict the linear and exponential segments of the curves. Note that the amplitude attack segment is linear, while the decay and release segments of both pitch and amplitude are exponential.

EG Attack Time

The attack time determines how long the attack segment lasts from the start of the note until the envelope hits peak level. The amplitude envelope attack segment is linear. (This is implemented as a convex curve that is passed through a linear to exponential transformation, resulting in a linear curve.) The default pitch envelope attack segment is an exponential curve. (This is implemented as a linear curve that is passed through a linear to exponential transformation, resulting in an exponential curve.)

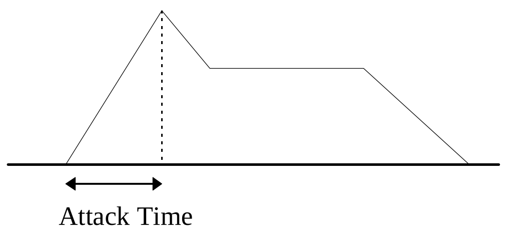


Figure 8. **Attack Time**

EG Decay Time

The decay time determines the duration of the decay segment. However, it represents the time for a full decay from peak level to a sustain level of 0 (effectively -96 dB). As a result, it can also be thought of as the decay rate. The decay segment's shape is linear on a dB scale which ultimately transforms into an exponential curve.

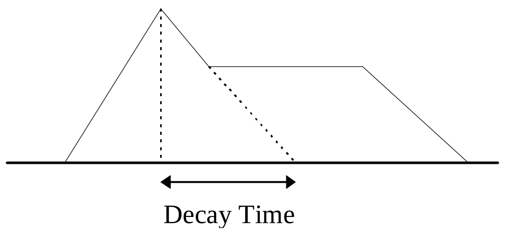


Figure 9. **Decay Time**

EG Sustain Level

The Decay segment ends when the Sustain Level is reached and the note sustains at this level until the note off is received. Raising the Sustain Level will shorten the Decay segment and lengthen the Release segment, while lowering it will have the opposite effect. Sustain is defined as a percentage of the envelope peak in 0.1% increments.

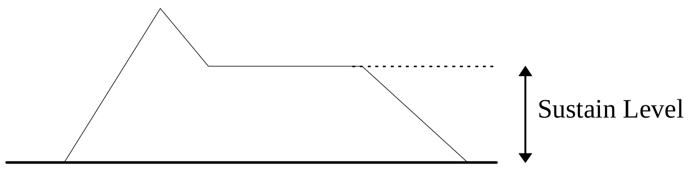


Figure 10. **Sustain Level**

EG Release Time

The release time constant determines the duration of the release segment. Like the decay value, the time is measured for full release from peak value to 0 (-96 dB) instead of from the sustain level. So, the actual duration is less for a starting point that is below peak.

The release segment's shape is linear on a dB scale.

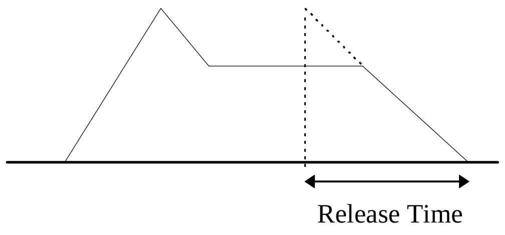


Figure 11. **Release Time**

EG Velocity to Attack Scaling

The MIDI velocity scales the duration of the attack segment.

Decay Scaling

The MIDI Key number scales the duration of the decay segment. For example, increasing the note number could decrease the duration of the decay, just as a piano note decays longer for lower notes and faster for higher notes.

EG Polarity

Polarity refers to whether the envelope runs positive or negative around zero. This is only applicable to using the envelope for pitch control. This is accomplished with the EG2 Pitch Range connection block. *Level 2 should explore additional segments for the envelopes as well as a standard one-shot envelope mode.*

Performance Controllers

MIDI performance data serve as control sources.

Key Number

The MIDI Key Number normally determines the playback frequency of the instrument. The default connection is to map a Equal Temperament 12-tone (ET-12) scale to the keyboard, such that the difference between each key is an ET-12 semitone. However, it is sometimes useful to change the default key scaling so that the pitch does not change across the keyboard. The Level 1 Device architecture allows the Key Number to Pitch connection to be specified with a scalar of zero, which means that the Key Number will have no effect on the pitch.

Level 2 should provide for other equal temperament tunings by allowing other values for this connection.

Volume

Continuous MIDI Volume events are connected by default to the volume summing node to set the volume. The MIDI Volume input is converted to attenuation in dB by the Concave Transform according to the following formula:

$$atten_{dB} = 20 \times \log_{10}\left(\frac{127^2}{Volume^2}\right)$$

The transform to convert from MIDI volume value to attenuation is managed by the connecting graph (see next section). A plot of the Concave Transform can be found in Appendix B.

Expression

Continuous MIDI Expression events are connected by default to the volume summing node to contribute to the volume. The MIDI Expression input is converted to attenuation in dB by the Concave Transform according to the following formula:

$$atten_{dB} = 20 \times \log_{10}\left(\frac{127^2}{Expression^2}\right)$$

The transform to convert from MIDI Expression to attenuation is managed by the connecting graph (see nest section). A plot of the Concave Transform can be found in Appendix B.

Velocity

The MIDI Note Velocity value is converted to attenuation in dB by the Concave Transform according to the following formula:

$$atten_{dB} = 20 \times \log_{10}\left(\frac{127^2}{Velocity^2}\right)$$

and fed to control either the volume or envelope generator peak level. A plot of the Concave Transform can be found in Appendix B, Figure 18.

Pan

The MIDI Pan event is fed directly to the digitally controlled amplifier where it is used to set the pan position in the stereo field. An equal power equation is used to define the distribution from left to right. The equation for individual channel attenuation in dB is given by the following formula:

$$left\,channel\,atten_{dB} = -20 \times \log_{10}\sqrt{\frac{127-x}{127}}$$

$$right\,channel\,atten_{dB} = 20 \times \log_{10}\sqrt{\frac{x}{127}}$$

A graph of left and right channel output plotted against the MIDI Pan controller can be found in Appendix B, Figure 19.

Mod Wheel

Level 1 specifies the use of the Mod Wheel control to set LFO depth, with separate level assignments for LFO pitch modulation and LFO volume modulation.

Pitch Bend

The MIDI Pitch Bend event is routed directly to the pitch summing node, scaled to the GM standard. Registered Parameter Number data entry can be used to set the pitch range.

Sustain

Sustain Pedal MIDI events go directly to the control logic, which uses them in conjunction with note on and off events to determine which notes are on.

Registered Parameter Number Data Entry

Registered Parameter Numbers (RPNs) for remote data entry are supported, as per the General MIDI specification. The parameters include Fine Tune, Pitch Bend Range, and Coarse Tuning.

RPN 0 - Pitch Bend is used to alter the range of the pitch bend wheel. The Level 1 Device must support a maximum pitch bend range of at least 12 semitones.

RPN 1 - Fine Tuning is used to alter the pitch in fine increments of less than a semitone. The Level 1 Device must support the full range of +/-8191/8192nd of a semitone with resolution to the best of its ability. It is recommended that the device be capable of supporting resolution of at least 1 cent (1/100th of a semitone). The default setting is 0, which does not alter the pitch.

RPN 2 - Coarse Tuning is used to transpose by semitones. Coarse Tuning alters the MIDI note number before the articulation data, region, and sample selections are made by the Control Logic. This placement prevents unwanted timbre shifts in instruments that have been multisampled.

Level 1 Devices do not support the use of Coarse Tuning on MIDI Channel 10. The default setting is 0, which does not alter the pitch.

The Level 1 Device is required to support the Data Entry MSB and Data Entry LSB Controllers (MIDI Controllers 6 and 38, respectively), but not required to support the Data Increment and Data Decrement Controllers (MIDI Controllers 96 and 97, respectively).

Channel Mode Messages

The Level 1 Device must also support the following Channel Mode messages:

Reset All Controllers (MIDI Controller 121)

The Reset All Controllers message makes use of the data byte to allow for multiple levels of functionality. There are currently two data byte values defined. When the data byte is 0, the device will reset all controllers to their default values *except* the following controllers:

- Volume (Controller 7)
- Pan Control (Controller 10)

When the data byte is 127, the device will reset *all* controllers to their power-on default values. This should include all MIDI continuous controllers and RPNs. It is left to the discretion of the device designer whether to reset any NRPNs used by the device. However, the behavior should be documented.

All Notes Off (MIDI Controller 123)

This message performs a Note-Off event for all notes on the specified MIDI channel. If the Sustain Pedal (MIDI Controller 64) is active, the notes should continue to sustain until a Sustain Pedal release event is sent.

Power-on Default Values

- Volume (default = 100)
- Expression (default = 127)
- Pan Control (default = 64)
- Modulation Wheel (default = 0)
- Pitch Bend (default = 0, no pitch change)
- Sustain Pedal (default = 0)

Articulation Architecture

The Level 1 specification defines the set of modules that must be supported and it itemizes a list of required connections between the nodes of these modules. These connections define the flow of signals between modules and ultimately to the digital oscillator and DCA to control the pitch and volume of a note as it plays.

The connection between a source and destination node is usually more complicated than a direct link. In fact, a link is usually a combination of one or two sources: signal attenuation and transform information.

Transforms

In many cases, the input data to a module must be transformed into units that can be accepted by that module. For example, the linear output of an envelope generator needs to be converted into a format compatible with the DCA, typically an exponential transform. Likewise, the link between a MIDI volume input and the DCA must undergo a similar transform. To accomplish this, the connection mechanism allows for an optional transform to be defined for every connection.

Connection Block

Each connection can be viewed as a block that defines the usSource, lScale, usControl, optional usTransform, and usDestination.

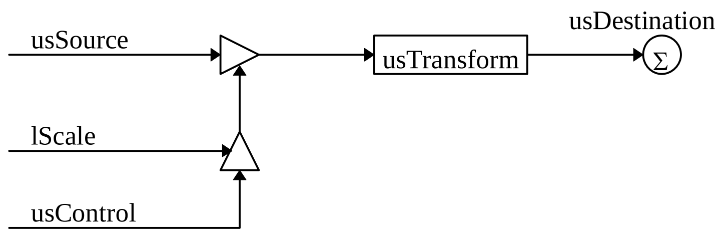


Figure 12. **Connection Block**

All connections to a single destination are summed together to produce the final input for that destination. The pseudo-code for a connection block is as follows:

```
usDestination = usDestination + usTransform(usSource * (usControl * lScale))
```

If the usSource or usControl are both set to SRC_NONE then both the inputs are taken to be a 1, and the other input to the multiply block is passed. All Connections must have a valid usDestination. Connection blocks can be broken down into three basic types: Scaled; Scaled Source; and Controlled Scaled Source.

Scaled connections are used to set the nominal level of a destination. A Scaled connection consists of a usDestination, lScale, and an optional usTransform. The usSource and usControl inputs are both set to SRC_NONE. The result is a constant signal, or bias, which is applied to the destination. An example of this is the LFO Frequency connection block, which sets the frequency of the LFO. The usDestination is DST_LFO_FREQ.

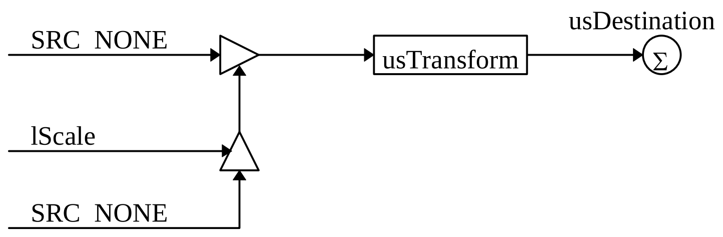


Figure 13. **Scaled Connection Block**

Scaled Source connection blocks scale the usSource input routed to the usDestination. A Source Scale connection block consists of usDestination, lScale, usSource, and an optional usTransform. The usControl input is set to SRC_NONE. The result is a signal which is modulated by the usSource input and applied to the usDestination. An example of this is the EG2 to Pitch connection block, the amount of modulation applied to pitch from the Pitch Envelope. The usDestination is DST_PITCH.

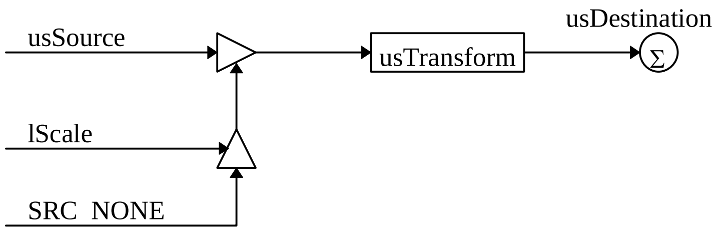


Figure 14. **Scaled Source Connection Block**

Controlled Scaled Source connection blocks use the usControl input to modulate or control the scaled usSource input routed to the usDestination. A Controlled Scaled Source connection consists of usDestination, lScale, usSource, usControl and an optional usTransform. The result is a signal which is modulated by both the usSource input and the usControl input. An example of this is the LFO Mod Wheel to Pitch connection block, which allows the Modulation Wheel (CC1) to determine the amount of LFO output applied to pitch. The usDestination is DST_PITCH.

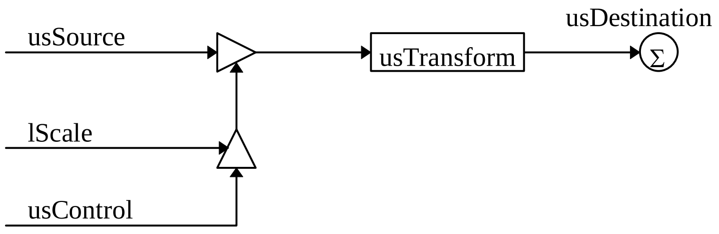


Figure 15. **Controlled Scaled Source Connection Block**

The parameters of each connection block are shown in the Connection Block Table.

Connection Block Sources

Level 1 Connection Block sources generate control signals for use by Connection Block destinations:

Generic Sources

CONN_SRC_NONE	No Source
CONN_SRC_LFO	Low Frequency Oscillator
CONN_SRC_KEYONVELOCITY	Key on Velocity
CONN_SRC_KEYNUMBER	Key Number
CONN_SRC_EG1	Envelope Generator 1
CONN_SRC_EG2	Envelope Generator 2
CONN_SRC_PITCHWHEEL	Pitch Wheel

MIDI Sources

CONN_SRC_CC1	Modulation Wheel
CONN_SRC_CC7	Channel Volume
CONN_SRC_CC10	Pan
CONN_SRC_CC11	Expression
CONN_SRC_RPN1	RPN1 - Fine Tune
CONN_SRC_RPN2	RPN2 - Coarse Tune

Note that the Gate signal is not included, because it is automatically required for all modules that use it. Likewise, the Sample Choice and Fine Tune outputs of the Note Routing logic are not standard modulation controls.

Connection Block Destinations

The Level 1 Connection Block destinations are:

Generic Destinations

CONN_DST_NONE	No Destination
CONN_DST_ATTENUATION	Attenuation
CONN_DST_PAN	Pan
CONN_DST_PITCH	Pitch

LFO Destinations

CONN_DST_LFO_FREQUENCY	LFO Frequency
CONN_DST_LFO_STARTDELAY	LFO Start Delay Time

EG1 Destinations

CONN_DST_EG1_ATTACKTIME	EG1 Attack Time
CONN_DST_EG1_DECAYTIME	EG1 Decay Time
CONN_DST_EG1_SUSTAINLEVEL	EG1 Sustain Level
CONN_DST_EG1_RELEASETIME	EG1 Release Time

EG2 Destinations

CONN_DST_EG2_ATTACKTIME	EG2 Attack Time
CONN_DST_EG2_DECAYTIME	EG2 Decay Time
CONN_DST_EG2_SUSTAINLEVEL	EG2 Sustain Level
CONN_DST_EG2_RELEASETIME	EG2 Release Time

Connection Block Controls

The Level 1 Connection Block controls are:

MIDI Controls

CONN_SRC_CC1	Modulation Wheel
CONN_SRC_RPN0	RPN0 - Pitch Bend Range

Connection Block Transforms

The Level 1 Connection Block destinations are:

Transforms

CONN_TRN_NONE	No Transform
CONN_TRN_CONCAVE	Concave Transform

Connection Graph

The Connection Graph illustrates all of the required articulation modules, connection blocks and MIDI controllers/inputs connected to the Control Logic and Digital Audio Engine portions. This is the necessary architecture to perform one voice in a Level 1 synthesizer.

The Control Logic module translates MIDI note and sustain events into region/sample choice and associated control signals, including Key (the note number), Velocity (note emphasis), and Gate. Gate defines the start and end times of the note, and is used to trigger modulators, such as the LFO and envelope generators. In a higher level implementation, Gate may also be used to set the loop status in the digital oscillator.

The digital oscillator unit plays the sample. It is primarily controlled by the Pitch Sum node which accumulates signals from modulation sources to deliver the final pitch for the instrument.

The digitally controlled amplifier takes the output of the digital oscillator and sets the volume. It is controlled by two signals: the Volume Sum node, which accumulates signals from modulation sources to deliver the final volume of the instrument, and the MIDI pan signal, which sets the stereo position of the stereo digital output.

Control signals from MIDI modulation sources, such as Volume and Pitch Bend, flow through connectors which use their attenuation values and transforms to define the range of the control. The Volume Sum node (DST_ATTENTION) can be looked at as a summing node if signals are represented in dB units, or as a multiplier if signals are represented in linear units. 0dB is defined as the maximum output level at the Volume summing node (DST_ATTENTION). For example, the output of the summing node expressed in terms of linear gain can be expressed as follows:

```
output = MIDI Volume * MIDI Expression * Velocity * EG1 * Sample Attenuation * AM
```

In the special case where the MIDI Mod Wheel signal controls the depth of the LFO output signal to either pitch or volume, the connector has an additional modulation input.

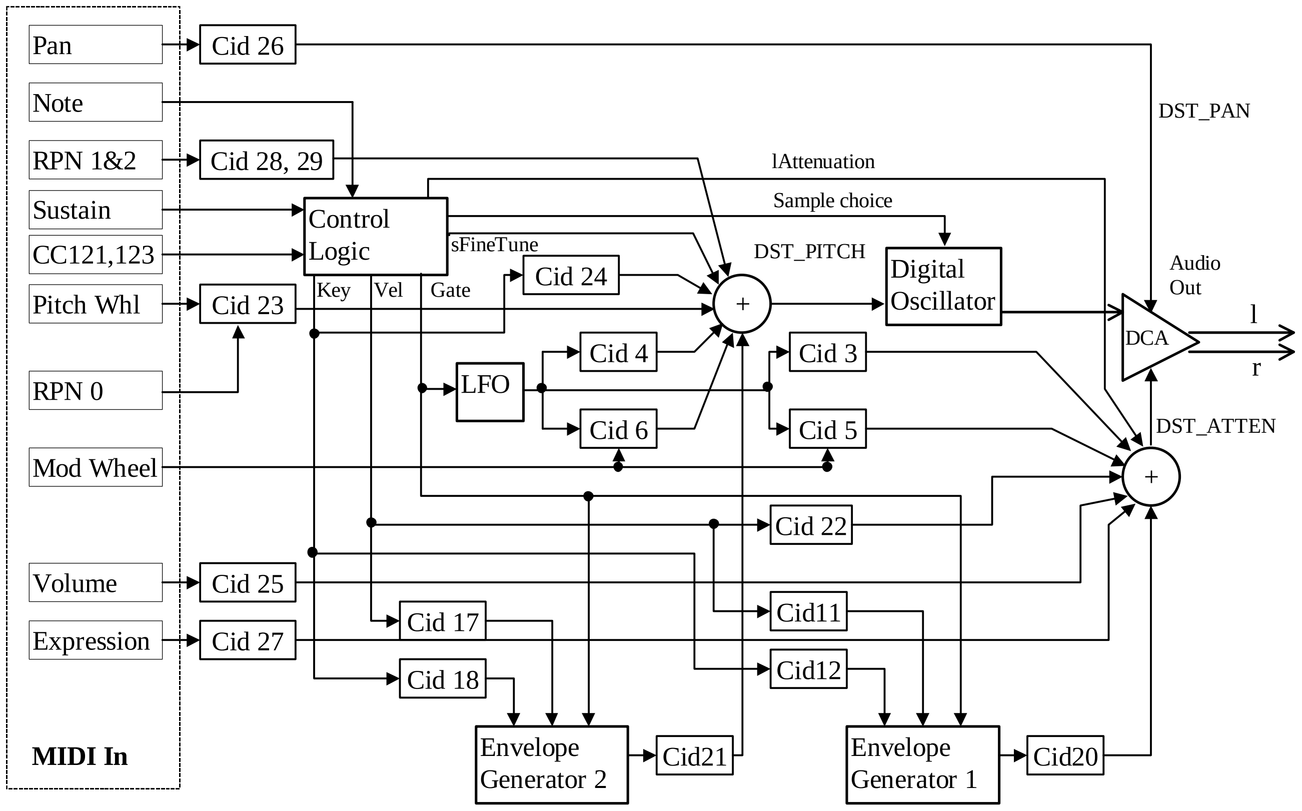


Figure 16. **Connection Graph**

(Note: “Cid#” = Connection Block ID number shown in Table 1)

Modulation Routing

The Level 1 synthesizer supports the connection blocks listed in the following table. Each connection lists a usSource, usControl, usDestination, and usTransform. Each is numbered to correspond with its Connection ID numbered box in the Connection Graph.

Cid#	Articulator Name	usSource	usControl	usDestination	usTransform
LFO Section					
1*	LFO Frequency	SRC_NONE	SRC_NONE	DST_LFO_FREQ	TRN_NONE
2*	LFO Start Delay	SRC_NONE	SRC_NONE	DST_LFO_DELAY	TRN_NONE
3*	LFO Attenuation Scale	SRC_LFO	SRC_NONE	DST_ATTENUATION	TRN_NONE
4	LFO Pitch Scale	SRC_LFO	SRC_NONE	DST_PITCH	TRN_NONE
5	LFO Modw to Attenuation	SRC_LFO	SRC_CC1	DST_ATTENUATION	TRN_NONE
6	LFO Modw to Pitch	SRC_LFO	SRC_CC1	DST_PITCH	TRN_NONE
EG1 Section					
7*	EG1 Attack Time	SRC_NONE	SRC_NONE	DST_EG1_ATTACKTIME	TRN_NONE
8*	EG1 Decay Time	SRC_NONE	SRC_NONE	DST_EG1_DECAYTIME	TRN_NONE
9*	EG1 Sustain Level	SRC_NONE	SRC_NONE	DST_EG1_SUSTAINLEVEL	TRN_NONE
10*	EG1 Release Time	SRC_NONE	SRC_NONE	DST_EG1_RELEASETIME	TRN_NONE
11	EG1 Velocity to Attack	SRC_KEYONVELOCITY	SRC_NONE	DST_EG1_ATTACKTIME	TRN_NONE
12	EG1 Key to Decay	SRC_KEYNUMBER	SRC_NONE	DST_EG1_DECAYTIME	TRN_NONE
EG2 Section					
13*	EG2 Attack Time	SRC_NONE	SRC_NONE	DST_EG2_ATTACKTIME	TRN_NONE
14*	EG2 Decay Time	SRC_NONE	SRC_NONE	DST_EG2_DECAYTIME	TRN_NONE
15*	EG2 Sustain Level	SRC_NONE	SRC_NONE	DST_EG2_SUSTAINLEVEL	TRN_NONE
16*	EG2 Release Time	SRC_NONE	SRC_NONE	DST_EG2_RELEASETIME	TRN_NONE
17	EG2 Velocity to Attack	SRC_KEYONVELOCITY	SRC_NONE	DST_EG2_ATTACKTIME	TRN_NONE
18	EG2 Key to Decay	SRC_KEYNUMBER	SRC_NONE	DST_EG2_DECAYTIME	TRN_NONE
Miscellaneous Section					
19*	Initial Pan	SRC_NONE	SRC_NONE	DST_PAN	TRN_NONE
Connections inferred by DLS1 Architecture					
20	EG1 To Attenuation	SRC_EG1	SRC_NONE	DST_ATTENUATION	TRN_NONE
21	EG2 To Pitch	SRC_EG2	SRC_NONE	DST_PITCH	TRN_NONE
22	Key On Velocity to Attenuation	SRC_KEYONVELOCITY	SRC_NONE	DST_ATTENUATION	TRN_CONCAVE
23	Pitch Wheel to Pitch	SRC_PITCHWHEEL	SRC_RPN0	DST_PITCH	TRN_NONE
24	Key Number to Pitch	SRC_KEYNUMBER	SRC_NONE	DST_PITCH	TRN_NONE
25	MIDI Controller 7 to Atten.	SRC_CC7	SRC_NONE	DST_ATTENUATION	TRN_CONCAVE
26	MIDI Controller 10 to Pan	SRC_CC10	SRC_NONE	DST_PAN	TRN_NONE
27	MIDI Controller 11 to Atten.	SRC_CC11	SRC_NONE	DST_ATTENUATION	TRN_CONCAVE
28	RPN1 to Pitch	SRC_RPN1	SRC_NONE	DST_PITCH	TRN_NONE
29	RPN2 to Pitch	SRC_RPN2	SRC_NONE	DST_PITCH	TRN_NONE

Table 1 - Connection Block Table

*(*Note: Scaled connection blocks are not shown on the connection graph.)*

The articulation configuration is designed to expand into future specification levels in two ways. First, each module defines a base level set of controls and connectors. We expect most modules will expand as more controls and connections are added in the future. Secondly, the number of connection blocks will also grow as we add more articulation modules, control inputs, and required connections.

The following table shows the Minimum, Maximum and Unit values for each connection block of the DLS Level 1 synthesis architecture.

Articulator	Default Value	Min Value	Max Value	Units
LFO Section				
LFO Frequency	5 Hz	0.1 Hz	10 Hz	32 bit pitch cents
LFO Start Delay	0.01 Secs	0.01 secs	10 secs	32 bit time cents
LFO Attenuation Scale	0 dB	0 dB	12 dB	32 bit gain (cB)
LFO Pitch Scale	0 cents	-1200 cents	1200 cents	32 bit pitch cents
LFO Mod Wheel to Attenuation	0 dB	0 dB	12 dB	32 bit gain (cB)
LFO Mod Wheel to Pitch	0 cents	-1200 cents	1200 cents	32 bit pitch cents
EG1 Section				
EG1 Attack Time	0 secs	0 secs	20 secs	32 bit time cents
EG1 Decay Time	0 secs	0 secs	40 secs	32 bit time cents
EG1 Sustain Level	100 %	0%	100%	0.1 % units
EG1 Release Time	0 secs	0 secs	20 secs	32 bit time cents
EG1 Velocity to Attack	0 secs	0 secs	20 secs	32 bit time cents
EG1 Key to Decay	0 secs	0 secs	20 secs	32 bit time cents
EG2 Section				
EG2 Attack Time	0 secs	0 secs	20 secs	32 bit time cents
EG2 Decay Time	0 secs	0 secs	40 secs	32 bit time cents
EG2 Sustain Level	100%	0%	100%	0.1 % units
EG2 Release Time	0 secs	0 secs	20 secs	32 bit time cents
EG2 Velocity to Attack	0 secs	0 secs	20 secs	32 bit time cents
EG2 Key to Decay	0 secs	0 secs	20 secs	32 bit time cents
EG2 to Pitch	0 cents	-1200 cents	1200 cents	32 bit pitch cents
Miscellaneous Section				
Initial Pan	0%	-50%	50%	0.1% units

Table 2 - DLS Level 1 Default, Minimum, Maximum and Unit Values for Connection Blocks

Any unspecified connection block in a connection block list is set to its default value by the DLS Level 1 synthesis engine.

Data Objects

This section covers in greater depth the data and objects contained within an instrument.

Because extensibility is important to the growth of the Downloadable Sounds specification, it is important to ensure that samples designed for the Level 1 specification will work in future Level specifications as well. The Level 2 specification must be an addition to the Level 1 specification that extends it without contradicting it or rewriting it.

To ensure such future compatibility, the specification follows these guidelines:

- Use 32-bit numbers for data formats, unless genuinely pointless.
- Define data structures in a chunked, expandable file format. Additional controls will be appended to existing objects as needed in future specifications.
- Allow for new structures to be added to the format.
- Design with backwards and forwards compatibility always in mind.

Please refer to the DLS Level 1 File Format for more detail.

Object Hierarchy

A view of the hierarchy of objects in a Level 1 instrument helps demonstrate the relationships between objects.

- **Melodic Instrument**
  - **Global**
    - **Connection Blocks**
  - **Region (1..16)**
    - **Wave Link**
    - **Wave Sample**
    - **Wave File**
      - **Wave Sample (optional)**
      - **Wave Data**

This shows how a melodic instrument is composed of one global articulation set and up to sixteen regions managing up to sixteen samples.

The Drum Kit is a special case because it can have up to 128 matched pairs of drum sample/regions and articulations.

- **Drum Kit**
  - **Region (1..128)**
    - **Wave Link**
    - **Wave Sample**
    - **Wave File**
      - **Wave Sample (optional)**
      - **Wave Data**
    - **Region Articulation**
      - **Connection Blocks**

All structures can be completely contained within the parent structure, with the exception of the Wave Data structure which should not be embedded within a Region. Instead, it should be referred to by the Region. In this way, one sample can be used in multiple places. In fact, a sample should be accessible from more than one Instrument.

Please refer to the DLS Level 1 File Format for more detail.

Data Formats

A Downloadable Sound can contain information in one or more of the following formats.

Cent 1/100th of a semitone.

Time Cents Time Cents, a signed 32-bit integer, represents time in a fractional logarithmic form. Since it is impossible to represent zero in this format, the value 8000000h has been chosen to denote an absolute zero. When the value 80000000h is applied to a destination, that destination cannot be modified by other sources; it will always be zero.

To convert from seconds to Time Cents, use the following formula:

$$Time\,Cents = \log_2\left(time_{\sec s}\right) \times 1200 \times 65536$$

To convert from Time Cents to seconds, use the following formula:

$$Time_{\sec s} = 2^{\left(\frac{tc}{1200 \times 65536}\right)}$$

Pitch Pitch, a signed 32-bit integer, represents tuning in a logarithmic scale based on 1/65,536th of a cent.:

Absolute tuning is determined by the following formula:

$$Pitch = \left[1200 \times \log_2\left(\frac{f}{440}\right) + 6900\right] \times 65536$$

where f is the frequency.

Relative tuning is determined by the following formula:

$$Pitch = 1200 \times 65536 \times \log_2\left(\frac{f}{F}\right)$$

where f is the frequency of interest and F is the reference frequency.

Attenuation A signed, 32-bit integer, is a relative measure of attenuation in fractions of a decibel, where each unit is 1/655,360th of a dB. Since it is impossible to represent infinite attenuation in this format, the value 8000000h has been chosen to denote infinite attenuation. When the value 80000000h is applied to a destination, that destination cannot be modified by other sources; it will always be infinite attenuation. Attenuation is calculated by the following formula:

$$attenuation_{dB} = 200 \times 65536 \times \log_{10}\left(\frac{V}{v}\right)$$

where v is the amplitude of interest and V is the reference amplitude.

Sample Frequency	A 32-bit unsigned integer, with the frequency specified in 1/1000th of Hertz.
Integer	A 32-bit value used for counting, indexing into samples, and so on.

Instrument

A Level 1 Instrument can be either a Drum Kit or Melodic Instrument. The instrument header determines the number of regions in an instrument as well as its bank and program numbers. If the instrument is Melodic, it can have up to 16 regions and one set of Global Articulation data. If it is Drum Kit, it can have up to 128 regions and sets of articulation data, one for each drum.

cRegions	Specifies the count of regions for this instrument.
Locale	Specifies the MIDI locale(Bank and Program Change) for this instrument.

Region

The Region is used by the Control Logic to decide which sample to use. Each region provides velocity and key number ranges to match against an incoming note. If these match, the sample is chosen for performance with the articulation and loop data selected by the region. The Region chunk also contains the Wave Link chunk and may contain articulation data (for Drum Instruments only in Level 1).

Although the field has been provided, Level 1 does not support velocity cross switching.

In Level 1, one instrument is only one sound; no layering is supported other than by using separate MIDI channels.

RangeKey	Specifies the key range for this region.
RangeVelocity	Specifies the velocity range for this region.
fusOptions	Specifies flag options for the synthesis of this region. The only flag defined at this time is the Self Non Exclusive flag. See Note Exclusivity section for more detail.
usKeyGroup	Specifies the key group for a drum instrument. Key group values allow multiple regions within a drum instrument to belong to the same “key group.” If a synthesis engine is instructed to play a note with a key group setting and any other notes are currently playing with this same key group, the synthesis engine should turn off all notes with the same key group value as soon as possible. Valid values are:

0	No Key group
1-15	Key groups 1 to 15.
All Others	Reserved

Wave Link

The Wave Link contains the link to the sample data. This was isolated from the region to allow for synthesis methods other than wavetable synthesis. The sample data is indexed through the Pool Table and the other fields provide the information to phase-lock multiple samples together to create stereo or even surround-sound images.

ulChannel	Specifies the channel placement of the file. This is used to place mono sounds within a stereo pair or for multi-track placement. Each bit position within the ulChannel field specifies a channel placement with bit 0 specifying a mono file or the left channel of a stereo file. Bit 1 specifies the right channel of a stereo file.
ulTableIndex	Specifies the 0 based index of the cue entry in the wave pool table. See the DLS Level 1 File Format document for information on the Pool Table.

Articulation

The Articulation data specifies a number of flexible connections for routing signals in the synthesizer. The Level 1 Device Architecture places a restriction on which connections are valid for a Level 1 Device. The chunk specifies the number of connections, followed by a list of connections.

cConnectionBlocks	Specifies the number (count) of ConnectionBlocks that are contained in the articulator chunk.
usSource	Specifies the source for the connection.
usControl	Specifies the control for the connection.
usDestination	Specifies the destination for the connection
usTransform	Specifies the transform used for the connection.
lScale	Specifies the scaling value used for the connection.

Wave Sample

The Wave Sample chunk contains the low level parameters for playing a sample. Normally it is contained within the Wave chunk itself, but may also be used at the Region level to override loop points or other parameters.

usUnityNote	Specifies the MIDI note which will replay the sample at original pitch. This value ranges from 0 to 127 (a value of 60 represents Middle C, as defined by the MMA).
sFineTune	Specifies the tuning offset from the usUnityNote in 16 bit relative pitch.
lAttenuation	Specifies the attenuation to be applied to this sample in 32 bit relative gain. This is used primarily to balance multi-sample splits.
fulOptions	Specifies flag options for the digital audio sample. Flags are defined for disabling 16 bit to 8 bit truncation of samples and compression of samples by the driver.
cSampleLoops	Specifies the number (count) of loop records that are contained in the wave sample chunk. One shot sounds will have the cSampleLoops field set to 0.
ulLoopType	Specifies the loop type. Only a forward loop is defined.
ulLoopStart	Specifies the start point of the loop in samples as an absolute offset from the beginning of the data in the 'data' chunk of the wave file.
ulLoopLength	Specifies the length of the loop in samples.

Minimum requirements

Except where noted, all specifications and features in this document are requirements for DLS Level 1. However, the following items are significant enough to warrant special mention:

- The hardware must have enough memory to support 256K words of sample memory (512 KB organized as 16-bit samples). After allowing for storage overhead in the device, the minimum number of sample words available for download is 240K samples.
- The hardware must be capable of supporting a minimum of 128 instruments, 128 sets of articulation data, 128 regions, and 128 samples at once. These may be used in the melodic instruments, drum kits or both.
- If a device claims support for both Level 1 and General MIDI, it must be able to support both of them simultaneously.
- 22.05 kHz is the minimum output sample rate of the DLS Level 1 synthesis engine.
- The hardware must support a minimum of 24 simultaneous voices.

Implementation Notes

This section contains technical advice for engineers implementing Level 1 devices in hardware or software.

- When a device supports both Level 1 and General MIDI, the device may share MIDI channels between Level 1 and General MIDI. In such a device, it will not be possible to use the General MIDI drum kit and the Level 1 drum kit simultaneously.
- Level 1 Devices do not support the use of the Coarse Tuning RPN on MIDI Channel 10.

Device Driver Design Notes

This section contains information for software engineers creating device drivers for the specification.

- If the device driver uses system memory for storage, it should allocate enough memory to satisfy the minimum Level 1 device requirements while it is being loaded. If it is unable to allocate the minimum amount of memory, it should report an error to the user and remove itself.
- The device driver model should include an inquiry to allow the application to determine if a Level 1 device also supports General MIDI.
- The Level 1 device driver supports only exclusive access to a Level 1 device. When a device is opened for exclusive access by an application, no other application can use that device until the application closes the device. Level 2 device drivers will address the issue of shared access of the device.

Application Design Notes

This section contains information for software engineers creating applications for use with Level 1 devices.

- Applications or sequences intended for use with a Level 1 device are required to use Instrument Lock messages. If the application also uses General MIDI, it is required to use the Instrument Lock messages for General MIDI channels as well.
- Level 1 Devices do not support the use of the Coarse Tuning RPN on MIDI Channel 10.
- Applications modifying the voice allocation mode should reset the DLS device to the default (static MIDI channel priority) upon exiting to insure predictable polyphony.

Instrument Design Notes

This section contains information for sound designers creating instruments (DLS Files) for use with Level 1 devices.

- While all Level 1 devices will support the full +2/-4 octave transposition range, certain Level 1 devices have other limits on the transposition range. This limitation is imposed by available memory bandwidth and is determined by the number of voices playing and the amount of transposition occurring on each voice. The following formula may be applied to ensure that problems are not encountered with such devices: For each active voice, divide the playback frequency by the frequency at which the sample was recorded. The sum of the ratios of all active voices must be less than or equal to 36. For example, if you have 24 voices playing and each sample is being transposed up 7 semitones (~1.5 times the original frequency), the sum is 24 * 1.5 = 36. However, if one of those voices was transposing up an octave (2 times the original frequency), the sum would be 23 * 1.5 + 2 = 36.5, which would exceed the limitation of some Level 1 devices. Note that when calculating the transposition, you must consider the transposition that occurs as the sample is transposed from its root key, tuning fields in the sample header and articulation data, the Fine Tuning RPN, and any effects from Pitch Bend.
- Some Level 1 devices support playback rates of only 22.05 kHz. To prevent samples from aliasing on these devices, it is recommended that samples for these devices be restricted to 22.05 kHz or below.
- 0dB is the defined as the maximum output level at the Volume summing node (DST_ATTENUATION). If a sound designer exceeds 0dB, the output level will be clipped to 0dB and output will be unpredictable.

Examples

This table (Table 3) represents a sample set of articulation data and control events. Using the information in this table, we can calculate the actual output values for the frequency and volume levels of the oscillator. For this example, we will calculate the outputs for 150 milliseconds after the receipt of the Note-On event.

Articulator Name	usSource	usControl	usDestination	lScale	usTransform
LFO Section					
LFO Frequency	SRC_NONE	SRC_NONE	DST_LFO_FREQ	FCACAE9Ch (5 Hz)	TRN_NONE
LFO Start Delay	SRC_NONE	SRC_NONE	DST_LFO_DELAY	80000000h (0 Sec)	TRN_NONE
LFO Attenuation Scale	SRC_LFO	SRC_NONE	DST_ATTENUATION	001E0000h (+3 dB)	TRN_NONE
LFO Pitch Scale	SRC_LFO	SRC_NONE	DST_PITCH	00000000h (0 Cents)	TRN_NONE
LFO Modw to Attenuation	SRC_LFO	SRC_CC1	DST_ATTENUATION	000A0000h (+1 dB)	TRN_NONE
LFO Modw to Pitch	SRC_LFO	SRC_CC1	DST_PITCH	00000000h (0 cents)	TRN_NONE
EG1 Section					
EG1 Attack Time	SRC_NONE	SRC_NONE	DST_EG1_ATTACKTIME	F9CDAFB1h (400 mSecs)	TRN_NONE
EG1 Decay Time	SRC_NONE	SRC_NONE	DST_EG1_DECAYTIME	0AE2504Fh (5 sec)	TRN_NONE
EG1 Sustain Level	SRC_NONE	SRC_NONE	DST_EG1_SUSTAINLEVEL	01F40000h (50%)	TRN_NONE
EG1 Release Time	SRC_NONE	SRC_NONE	DST_EG1_RELEASETIME	FB500000h (500 mSecs)	TRN_NONE
EG1 Velocity to Attack	SRC_VELOCITY	SRC_NONE	DST_EG1_ATTACKTIME	FB500000h	TRN_NONE
EG1 Key to Decay	SRC_KEYNUM	SRC_NONE	DST_EG1_DECAYTIME	FDA80000h	TRN_NONE
EG2 Section					
EG2 Attack Time	SRC_NONE	SRC_NONE	DST_EG2_ATTACKTIME	80000000h (0 Sec)	TRN_NONE
EG2 Decay Time	SRC_NONE	SRC_NONE	DST_EG2_DECAYTIME	0F92504Fh (10 Sec)	TRN_NONE
EG2 Sustain Level	SRC_NONE	SRC_NONE	DST_EG2_SUSTAINLEVEL	01F40000h (50%)	TRN_NONE
EG2 Release Time	SRC_NONE	SRC_NONE	DST_EG2_RELEASETIME	04B00000h (1 Sec)	TRN_NONE
EG2 Velocity to Attack	SRC_VELOCITY	SRC_NONE	DST_EG2_ATTACKTIME	00000000h	TRN_NONE
EG2 Key to Decay	SRC_KEYNUM	SRC_NONE	DST_EG2_DECAYTIME	00000000h	TRN_NONE
EG2 To Pitch	SRC_EG2	SRC_NONE	DST_PITCH	001E0000h (30 Cents)	TRN_NONE
Connections Inferred by DLS1 Architecture					
EG1 To Attenuation	SRC_EG1	SRC_NONE	DST_ATTENUATION	96 dB	TRN_NONE
Key On Velocity to Attenuation	SRC_KEYONVELOCITY	SRC_NONE	DST_ATTENUATION	96 dB	TRN_CONCAVE
Pitch Wheel to Pitch	SRC_PITCHWHEEL	SRC_NONE	DST_PITCH	Set by RPN 0	TRN_NONE
Key Number to Pitch	SRC_KEYNUMBER	SRC_NONE	DST_PITCH	12,800 Cents	TRN_NONE
MIDI Controller 7 to Attenuation	SRC_CC7	SRC_NONE	DST_ATTENUATION	96 dB	TRN_CONCAVE
MIDI Controller 10 to Pan	SRC_CC10	SRC_NONE	DST_PAN	96 dB	TRN_NONE
MIDI Controller 11 to Attenuation	SRC_CC11	SRC_NONE	DST_ATTENUATION	96 dB	TRN_CONCAVE
Sample					
usUnityNote				54	
sFineTune				00010000h (1 Cent)	
lAttenuation				00000000h (0 dB)	
Control Events					
CC1				25	
CC7				100	
CC10				75	
CC11				120	
Pitch Wheel				365	
RPN 0				200 Cents	
Key Number				60	
Key On Velocity				110	

Table 3 - Example DLS Parameter Table

LFO Generator

The LFO generator is set for a delay of 0 seconds and the frequency is 5 Hz which means the period is 200 milliseconds. If we assume a standard sine wave, after 150 milliseconds, the LFO will be at phase $3\pi/2$ radians, its most negative excursion, which means its output is -1. The Attenuation routing connection specifies that a full scale LFO output will generate a 3 dB attenuation. This is further modified by the CC1 connection, which specifies that a full scale CC1 excursion will generate an additional 1 dB of gain. Since CC1 is 25, the output level is 25/128 dB. The calculation for the LFO to Attenuation connection will look like this:

$$LFOVolume\ Attenuation_{dB} = -1 \times \left(3\,dB + \frac{25}{128} \times 1dB\right) = -3.195\,dB$$

Envelope Generator 1

The EG1 generator has an attack time of 400 milliseconds. However, this is modified by the Velocity to Attack Time connection. With a velocity of 110, the attack time calculation looks as follows:

$$Velocity\ to\ Attack\ Time = -78643200 \times \frac{110}{128} = -68116157\ Time\ Cents$$

$$Attack\ Time = -103960655 + (-68116157) = -172076812\ Time\ Cents$$

$$Attack\ Time = 2^{\left(\frac{-172076812}{65536 \times 1200}\right)} = 219\,m\sec$$

Since the first segment is linear and always begins at 0% and ends at 100%, we can calculate the attenuation at 150 milliseconds by dividing 219 by 150. The calculation for the EG1 to Attenuation connection will look like this:

$$EG1\ Attenuation_{dB} = 20 \times \log_{10}\left(\frac{219}{150}\right) = 3.287\,dB$$

Envelope Generator 2

The EG2 generator has an attack time of 0, meaning that it instantly goes to 100% of full scale, so at 150 milliseconds, the envelope will be in the decay segment. Full scale is 30 cents, so with a decay time of 10 seconds, the decay rate is 0.003 cents per millisecond, and at 150 milliseconds, the output level will be:

$$EG2\ Pitch\ Modulation = 30cents - 150m\sec \times 0.0003\,cents/m\sec = 29.55\,cents$$

This is well above the sustain level of 50%, or 15 cents, so EG2 will still be in decay phase.

Velocity to Attenuation

Velocity to Attenuation is an implied connection in the Level 1 architecture that uses the Concave Transform with a fixed scalar of 96 dB. The calculations follow:

$$Velocity\ Attenuation_{dB} = 20 \times \log_{10}\left(\frac{127^2}{110^2}\right) = 2.496\,dB$$

Pitch Wheel to Pitch

Pitch Wheel to Pitch is an implied connection in the Level 1 architecture that is controlled by Registered Parameter Number 0. The default setting for RPN 0 is 2 semitones or 200 cents. The calculations are as follows:

$$PitchWheel\ Modulation = 200\ cents \times \frac{365}{8192} = 8.911\,cents$$

Key Number to Pitch

Key Number to Pitch is an implied connection in the Level 1 architecture of 128 semitones or 12,800 cents:

$$Key\ Number\ Modulation = 12800\ cents \times \frac{60}{128} = 6000\,cents$$

MIDI Controller 7 to Attenuation

MIDI Controller 7 to Attenuation is another implied connection in the Level 1 architecture that uses the Concave Transform with a fixed scalar of 96 dB. The calculations follow:

$$CC7\ Attenuation_{dB} = 20 \times \log_{10}\left(\frac{127^2}{100^2}\right) = 4.152\,dB$$

MIDI Controller 10 to Pan

MIDI Controller 10 to Pan is an implied connection in the Level 1 architecture. The calculations for each channel are as follows:

$$Left\ Channel\ Attenuation_{dB} = 20 \times \log_{10}\sqrt{\frac{127}{127-75}} = 3.878\,dB$$

$$Right\ Channel\ Attenuation_{dB} = 20 \times \log_{10}\sqrt{\frac{127}{75}} = 2.287\,dB$$

The final output for the left and right channels are determined by adding the calculated attenuation to the accumulated attenuation for all other amplitude modulation sources.

MIDI Controller 11 to Attenuation

MIDI Controller 11 to Attenuation is another implied connection in the Level 1 architecture that uses the Concave Transform with a fixed scalar of 96 dB. The calculations follow:

$$CC11\ Attenuation_{dB} = 20 \times \log_{10}\left(\frac{127^2}{120^2}\right) = 0.985\,dB$$

Accumulated Pitch Calculation

The final outputs for pitch and volume are an accumulation of all the other calculations for volume and pitch. The output frequency is determined by summing all of the pitch modulation sources, i.e. Key Number to Pitch, Pitch Wheel to Pitch, LFO and EG2, dwFineTune and then subtracting out the unity note after converting it to pitch units. The calculation for the example is as follows:

$$Pitch = 29.55 + 8.911 + 6000.0 + 1.0 - 5400.0 = 639.461\,cents$$

$$Playback\ ratio = 2^{\left(\frac{639.461}{1200}\right)} = 1.447$$

The actual playback rate will be determined by the ratio of the recorded sample rate and the sample playback rate multiplied by the above ratio. For example, if the original material was recorded at 22050 kHz and the playback rate is 32000 kHz, then the final playback ratio will be:

$$Playback\ ratio = \frac{22050}{32000} \times 1.447 = 0.997$$

This means that the sample will playback at nearly unity, but because the target device has a higher playback rate, the pitch will be raised by over 6 semitones.

Accumulated Attenuation Calculation

The output volume is determined by summing all of the attenuation sources, i.e. EG1, LFO Velocity, CC7, CC11, and dwAttenuation, and then calculating left and right outputs individually based on the CC10 calculations. The example can be calculated as follows:

$$Mono\ Attenuation_{dB} = -3.195 + 3.287 + 2.496 + 4.152 + 0.985 + 0.0 = 7.725\,dB$$

$$Left\ Channel\ Attenuation_{dB} = 7.725 + 3.878 = 11.603\,dB$$

$$Right\ Channel\ Attenuation_{dB} = 7.725 + 2.287 = 10.012\,dB$$

DLS-1 System Messages

The following Universal Non-Real Time SysEx messages are explained in more detail in Chapter 1, pages 8-9.

Turn DLS Level 1 On:

F0 7E < device ID > 0A 01 F7

F0 7E	Universal Non-Real Time SysEx header
< device ID >	ID of target device (suggest using 7F: Broadcast)
0A	sub-ID #1 = DLS Level 1 message
01	sub-ID #2 = DLS Level 1 On
F7	EOX

Turn DLS Level 1 Off:

F0 7E < device ID > 0A 02 F7

F0 7E	Universal Non-Real Time SysEx header
< device ID >	ID of target device (suggest using 7F: Broadcast)
0A	sub-ID #1 = DLS Level 1 message
02	sub-ID #2 = DLS Level 1 Off
F7	EOX

Turn DLS Level 1 Voice Allocation Off:

F0 7E < device ID > 0A 03 F7

F0 7E	Universal Non-Real Time SysEx header
< device ID >	ID of target device (suggest using 7F: Broadcast)
0A	sub-ID #1 = DLS Level 1 message
03	sub-ID #2 = DLS Level 1 Voice Allocation Off
F7	EOX

Turn DLS Level 1 Voice Allocation On:

F0 7E < device ID > 0A 04 F7

F0 7E	Universal Non-Real Time SysEx header
< device ID >	ID of target device (suggest using 7F: Broadcast)
0A	sub-ID #1 = DLS Level 1 message
04	sub-ID #2 = DLS Level 1 Voice Allocation On
F7	EOX

Chapter 2: File Format

Purpose

This chapter defines a file format for downloadable instruments and sound effects. It uses a structure that can encapsulate both digital audio and sampler/sound card parameters in a non-proprietary and expandable form. This format provides for the needs of both musicians and multimedia developers. It allows musicians to create and save multi-sample instruments which can be easily interchanged between samplers. Multimedia developers can store all of the sounds and basic control information needed to exactly reproduce a MIDI soundtrack or sound effects on any sampler or RAM based soundcard.

The DLS (Downloadable Sounds) file format is used to store both the digital sound data and articulation parameters needed to create one or more “instruments.” An instrument contains “regions” which point to WAVE “files” also embedded in the DLS file. Each region specifies a MIDI note and velocity range which will trigger the corresponding sound and also contains articulation information such as envelopes and loop points. Articulation information can be specified for each individual region or for the entire instrument.

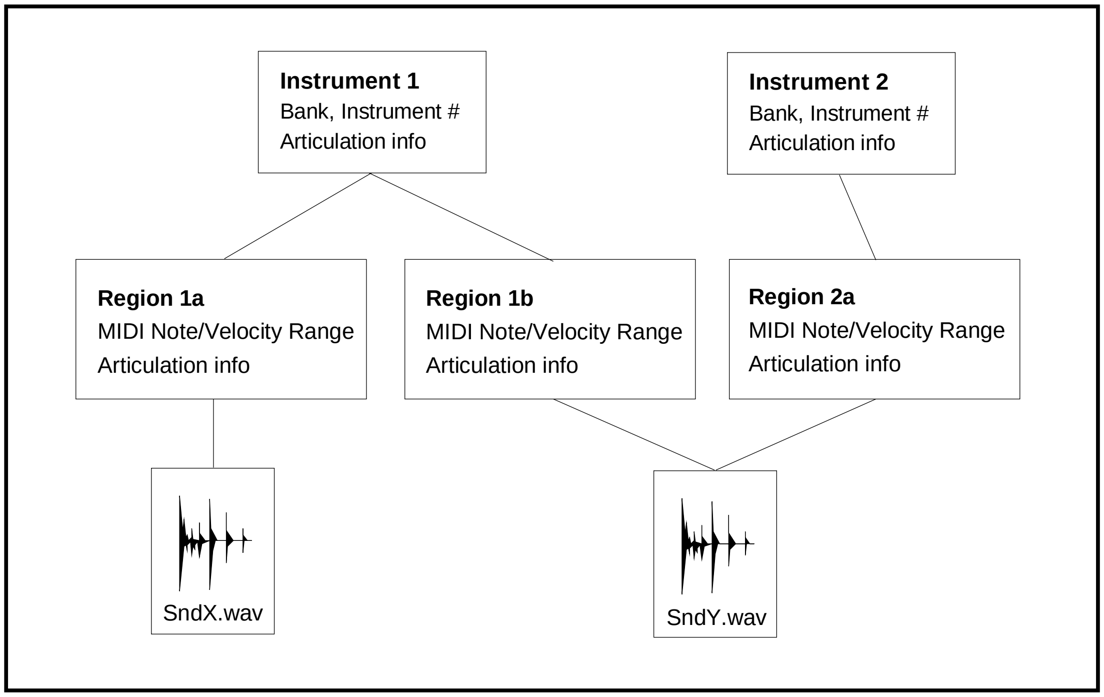


Figure 17. **DLS File Structure**

Advantages of the DLS Format

Adheres to standard RIFF layout

- Existing RIFF handling code including support libraries can be used to read and write DLS files.
- Use of the LIST type throughout the file allows existing RIFF parsing tools like RIFFWALK to easily navigate, verify, and display all sub chunks.

Simple structure

- The structure is very similar for all levels of main chunk types (collection, instrument, region).
- The format is straight forward and conceptually easy to understand.
- Allows for re-use of code when reading and writing chunks at each level.

Uses standard WAVE files

- The format is well defined and is expandable.
- Future issues such as compression have already been dealt with in wave files.
- Existing wave file code will need little modification to read digital audio data within DLS files. See Appendix D — DLS Level 1 Wave File Format for examples.

Easy and unlimited expansion

- The format is written so additional chunks can be embedded at any level (collection, instrument, region), allowing a file to meet DLS Level 1 specifications yet contain additional information for proprietary synthesis features. See “Proprietary Chunk IDs” for details.

Downloadable Sound Collection RIFF Files

Downloadable sound collections are stored in a RIFF file with a form type of 'DLS '. The subchunks of this form of RIFF file are the ‘vers’, ‘dlid’, 'colh', 'ptbl' and 'LIST' chunks. There are three top-level LIST chunks: the <INFO-list> chunk contains textual informative details about the instrument collection, the <lins-list> instrument list chunk contains <ins-list> instrument subchunks, and the <wvpl-list> wave pool chunk contains <wave-list> subchunks (similar to wave files). The optional top-level <dlid-ck> chunk specifies a globally unique identifier for the entire collection. The <colh-ck> chunk defines the number of instruments in the collection. The optional <vers-ck> chunk specifies the version of the file.

The <ptbl-ck> chunk contains a list of reference entries to digital audio data. The <ins-list> subchunks within the <lins-list> chunk are the actual instruments stored in this collection. Additional <dlid-ck> chunks may be used to specify globally unique identifiers for each <ins-list> instrument and <wave-list> wave file chunk.

The following is a RIFF grammar (as defined in the Microsoft Windows Multimedia Programmer's Reference) that describes the downloadable instrument collection:

```
<DLS-form>     →     RIFF( 'DLS '                     // Collection
                          [<vers-ck>]
                          [<dlid-ck>]
                          <colh-ck>
                          <lins-list>
                          <ptbl-ck>
                          <wvpl-list>
                          [<INFO-list>] )

<wvpl-list>    →    LIST( 'wvpl' <wave-list>...)      // Wave Pool

<wave-list>    →    LIST( 'wave'                      // Wave File
                          [<dlid-ck>]
                          <fmt-ck>
                          <data-ck>
                          [<wsmp-ck>]
                          [<INFO-list>] )

<lins-list>    →    LIST( 'lins' <ins-list>...   )    // List of Instruments

<ins-list>     →    LIST( 'ins '                      // Instrument
                          [<dlid-ck>]
                          <insh-ck>
                          <lrgn-list>
                          [<lart-list>]
                          [<INFO-list>]   )

<lrgn-list>    →    LIST( 'lrgn' <rgn-list>...   )    // List of Regions

<rgn-list>     →    LIST( 'rgn '                      // Region
                          <rgnh-ck>
                          [<wsmp-ck>]
                          <wlnk-ck>
                          [<lart-list>]   )

<lart-list>    →    LIST( 'lart' <art1-ck>… )         // List of Articulators

<INFO-list>    →    LIST( 'INFO' <info_text-ck>...)
```

Chunk	Definition
<colh-ck>	A DLS collection header as defined later in this document.
<dlid-ck>	A globally unique identifier chunk as defined later in this document.
<insh-ck>	An instrument header chunk as defined later in this document.
<rgnh-ck>	A region header chunk as defined later in this document.
<art1-ck>	A level 1 articulator data chunk as defined later in this document.
<wlnk-ck>	A wave link chunk as defined later in this document.
<wsmp-ck>	A wave sample chunk as defined later in this document.
<ptbl-ck>	A pool table data chunk as defined later in this document
<vers-ck>	An optional version chunk as defined later in this document
<wave-list>	A DLS wave file chunk as defined later in this document, Appendix D: DLS Level 1 Wave File Format
<info_text-ck>	A text chunk within an <INFO-list> chunk as defined later in this document.

Table 4 - RIFF Definitions

LIST Chunk

A LIST chunk contains a list, or ordered sequence, of subchunks. A LIST chunk is defined as follows:

LIST(<list-type> [<chunk>]...)

The <list-type> is a four-character code that identifies the contents of the list.

If an application recognizes the list type, it should know how to interpret the sequence of subchunks. However, since a LIST chunk may contain only subchunks (after the list type), an application that does not know about a specific list type can still walk through the sequence of subchunks.

Like chunk IDs, list types must be registered, and an all-lowercase list type has meaning relative to the form that contains it.

<colh-ck>, Collection Header Chunk

Defined for: DLS form

The <colh-ck> collection header defines an instrument collection. The <colh-ck> is defined as follows:

```
<colh-ck>        →        colh( <cInstruments:ULONG> )
```

The <colh-ck> chunk:

Field	Description
cInstruments	Specifies the count of instruments in this collection.

A <colh-ck> chunk is typically (but not necessarily) the ***second*** chunk within the enclosing <DLS-form> collection chunk. Other chunks at the same nesting level include an optional <dlid-ck> collection DLSID chunk, a <lins-list> list of instruments chunk, a <ptbl-ck> pool table chunk, a <wvpl-list> wave pool chunk, and an optional <INFO-list> collection information chunk.

<dlid-ck>, DLSID Chunk

Defined for: DLS form

The <dlsd-ck> defines an optional globally unique identifier (DLSID) for a complete <DLS-form> or for an element within it. A DLSID is identical to an OSF/DCE 'UUID' (also known as a Microsoft® 'GUID', 'CLSID', 'IID' or 'FMTID'). The <dlid-ck> is defined as follows:

```
<dlid-ck>        →        dlid( <DLSID>    )

<DLSID>          →        struct
                          {
                          ULONG                   ulData1;
                          USHORT        usData2;
                          USHORT        usData3;
                          BYTE          abData4[8];
                          }
```

The <dlid-ck> chunk:

Field	Description
DLSID	Specifies a 128-bit (16 byte) integer used as a globally unique identifier for a content element. The DCE standard specifies that the string representation for a UUID contains five fixed-length, hyphen-delimited groups of hexadecimal digits, grouped 8-4-4-4-12, as shown: 60DF3430-0266-11cf-BAA6-00AA003E0EED

The <DLSID> structure:

Field	Description
ulData1	Contains the binary value of the first group of the string value (8 hexadecimal digits).
usData2	Contains the binary value of the second group of the string value (4 hexadecimal digits).
usData3	Contains the binary value of the third group of the string value (4 hexadecimal digits).
abData4[8]	Contains the fourth and fifth groups of the string value. Array elements 0 and 1 of hold the fourth group (4 digits), while elements 2 through 7 hold the final group (12 digits). The C declaration for the example DLSID would be: const DLSID example = { 0x60df3430, 0x0266, 0x11cf, { 0xba, 0xa6, 0x00, 0xaa, 0x00, 0x3e, 0x0e, 0xed } }; The actual hexadecimal byte sequence (in ascending order) would be: 30 34 DF 60 66 22 CF 11 BA A6 00 AA 00 3E 0E ED Byte order in the first three fields appears reversed because Intel® (little-endian) binary representation is used.

The DLSID value must be generated *by computer and never manually*, in order to obtain a unique value (software for this purpose is commonly available). The DLSID must be generated using the UUID algorithm specified by OSF DCE[1], which uses a combination of the following information to generate the value:

- The current date and time
- A *clock sequence* and related persistent state to deal with retrograde motion of clocks
- A forcibly incremented counter to deal with high-frequency allocations
- A globally unique IEEE machine identifier, obtained from a network card (if no network card is present, a machine identifier may be synthesized from highly variable machine states and stored persistently).

A DLSID uniquely designates a particular DLS resource (collection, instrument or wave file). If the resource is modified in any way *whatsoever*, a new DLSID must be generated and attached to that resource. Therefore, changing an instrument or wave file also requires changing the DLSID for the enclosing collection.

DLSIDs provide four principal benefits:

- They allow DLS Devices to detect multiply-referenced resources and share a single copy of the data.
- They allow an application to determine whether a DLS resource located on a remote network site is already present on the local system, before downloading the resource.
- They facilitate searching for a particular resource and building special-purpose collections.
- They support error recovery. A DLS manager applet can identify and recover "orphan" resources left in a locked state by a faulty application.

DLSID chunks are not required in DLS files.

[1] Chapter 10, "DEC/HP Network Computing Architecture Remote Procedure Call RunTime Extensions Specification Version OSF TX1.0.11", Steven Miller, July 23 1992.

<insh-ck>, Instrument Header Chunk

Defined for: DLS form

The <insh-ck> defines an instrument within a collection. The <insh-ck> is defined as follows:

```
<insh-ck>     →     insh(
                          <cRegions:ULONG>
                          <Locale:MIDILOCALE>
                    )
```

The <insh-ck> chunk:

Field	**Description**
cRegions	Specifies the count of regions for this instrument.
Locale	Specifies the MIDI locale for this instrument.

Additional Structure Definitions:

```
typedef struct    _MIDILOCALE
                  {
                     ULONG ulBank;
                     ULONG ulInstrument;
                  }
            MIDILOCALE, *MIDILOCALE;
```

Field	**Description**
ulBank	Specifies the MIDI bank location. Bits 0-6 are defined as MIDI CC32 and bits 8-14 are defined as MIDI CC0. Bits 7 and 15-30 are reserved and should be written to zero. If Bit 31 is equal to 1 then the instrument is a drum instrument; if equal to 0 then the instrument is a melodic instrument.
F_INSTRUMENT_DRUMS	Specifies that this instrument is a drum instrument.
ulInstrument	Specifies the MIDI Program Change (PC) value. Bits 0-6 are defined as PC value and bits 7-31 are reserved and should be written to zero.

An <insh-ck> instrument header is typically (but not necessarily) the ***second*** chunk within the enclosing <ins-list> instrument chunk. Other chunks at the same nesting level include an optional <dlid-ck> instrument DLSID chunk, a <lrgn-list> list of regions chunk, an optional <lart-list> list of articulators chunk (for melodic instruments only), and an optional <INFO-list> instrument information chunk.

<rgnh-ck>, Region Header Chunk

Defined for: DLS form

The <rgnh-ck> defines a region within an instrument. The <rgnh-ck> is defined as follows:

```
<rgnh-ck>      →      rgnh(
                           <RangeKey:RGNRANGE>
                           <RangeVelocity:RGNRANGE>
                           <fusOptions:USHORT>
                           <usKeyGroup:USHORT>
                           )
```

The <rgnh-ck> chunk:

Field	Description
RangeKey	Specifies the key range for this region.
RangeVelocity	Specifies the velocity range for this region.
fusOptions	Specifies flag options for the synthesis of this region. Current options are: **F_RGN_OPTION_SELFNONEXCLUSIVE** This option specifies that if a second note on of the same note is received by the synthesis engine then the second note will be played as well as the first. This option is off by default and the synthesis engine will force a note off of the prior note if a second note is received of the same value.
UsKeyGroup	Specifies the key group for a drum instrument. Key group values allow multiple regions within a drum instrument to belong to the same "key group." If a synthesis engine is instructed to play a note with a key group setting and any other notes are currently playing with this same key group, then the synthesis engine should turn off all notes with the same key group value as soon as possible. Valid values are: 0 — No Key group 1-15 — Key groups 1 to 15. All Others — Reserved

Additional Structure Definitions:

```
typedef struct     _RGNRANGE
                       {
                         USHORT usLow;        /* Low Value of Range */
                         USHORT usHigh;       /* High Value of Range*/
                       }
                   RGNRANGE, *RGNRANGE;
```

A <rgnh-ck> region header is typically (but not necessarily) the ***first*** chunk within the enclosing <rgn-list> region chunk. Other chunks at the same nesting level include a <wsmp-ck> wave sample chunk, a <wlnk-ck> wave link chunk, and an optional <lart-list> list of articulators chunk (for drum instruments only).

<art1-ck>, Level 1 Articulator Chunk

Defined for: DLS form

The <art1-ck> articulator chunk specifies parameters which modify the playback of a wave file used in downloadable instruments. An <art1-ck> chunk may be used in either melodic instruments (for global articulation) or in drum instruments (for region articulation). The <art1-ck> is defined as follows:

```
        < art1-ck>      →     art1(
                              <cbSize:ULONG>
                              <cConnectionBlocks:ULONG>
                              <ConnectionBlock(s)>...
                        )

<ConnectionBlock>       →     struct
                        {
                          USHORT      usSource;
                          USHORT      usControl;
                          USHORT      usDestination;
                          USHORT      usTransform;
                          LONG        lScale;
                        }
```

The <art1-ck> chunk:

Field	**Description**
cbSize	Specifies the size of the structure in bytes. *This size does not include the connection blocks.* This field is needed to distinguish the amount of data in the structure versus the list of connections and allow for additions to this structure in the future. This cannot be determined from the chunk size.
cConnectionBlocks	Specifies the number (count) of <ConnectionBlock> records that are contained in the <art1-ck> articulator chunk. The <ConnectionBlock> records are stored immediately following the **cConnectionBlocks** data field.

The <ConnectionBlock> structure:

Field	**Description**
usSource	Specifies the source for the connection.
usControl	Specifies the control for the connection.
usDestination	Specifies the destination for the connection
usTransform	Specifies the transform used for the connection.
lScale	Specifies the scaling value used for the connection.

The list of connection blocks defines both the architecture and settings for an instrument or instrument region. For the DLS Level 1 synthesizer architecture, there are 29 connection blocks in the connection graph, 10 of which are implicit and have no attached values, and 19 that define the articulation for a DLS Level 1 sound. Although this could be defined as a simple structure of values, the connection graph model allows for future chunk types which will have a much greater possible number of connections, making the use of a structure unwieldy.

By using the connection graph, a single model can be used for future architectures. See the Downloadable Sound Device Architecture document for detail on each connection block. Below is a list of the defined sources, controls, destinations, and transforms of a DLS Level 1 connection block:

List of defined Sources, Controls, Destinations, and Transforms

Generic Sources

CONN_SRC_NONE	No Source
CONN_SRC_LFO	Low Frequency Oscillator
CONN_SRC_KEYONVELOCITY	Key on Velocity
CONN_SRC_KEYNUMBER	Key Number
CONN_SRC_EG1	Envelope Generator 1
CONN_SRC_EG2	Envelope Generator 2
CONN_SRC_PITCHWHEEL	Pitch Wheel

MIDI Sources

CONN_SRC_CC1	Modulation Wheel
CONN_SRC_CC7	Channel Volume
CONN_SRC_CC10	Pan
CONN_SRC_CC11	Expression
CONN_SRC_RPN0	RPN0 - Pitch Bend Range
CONN_SRC_RPN1	RPN1 - Fine Tune
CONN_SRC_RPN2	RPN1 - Coarse Tune

Generic Destinations

CONN_DST_NONE	No Destination
CONN_DST_ATTENUATION	Attenuation
CONN_DST_PAN	Pan
CONN_DST_PITCH	Pitch

LFO Destinations

CONN_DST_LFO_FREQUENCY	LFO Frequency
CONN_DST_LFO_STARTDELAY	LFO Start Delay Time

EG1 Destinations

CONN_DST_EG1_ATTACKTIME	EG1 Attack Time
CONN_DST_EG1_DECAYTIME	EG1 Decay Time
CONN_DST_EG1_SUSTAINLEVEL	EG1 Sustain Level
CONN_DST_EG1_RELEASETIME	EG1 Release Time

EG2 Destinations

CONN_DST_EG2_ATTACKTIME	EG2 Attack Time
CONN_DST_EG2_DECAYTIME	EG2 Decay Time
CONN_DST_EG2_SUSTAINLEVEL	EG2 Sustain Level
CONN_DST_EG2_RELEASETIME	EG2 Release Time

Transforms

CONN_TRN_NONE	No Transform
CONN_TRN_CONCAVE	Concave Transform

<wlnk-ck>, Wave Link Chunk

Defined for: DLS form

The <wlnk-ck> specifies where the wave data can be found for an instrument region in a DLS file. The <wlnk-ck> is defined as follows:

```
<wlnk-ck>    →    wlnk(
                           <fusOptions:USHORT>
                           <usPhaseGroup:USHORT>
                           <ulChannel:ULONG>
                           <ulTableIndex:ULONG>
                      )
```

The <wlnk-ck> chunk:

Field	Description	
fusOptions	Specifies flag options for this wave link.	
	F_WAVELINK_PHASE_MASTER	Specifies that this link is the master in a group of phase locked wave links.
usPhaseGroup	Specifies a group number for samples which are phase locked. All waves in a set of wave links with the same group are phase locked and follow the wave in the group with the F_WAVELINK_PHASE_MASTER flag set. If a wave is not a member of a phase locked group, this value should be set to 0.	
ulChannel	Specifies the channel placement of the file. This is used to place mono sounds within a stereo pair or for multi-track placement. Each bit position within the ulChannel field specifies a channel placement with bit 0 specifying a mono file or the left channel of a stereo file. Bit 1 specifies the right channel of a stereo file.	
ulTableIndex	Specifies the 0 based index of the cue entry in the wave pool table.	

<wsmp-ck>, Wave Sample Chunk

Defined for : DLS form

The <wsmp-ck> wave sample chunk describes the minimum necessary information needed to allow a synthesis engine to use a <wave-list> wave file chunk. The <wsmp-ck> is defined as follows:

```
<wsmp-ck>        →        wsmp(
                                  <cbSize:ULONG>
                                  <usUnityNote:USHORT>
                                  <sFineTune:SHORT>
                                  <lAttenuation:LONG>
                                  <fulOptions:ULONG>
                                  <cSampleLoops:ULONG>
                                  <wavesample-loop>...
                                   )

<wavesample-loop>       →         struct
                                  {
                                  ULONG          cbSize;
                                  ULONG          ulLoopType;
                                  ULONG          ulLoopStart;
                                  ULONG          ulLoopLength;
                                  }
```

The <wsmp-ck> chunk:

Field	Description	
cbSize	Specifies the size of the structure in bytes. *This size does not include the loop records.* This field is needed to distinguish the amount of data in the structure versus the list of loops and allow for additions to this structure in the future. This cannot be determined from the chunk size.	
usUnityNote	Specifies the MIDI note which will replay the sample at original pitch. This value ranges from 0 to 127 (a value of 60 represents Middle C, as defined in MIDI 1.0).	
sFineTune	Specifies the tuning offset from the usUnityNote in 16 bit relative pitch.	
lAttenuation	Specifies the attenuation to be applied to this sample in 32 bit relative gain.	
fulOptions	Specifies flag options for the digital audio sample. Current options are:	
	F_WSMP_NO_TRUNCATION	This option specifies that a synthesis engine is not allowed to truncate the bit depth of this sample if it can't synthesize at the bit depth of the digital audio. For instance, if the data was 16 bit and the engine synthesized using 8 bit data, the engine is not allowed to truncate the audio to 8 bits in order to synthesize.
	F_WSMP_NO_COMPRESSION	This option specifies that a synthesis engine is not allowed to use compression in its internal synthesis engine for the digital audio sample. For instance, if the synthesis engine used Ulaw compression prior to storing data for synthesis, the engine is not allowed to compress this sample prior to synthesis.

cSampleLoops	Specifies the number (count) of <wavesample-loop> records that are contained in the <wsmp-ck> chunk. The <wavesample-loop> records are stored immediately following the **cSampleLoops** data field. One shot sounds will have the **cSampleLoops** field set to 0. Looped sounds will have the **cSampleLoops** field set to 1. Values greater than 1 are not yet defined.

The <wavesample-loop> structure:

Field	**Description**
cbSize	Specifies the size of the structure in bytes.
ulLoopType	Specifies the loop type: **WLOOP_TYPE_FORWARD** Forward Loop
ulLoopStart	Specifies the start point of the loop in samples as an absolute offset from the beginning of the data in the <data-ck> subchunk of the <wave-list> wave file chunk.
ulLoopLength	Specifies the length of the loop in samples.

<ptbl-ck>, Pool Table Chunk

Defined for: DLS form

The <ptbl-ck> pool table chunk contains a list of cross-reference entries to digital audio data within the wave pool. The <ptbl-ck> is defined as follows:

```
<ptbl-ck>     →     ptbl(
                        <cbSize:ULONG>
                        <cCues:ULONG>
                        <poolcues(s)>...
                        )

<poolcue>     →     struct
                        {
                          ULONG       ulOffset;
                        }
```

The <ptbl-ck> chunk:

Field	**Description**
cbSize	Specifies the size of the structure in bytes. *This size does not include the poolcue records.* This field is needed to distinguish the amount of data in the structure versus the list of cues and allow for additions to this structure in the future. This cannot be determined from the chunk size.
cCues	Specifies the number (count) of <poolcue> records that are contained in the <ptbl-ck> chunk. The <poolcue> records are stored immediately following the **cCues** data field.

The <poolcue> structure:

Field	Description
ulOffset	Specifies the absolute offset in bytes from the beginning of the wave pool data to the correct entry in the wave pool. By using an offset to the data of each pool entry, a synthesis engine does not have to walk from the beginning of the list to find any given set of wave data but can immediately seek to the correct location in the file. This is especially important for synthesis engines which make use of memory mapped files.

Additions and deletions of entries in the wave pool need only modify the pool table as opposed to modifying every <wave-link> in the instrument chunks.

<vers-ck>, Version Chunk

Defined for: DLS form

The <vers-ck> defines an optional version stamp within a collection. The version stamp is intended primarily as a mechanism for managing DLS resources, in particular when installing DLS files. A setup program reads the version chunk in a previously installed DLS file to identify whether it should install a newer file over it. The version stamp follows the same four number format as the version stamps used in executable code (.exe and .dll files). It is purely up to the discretion of the DLS file author to assign a version number scheme to a file.

The <vers-ck> is defined as follows:

```
<vers -ck>      →      vers(
                               <dwVersionMS:DWORD>
                               <dwVersionLS:DWORD >
                        )
```

The <vers -ck> chunk:

Field	Description
dwVersionMS	Specifies the high-order 32 bits of the binary version number for the file. The value of this member is used with the value of the dwVersionLS member to form a 64-bit version number. This 32 bit value can be further broken down with HIWORD(dwVersionMS) and LOWORD(dwVersionMS) to get two 16 bit values (version info is actually 4 16 bit values.)
dwVersionLS	Specifies the low-order 32 bits of the binary version number for the file. The value of this member is used with the dwVersionMS value to form a 64-bit version number. Use HIWORD(dwVersionLS) and LOWORD(dwVersionLS) to extract the sixteen bit values.

The version chunk is essentially borrowed from the dwFileVersionMS and dwFileVersionLS fields in the Microsoft® VS_FIXEDFILEINFO structure which is used to store version information in .exe and .dll files. As such, the version can be broken down into four segments: the high and low words of the MS and the high and low words of the LS. For example, “VERSION 3,10,0,61” is translated into: dwVersionMS = 0x0003000a and dwVersionLS = 0x0000003d.

A <vers-ck> version chunk is typically (but not necessarily) near the top of the enclosing DLS chunk so it can be quickly found by installation software.

<INFO-list>, INFO List Chunk

Defined for: DLS form

The INFO list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An INFO list is a LIST chunk with list type INFO. The following shows a sample INFO list chunk:

```
< INFO-list >    →    LIST(  'INFO'  [<info_text-ck>]...)
```

Example:

```
< INFO-list >    →    LIST(  'INFO'
                                  INAM("Distorted Tuba"Z)
                                  ICMT("The Tuba from hell meets Eddie Van Halen"Z)
                           )
```

An INFO list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an INFO list. Each chunk contains a ZSTR, or null-terminated text string.

The <info_text-ck> chunks include:

Chunk ID	Description
IARL	Archival Location. Indicates where the subject of the file is archived.
IART	Artist. Lists the artist of the original subject of the file. For example, Les Paul.
ICMS	Commissioned. Lists the name of the person or organization that commissioned the subject of the file. For example, Pope Julian II.
ICMT	Comments. Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do not include newline characters.
ICOP	Copyright. Records the copyright information for the file. For example, Copyright Encyclopedia International 1991. If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	Creation date. Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example, 1553-05-03 for May 3, 1553.
IENG	Engineer. Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example, Smith, John; Adams, Joe.
IGNR	Genre. Describes the original work, such as, jazz, classical, rock, techno, rave, neo british pop grunge metal, etc.
IKEY	Keywords. Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example, FX; death; murder.
IMED	Medium. Describes the original subject of the file, such as, record, CD, and so forth.

INAM	Name. Stores the title of the subject of the file, such as, Seattle From Above.
IPRD	Product. Specifies the name of the title the file was originally intended for, such as World Ruler V.
ISBJ	Subject. Describes the contents of the file, such as Music of the New World Order.
ISFT	Software. Identifies the name of the software package used to create the file, such as Sonic Foundry Sound Forge.
ISRC	Source. Identifies the name of the person or organization who supplied the original subject of the file. For example, Trey Research.
ISRF	Source Form. Identifies the original form of the material that was digitized, such as record, sampling CD, TV sound track, and so forth. This is not necessarily the same as IMED.
ITCH	Technician. Identifies the technician who sampled the subject file. For example, Smith, John.

Coding Requirements and Recommendations

When writing code to read and write DLS files, it is imperative that the programmer takes into account the following:

Since this is an expandable format, DLS file parsers must be able to ignore and correctly skip unrecognized chunks in an DLS file. This is standard practice for RIFF files but is extremely important with the DLS format since one of its purposes is to allow manufacturers to embed proprietary chunks which exploit advanced features of their synthesis engines.

Ordering of chunks within the file format should not be assumed. For instance, a programmer should not assume that within a <rgn-list> instrument region, chunks will always appear in the order of <rgnh-ck>,<wsmp-ck>,<wlnk-ck>, <lart-list>. They may actually appear in any order within the list.

The following restrictions are placed upon the file format and must be followed for DLS Level 1 compatibility:

For the <wlnk-ck> wave link the only allowed type for the **ulChannel** field is WAVELINK_CHANNEL_LEFT. This means that for a DLS Level 1 file only mono sound data is allowed. **fusOptions, usPhase Group** and other values for **ulChannel** other than **ulChannel** = WAVELINK_CHANNEL_LEFT should be set to default values since these fields are optional and are not required to be supported in DLS Level 1.

For the embedded wave files <wave-list> within the <wvpl-list> the only data format allowed is WAVE_FORMAT_PCM. The files also must be 8 or 16 bit mono files. This means that only mono files which are 8 or 16 bit PCM data are allowed for DLS Level 1. See Appendix D.

For the <rgnh-ck> the allowed key groupings for the **ulKeyGroup** field are 0 for no key grouping and 1-15 for key groupings. This means there are 15 allowable key groupings within an instrument for DLS Level 1.

For the <lrgn-list> a maximum of 16 regions can exist for any given melodic instrument and 128 regions for a drum instrument.

The **RangeVelocity** field is not supported by the DLS Level 1 specification, so all regions' **RangeVelocity** fields for a DLS Level 1 collection should be set to 0 for the **usLow** field and 127 for the **usHigh** field.

DLS Level 1 regions are not allowed to overlap. This means that “splits” are supported by DLS Level 1 but “layers” are not. Therefore, an instrument's regions **RangeKey** field can not overlap.

Articulation data lists <lart-list> may be embedded for each region in a drum instrument but only one <lart-list> is allowed for a melodic instrument. This means that DLS Level 1 drum instruments have articulation data for each region, but melodic instruments have only one set of articulation data for all regions.

To detect whether a given instrument is a drum or melodic instrument, a program should check the instrument header's **ulBank** field bit 31. If ulBank bit 31 is equal to 1, then the instrument is a drum instrument. If **ulBank** bit 31 is equal to 0, then the instrument is a melodic instrument. The distinction between drum and melodic instruments will cease to exist in a future revision of the DLS specification.

For the <art1-ck> the **ulDefaultPan** field is only used when specifying articulation data for regions of Drum instruments. For melodic instruments this value should not be specified in the connection list.

Connections that are inferred by the DLS device architecture must not be written to the DLS file. Inferred connections include: EG1 to Attenuation, Velocity to Attenuation, Pitch Wheel/RPN0 to Pitch, Key Number to Pitch, MIDI Controller to Attenuation, MIDI Controller 10 to Pan, MIDI Controller 11 to Attenuation, RPN 1 to Pitch, RPN2 to Pitch.

If a <wsmp-ck> does not exist in a region list, the synthesis engine will use the <wsmp-ck> found in the referenced wave data. If a <wsmp-ck> exists in a region list, its values are used instead of any found in the referenced wave data. If a <wsmp-ck> exists in neither the region list nor the referenced data, then default values are used. These are One shot Sound, with a MIDI Unity note of 60 (Middle C), no attenuation, and no fine tune.

The following requirements must be observed to maintain the integrity of DLSID Globally Unique Identifiers:

Support for DLSID Identifiers is optional, not required. However, a DLS implementation which does not support the use of DLSID Identifiers or the generation of new DLSID values must still observe several requirements. First, such implementations must be prepared to ignore and skip over any DLSIDs found within a DLS file, just as if the <dlid-ck> DLSID were an unrecognized chunk. Second, if such an implementation ***modifies*** a DLS file, it **must** remove any <dlid-ck> chunk found within the top-level <DLS-form> and any additional <dlid-ck> chunks found within the modified portions of the DLS file. Leaving the unchanged <dlid-ck> chunks within the modified file would violate the DLSID “uniqueness” rule. Finally, when exporting a DLS wave file from a <DLS-form> collection to a raw <WAVE-form> wave file, any embedded <dlid-ck> chunks should be stripped out.

For implementations which **do** support the use of <dlid> DLSID chunks:

If an object has a DLSID associated with it, a new DLSID value *must* be generated whenever the associated object is modified. This includes changes to an <INFO-list> chunk. If an internal object, such as an instrument or wave is changed, one must change the DLSID for *both* the object (instrument or wave) *and* the enclosing <DLS-form> collection Note that if <vers-ck> version chunks and <dlid-ck> identifiers are both used in the same file, changing the <vers-ck> requires a concurrent change to all associated <dlid-ck> values.

A DLS-aware utility which is importing a DLS object from one <DLS-form> collection into another <DLS-form> collection should not change the DLSID of the object being copied. In other words, if one is copying a <ins-list> or <wave-list> chunk from one <DLS-form> into another, one should *not* change the <dlid-ck> within the <ins-list> or <wave-list>, since this data is not being changed. Of course, one *must* change the collection DISID (the top-level <dlid-ck> within the target <DLS-form>), because importing an object into the collection changes that collection.

Wave files need special handling, since they are often generated or processed by applications which are not aware of DLSID chunks. A ***raw*** wave file is a stand-alone <WAVE-form> file, distinguished by the initial sequence 'RIFF'[size]'WAVE'. A ***DLS*** wave file is an embedded <wave-list> chunk, normally kept inside a <DLS-form> collection, distinguished by the initial sequence 'LIST'[size]'wave' (upper case vs. lower case is significant).

When importing a raw <WAVE-form> wave file into a <DLS-form> collection, generate and insert a new <dlid-ck> for the wave file (replacing any pre-existing <dlid-ck> within the raw wave file).

When exporting a DLS wave file from a <DLS-form> collection to a raw <WAVE-form> wave file, the embedded <dlid-ck> should be stripped out.

FROM	TO	Comment
<DLS-form>	<DLS-form>	Keep DLSID for instrument or wave file (data is unchanged) Change DLSID for target <DLS-form>
<WAVE-form> (raw wave file)	<DLS-form>	Remove pre-existing DLSID from raw wave file, if any Generate new DLSID for wave file object within DLS collection.
<DLS-form>	<WAVE-form>	Remove DLSID during export

It is recommended that each <dlid-ck> DLSID chunk should be placed as the *first* chunk within the enclosing LIST or FORM chunk, in order to facilitate searching. However, this ordering should not be assumed.

File Examples

Generic DLS Level 1 File

This instrument collection contains two instruments, an ocean surf sound and a flute instrument. The ocean surf is loaded in bank 1 instrument 1 and responds across the whole key range. The flute sound is loaded in bank 1 instrument 2 and consists of two regions with one region for keys from 0-63 and one region for keys from 64-127.

```
RIFF 'DLS '
    <vers>      (1,0,0,23)
    LIST 'INFO'
          inam  "Demo Collection with Ocean Surf and Flute"
          icop   "Copyright © 1996 MIDI Manufacturers Association"
    <dlid>   (globally unique identifier for entire collection)
    <colh>  (2 Instruments in this collection)
    LIST 'lins'
          LIST 'ins '
                LIST 'INFO'
                      inam  "Ocean Surf"
                <dlid>   (globally unique identifier for "Ocean Surf" instrument)
                <insh>  (1 Region, location bank 5600h instrument 1)
                LIST 'lrgn'
                      LIST 'rgn '
                            <rgnh> ( responds to all keys from 0-127, velocities from 0-127)
                            <wsmp>  (specifies loop points and midi root note)
                            <wlnk>          (specifies Wave #1 (index = 0) in the Pool Table)
                LIST 'lart'
                      <art1>    (specifies the Level 1 articulation for this instrument)
          LIST 'ins '
                LIST 'INFO'
                      inam  "My Flute"
                <dlid>   (globally unique identifier for "My Flute" instrument)
                <insh> (2 Regions, location bank 5600h instrument 2)
                LIST 'lrgn'
                      LIST 'rgn '
                            <rgnh> ( responds to all keys from 0-63, velocities from 0-127)
                            <wsmp>  (specifies loop points and midi root note)
                            <wlnk>          (specifies Wave #2 (index= 1) in the Pool Table)
                      LIST 'rgn '
                            <rgnh> ( responds to all keys from 64-127, velocities from 0-127)
                            <wsmp>  (specifies loop points and midi root note)
                            <wlnk>          (specifies Wave #3 (index = 2) in the Pool Table)
                LIST 'lart'
                      <art1>    (specifies the Level 1 articulation for this instrument)
    <ptbl> [3 entries, [0,offset to surf][1,offset to flute lower][2,offset to flute upper]]
    LIST 'wvpl'
          LIST 'wave' (Ocean surf sound)
                <dlid>   (globally unique identifier for "Ocean Surf" wave file)
          LIST 'wave' (Flute for lower half of keyboard)
                <dlid>   (globally unique identifier for lower-register flute wave file)
          LIST 'wave' (Flute for upper half of keyboard)
                <dlid>   (globally unique identifier for upper-register flute wave file)
```

DLS Level 1 File With 3rd Party Extensions

The following example shows the exact same file but with additional **Proprietary Chunks** embedded for a more complex synthesis engine. The file is still compatible with DLS Level 1. The additional chunks embedded in this case are all assigned to manufacturer ID “MMA”, and used as follows:

<MMAX> A third party effect chunk which contains parameters for overall effects which can be applied to a full collection of instruments.

<MMA1> A third party articulation chunk which specifies parameters which can be applied to each instrument in a collection, or each region within an instrument, depending on where the chunk is located in the DLS file structure. Since the chunk appears at different locations, the same ID may be used.

<MMA2> Another third party articulation chunk but intended for a different DLS device from manufacturer “MMA”. Since this is the second proprietary chunk to be used at the same level, it must have a unique ID.

```
RIFF 'DLS '
    LIST 'INFO'
          inam  "Demo Collection with Ocean Surf and Flute"
          icop  "Copyright © 1996 MIDI Manufacturers Association"
    <dlid>  (globally unique identifier for entire collection)
    <colh>  (2 Instruments in this collection)
    LIST 'lins'
          LIST 'ins '
                LIST 'INFO'
                      inam  "Ocean Surf"
                <dlid>  (globally unique identifier for "Ocean Surf" instrument)
                <insh>  (1 Region, location bank 5600h instrument 1)
                LIST 'lrgn'
                      LIST 'rgn '
                            <rgnh> ( responds to all keys from 0-127, velocities from 0-127)
                            <wsmp>  (specifies loop points and midi root note)
                            <wlnk>      (specifies Wave #1 (index = 0) in the Pool Table)
                            LIST 'lart'
                            <MMA1> ( specifies 3rd party region level articulation)
                LIST 'lart'
                      <art1>   (specifies the Level 1 articulation for this instrument)
                      <MMA1> ( specifies 3rd party instrument level articulation)
          LIST 'ins '
                LIST 'INFO'
                      inam  "My Flute"
                <dlid>  (globally unique identifier for "My Flute" instrument)
                <insh> (2 Regions, location bank 5600h instrument 2)
                LIST 'lrgn'
                      LIST 'rgn '
                            <rgnh> ( responds to all keys from 0-63, velocities from 0-127)
                            <wsmp>  (specifies loop points and midi root note)
                            <wlnk>      (specifies Wave #2 (index = 1) in the Pool Table)
                            LIST 'lart'
                            <MMA2> ( specifies 3rd party region level articulation for a different device)
```

(continues …)

```
                  LIST ‘rgn ‘
                        <rgnh> ( responds to all keys from 64-127, velocities from 0-127)
                        <wsmp>  (specifies loop points and midi root note)
                        <wlnk>        (specifies Wave #3 (index = 2) in the Pool Table)
                        LIST ‘lart’
                        <MMA1> ( specifies 3rd party region level articulation)
            LIST ‘lart’
                  <art1>    (specifies the Level 1 articulation for this instrument)
                  <MMA1> ( specifies 3rd party instrument level articulation)
<ptbl> [3 entries, [0,offset to surf][1,offset to flute lower][2,offset to flute upper]]
LIST ‘wvpl’
      LIST ‘wave’ (Ocean surf sound)
            <dlid>   (globally unique identifier for “Ocean Surf” wave file)
      LIST ‘wave’ (Flute for lower half of keyboard)
            <dlid>   (globally unique identifier for lower-register flute wave file)
      LIST ‘wave’ (Flute for upper half of keyboard)
            <dlid>   (globally unique identifier for upper-register flute wave file)
<MMAX> ( specifies 3rd party effects for the complete collection)
```

Proprietary Chunk IDs

As mentioned earlier in this book (and described in the File Examples on pages 54-55), manufacturers of DLS devices may embed product-specific data into a DLS file using a Proprietary Chunk ID.

These IDs are assigned exclusively by the MMA, in order to prevent any potential error or conflicts between manufacturers accidentally using the same ID.

Manufacturers wishing to obtain their own Proprietary Chunk ID should visit the MMA web site (www.midi.org) or contact the MMA for an application form.

Proprietary Chunk IDs work as follows:

1. The ID is alpha-numeric and 4 bytes in length.
2. The first three digits are assigned by the MMA. Manufacturers may receive three digits of their own choosing, if available and approved by the MMA. (Some choices are reserved).
3. The fourth digit is assigned by the manufacturer, and should indicate a specific product (or product group with consistent features).
4. Allowable characters are A-Z and 0-9, supporting 36 different products or product groups.
5. Manufacturers needing more variations may apply for an additional ID.

Appendix A — Parameter Units

32-bit Time Cents
where tc = Time Cents:

tc = log2(time[secs]) * 1200*65536
time[secs] = 2^(tc/(1200*65536))

32-bit Absolute Pitch Cents
where pc = Pitch Cents, f = linear frequency:

pc = (log2(f/440) * 1200 + 6900) * 65536
f = 2^((pc/65536 - 6900)/1200) * 440

16-bit Relative Pitch Cents
where pc = Pitch Cents, f = linear frequency, F = reference frequency:

pc = log2(f/F) * 1200
f = 2^(pc/(1200)) * F

32-bit Relative Gain
where cb = Centibels, v = voltage, V = reference voltage:

cb = log10(v/V) * 200 * 65536
v = 10^(cb/(200*65536)) * V

0.1 % units
where pcunits = PercentUnits, percent = value between 0 and 100

pcunits = percent * 10 * 65536
percent = pcunits / (10 * 65536)

Appendix B — Transform and Pan Functions

The following chart shows the Concave Transform which is applied to Volume, Expression, and Velocity inputs. The plot shows the output amplitude in percent of full scale plotted against the control input.

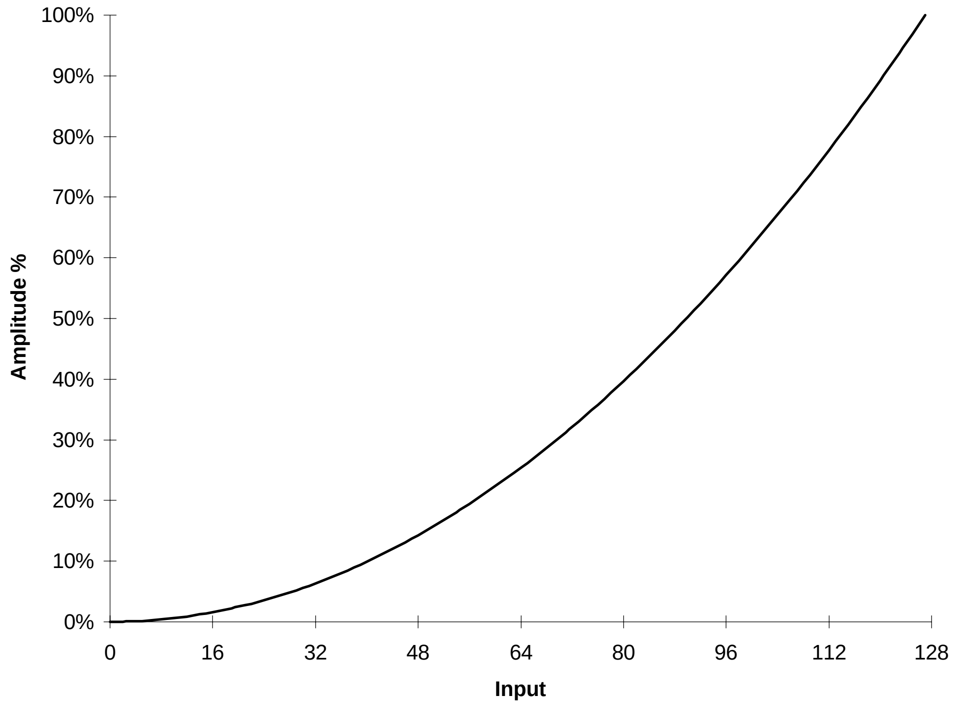


Figure 18. **Concave Transform**

The following chart depicts the Left and Right volume outputs plotted against the Pan Control (MIDI Controller 10):

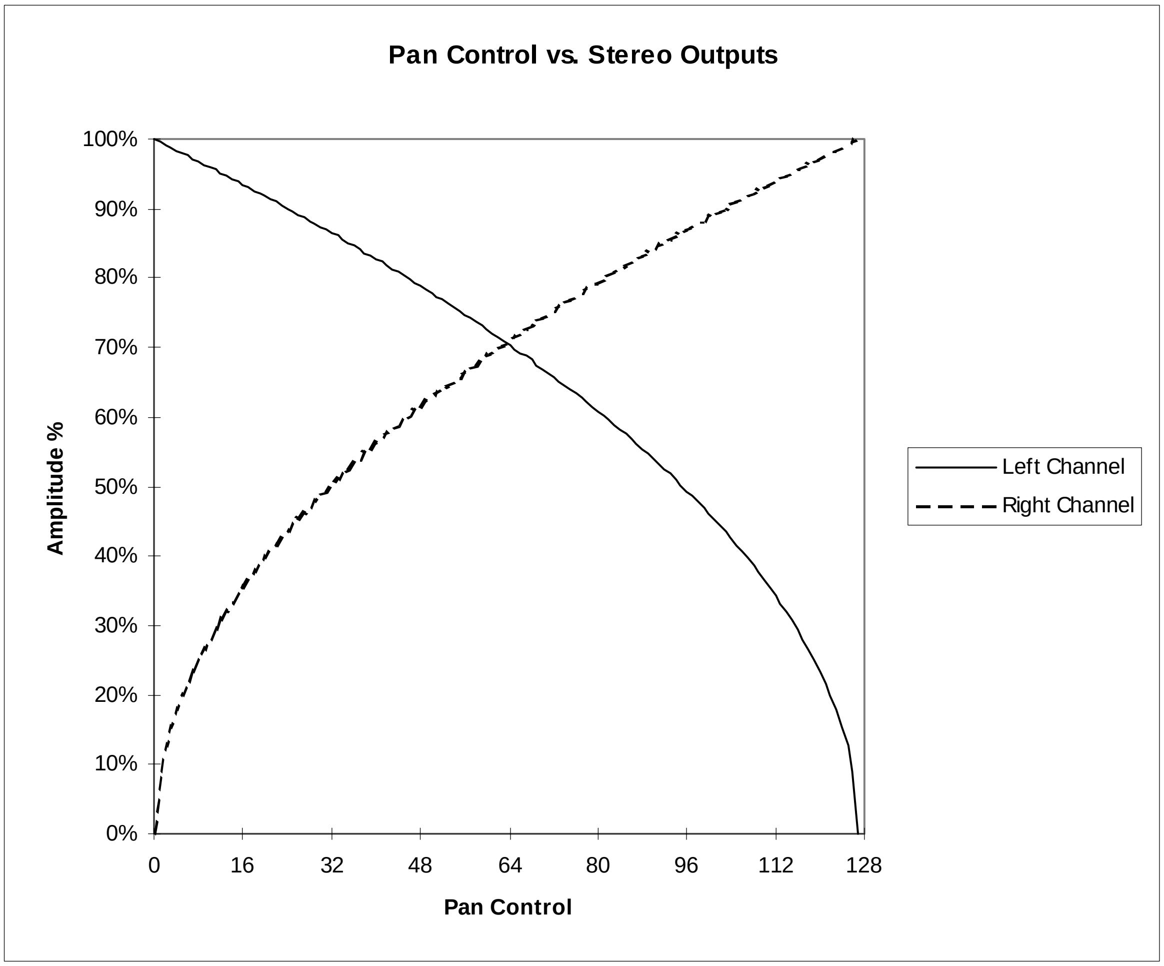


Figure 19. **Pan Control versus Stereo Outputs**

Appendix C — DLS Header File

```
//==========================================================================;
//
//  dls1.h
//
//  Description:
//
//  Interface defines and structures for the Instrument Collection Form
//  RIFF DLS (Level 1).
//
//
//  Written by Sonic Foundry 1996.  Released for public use.
//
//==========================================================================;

#ifndef _INC_DLS1
#define _INC_DLS1

//////////////////////////////////////////////////////////////////////////
//
//
// Layout of an instrument collection:
//
//
// RIFF [] 'DLS ' [dlid,colh,INSTLIST,WAVEPOOL,INFOLIST]
//
// INSTLIST
// LIST [] 'lins'
//               LIST [] 'ins ' [dlid,insh,RGNLIST,ARTLIST,INFOLIST]
//               LIST [] 'ins ' [dlid,insh,RGNLIST,ARTLIST,INFOLIST]
//               LIST [] 'ins ' [dlid,insh,RGNLIST,ARTLIST,INFOLIST]
//
// RGNLIST
// LIST [] 'lrgn'
//               LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]
//               LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]
//               LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]
//
// ARTLIST
// LIST [] 'lart'
//         'art1' level 1 Articulation connection graph
//         'art2' level 2 Articulation connection graph
//         '3rd1' Possible 3rd party articulation structure 1
//         '3rd2' Possible 3rd party articulation structure 2 .... and so on
//
// WAVEPOOL
// ptbl [] [pool table]
// LIST [] 'wvpl'
//               [path],
//               [path],
//               LIST [] 'wave' [dlid,RIFFWAVE]
//               LIST [] 'wave' [dlid,RIFFWAVE]
//               LIST [] 'wave' [dlid,RIFFWAVE]
//               LIST [] 'wave' [dlid,RIFFWAVE]
//               LIST [] 'wave' [dlid,RIFFWAVE]
//
// INFOLIST
// LIST [] 'INFO'
//               'icmt' 'One of those crazy comments.'
//               'icop' 'Copyright (C) 1996 Sonic Foundry'
//
//////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////
// FOURCC's used in the DLS file
//////////////////////////////////////////////////////////////////////////

#define FOURCC_DLS   mmioFOURCC('D','L','S',' ')
#define FOURCC_DLID  mmioFOURCC('d','l','i','d')
#define FOURCC_COLH  mmioFOURCC('c','o','l','h')
#define FOURCC_WVPL  mmioFOURCC('w','v','p','l')
#define FOURCC_PTBL  mmioFOURCC('p','t','b','l')
```

```
#define FOURCC_PATH  mmioFOURCC('p','a','t','h')
#define FOURCC_wave  mmioFOURCC('w','a','v','e')
#define FOURCC_LINS  mmioFOURCC('l','i','n','s')
#define FOURCC_INS   mmioFOURCC('i','n','s',' ')
#define FOURCC_INSH  mmioFOURCC('i','n','s','h')
#define FOURCC_LRGN  mmioFOURCC('l','r','g','n')
#define FOURCC_RGN   mmioFOURCC('r','g','n',' ')
#define FOURCC_RGNH  mmioFOURCC('r','g','n','h')
#define FOURCC_LART  mmioFOURCC('l','a','r','t')
#define FOURCC_ART1  mmioFOURCC('a','r','t','1')
#define FOURCC_WLNK  mmioFOURCC('w','l','n','k')
#define FOURCC_WSMP  mmioFOURCC('w','s','m','p')
#define FOURCC_VERS  mmioFOURCC('v','e','r','s')

///////////////////////////////////////////////////////////////////////////
// Articulation connection graph definitions
///////////////////////////////////////////////////////////////////////////

// Generic Sources
#define CONN_SRC_NONE              0x0000
#define CONN_SRC_LFO               0x0001
#define CONN_SRC_KEYONVELOCITY     0x0002
#define CONN_SRC_KEYNUMBER         0x0003
#define CONN_SRC_EG1               0x0004
#define CONN_SRC_EG2               0x0005
#define CONN_SRC_PITCHWHEEL        0x0006

// Midi Controllers 0-127
#define CONN_SRC_CC1               0x0081
#define CONN_SRC_CC7               0x0087
#define CONN_SRC_CC10              0x008a
#define CONN_SRC_CC11              0x008b

// Generic Destinations
#define CONN_DST_NONE              0x0000
#define CONN_DST_ATTENUATION       0x0001
#define CONN_DST_RESERVED          0x0002
#define CONN_DST_PITCH             0x0003
#define CONN_DST_PAN               0x0004

// LFO Destinations
#define CONN_DST_LFO_FREQUENCY     0x0104
#define CONN_DST_LFO_STARTDELAY    0x0105

// EG1 Destinations
#define CONN_DST_EG1_ATTACKTIME    0x0206
#define CONN_DST_EG1_DECAYTIME     0x0207
#define CONN_DST_EG1_RESERVED      0x0208
#define CONN_DST_EG1_RELEASETIME   0x0209
#define CONN_DST_EG1_SUSTAINLEVEL  0x020a

// EG2 Destinations
#define CONN_DST_EG2_ATTACKTIME    0x030a
#define CONN_DST_EG2_DECAYTIME     0x030b
#define CONN_DST_EG2_RESERVED      0x030c
#define CONN_DST_EG2_RELEASETIME   0x030d
#define CONN_DST_EG2_SUSTAINLEVEL  0x030e

#define CONN_TRN_NONE              0x0000
#define CONN_TRN_CONCAVE           0x0001

typedef struct _DLSID {
  ULONG    ulData1;
  USHORT   usData2;
  USHORT   usData3;
  BYTE     abData4[8];
} DLSID, FAR *LPDLSID;

typedef struct _DLSVERSION {
  DWORD    dwVersionMS;
  DWORD    dwVersionLS;
}DLSVERSION, FAR *LPDLSVERSION;

typedef struct _CONNECTION {
  USHORT   usSource;
  USHORT   usControl;
  USHORT   usDestination;
  USHORT   usTransform;
  LONG     lScale;
}CONNECTION, FAR *LPCONNECTION;


// Level 1 Articulation Data

typedef struct _CONNECTIONLIST {
  ULONG    cbSize;           // size of the connection list structure
  ULONG    cConnections;     // count of connections in the list
  } CONNECTIONLIST, FAR *LPCONNECTIONLIST;



/////////////////////////////////////////////////////////////////////////////
// Generic type defines for regions and instruments
/////////////////////////////////////////////////////////////////////////////

typedef struct _RGNRANGE {
  USHORT usLow;
  USHORT usHigh;
}RGNRANGE, FAR * LPRGNRANGE;

#define F_INSTRUMENT_DRUMS      0x80000000

typedef struct _MIDILOCALE {
  ULONG ulBank;
  ULONG ulInstrument;
}MIDILOCALE, FAR *LPMIDILOCALE;

/////////////////////////////////////////////////////////////////////////////
// Header structures found in an DLS file for collection, instruments, and
// regions.
/////////////////////////////////////////////////////////////////////////////

#define F_RGN_OPTION_SELFNONEXCLUSIVE  0x0001

typedef struct _RGNHEADER {
  RGNRANGE RangeKey;            // Key range
  RGNRANGE RangeVelocity;       // Velocity Range
  USHORT   fusOptions;          // Synthesis options for this range
  USHORT   usKeyGroup;          // Key grouping for non simultaneous play
                                // 0 = no group, 1 up is group
                                // for Level 1 only groups 1-15 are allowed
}RGNHEADER, FAR *LPRGNHEADER;

typedef struct _INSTHEADER {
  ULONG      cRegions;          // Count of regions in this instrument
  MIDILOCALE Locale;            // Intended MIDI locale of this instrument
}INSTHEADER, FAR *LPINSTHEADER;

typedef struct _DLSHEADER {
  ULONG      cInstruments;      // Count of instruments in the collection
}DLSHEADER, FAR *LPDLSHEADER;

////////////////////////////////////////////////////////////////////////////////
// definitions for the Wave link structure
////////////////////////////////////////////////////////////////////////////////

// ****  For level 1 only WAVELINK_CHANNEL_MONO is valid  ****
// ulChannel allows for up to 32 channels of audio with each bit position
// specifiying a channel of playback

#define WAVELINK_CHANNEL_LEFT    0x0001l
#define WAVELINK_CHANNEL_RIGHT   0x0002l

#define F_WAVELINK_PHASE_MASTER  0x0001

typedef struct _WAVELINK { // any paths or links are stored right after struct
  USHORT   fusOptions;     // options flags for this wave
  USHORT   usPhaseGroup;   // Phase grouping for locking channels
  ULONG    ulChannel;      // channel placement
  ULONG    ulTableIndex;   // index into the wave pool table, 0 based
}WAVELINK, FAR *LPWAVELINK;

#define POOL_CUE_NULL  0xffffffffl

typedef struct _POOLCUE {
//  ULONG    ulEntryIndex;   // Index entry in the list
```

```
  ULONG    ulOffset;       // Offset to the entry in the list
}POOLCUE, FAR *LPPOOLCUE;

typedef struct _POOLTABLE {
  ULONG    cbSize;            // size of the pool table structure
  ULONG    cCues;             // count of cues in the list
  } POOLTABLE, FAR *LPPOOLTABLE;

//////////////////////////////////////////////////////////////////////////////
// Structures for the "wsmp" chunk
//////////////////////////////////////////////////////////////////////////////

#define F_WSMP_NO_TRUNCATION     0x0001l
#define F_WSMP_NO_COMPRESSION    0x0002l


typedef struct _rwsmp {
  ULONG   cbSize;
  USHORT  usUnityNote;         // MIDI Unity Playback Note
  SHORT   sFineTune;           // Fine Tune in log tuning
  LONG    lAttenuation;        // Overall Attenuation to be applied to data
  ULONG   fulOptions;          // Flag options
  ULONG   cSampleLoops;        // Count of Sample loops, 0 loops is one shot
  } WSMPL, FAR *LPWSMPL;


// This loop type is a normal forward playing loop which is continually
// played until the envelope reaches an off threshold in the release
// portion of the volume envelope

#define WLOOP_TYPE_FORWARD   0

typedef struct _rloop {
  ULONG cbSize;
  ULONG ulType;              // Loop Type
  ULONG ulStart;             // Start of loop in samples
  ULONG ulLength;            // Length of loop in samples
} WLOOP, FAR *LPWLOOP;

#endif //_INC_DLS1
```

Appendix D — DLS Level 1 Wave File Format

The <wave-list> wave file chunk is defined as follows. **Programs must expect (and ignore) any unknown chunks encountered**, **as with all RIFF forms**. However, <fmt-ck> must always occur before <data-ck>, and both <fmt-ck> and <data-ck> chunks are mandatory in a <wave-file>.

A <wave-list> is similar to an embedded wave file, as would be generated by a wave editing program. These wave file chunks are referenced from the <rgn-list> chunks inside each <ins-list> chunk, and accessed via the <poolcue> entries in the <ptbl-ck> pool table chunk. The <wave-list> data is equivalent to that found in a normal <WAVE-form> WAV file, with two exceptions. First, the initial three DWORD values in a normal <WAVE-form> WAV file (which are 'RIFF' [size] 'WAVE') are replaced with a <wave-list> chunk header ('LIST' [size] 'wave'). Second, an optional <dlid-ck> is inserted as the first element following the <wave-list> chunk header, in order to provide a globally unique identifier for the <wave-list> wave file data.

DLS Level 1 does not require the use of <fact-ck>, <cue-ck>, <playlist-ck> or <assoc-data-list> chunks; therefore, they are intentionally left out of the specification.

```
<wave-list>       →       LIST( 'wave'            // note: 'wave' is lower case
                                    [<dlid-ck>]
                                    <fmt-ck>
                                    [<wsmp-ck>]
                                    [<fact-ck>]
                                    [<cue-ck>]
                                    [<playlist-ck>]
                                    [<assoc-data-list>]
                                    [<INFO-list>]
                                    <data-ck>
                              )
```

<fmt-ck>, Format Chunk

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>
                                   )

<format-specific-fields>  →    <PCM-format-specific>          // only valid format for DLS Level 1

<PCM-format-specific>     →    struct
                               {
                               WORD         wBitsPerSample;
                               }

<common-fields>           →    struct
                               {
                               WORD         wFormatTag;
                               WORD         wChannels;
                               DWORD        dwSamplesPerSec;
                               DWORD        dwAvgBytesPerSec;
                               WORD         wBlockAlign;
                               }
```

The <PCM-format-specific> chunk:

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. DLS level 1 supports only 8 or 16 bit samples.

The <common-fields> chunk:

Field	Description
wFormatTag	A number indicating the WAVE format category of the file. The content of the <format-specific-fields> portion of the fmt chunk, and the interpretation of the waveform data, depend on this value. DLS Level 1 only supports WAVE_FORMAT_PCM (0x0001) Microsoft Pulse Code Modulation (PCM) format
wChannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo. DLS Level 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.

<data-ck>, Data Chunk

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 PCM wave data. See Data Packing for WAVE_FORMAT_PCM Files and Data Format of the WAVE_FORMAT_PCM Samples sections

Data Packing for WAVE_FORMAT_PCM Files

In a single-channel <wave-file>, samples are stored consecutively. For stereo <wave-file>, channel 0 represents the left channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel <wave-file>, samples are interleaved. Only channel 0 is supported for DLS Level 1. Stereo file format are shown for completeness.

The following diagrams show the data packing for a 8-bit mono and stereo WAVE files:

Data Packing for 8-Bit Mono PCM

Sample 1	Sample 2	Sample 3	Sample 4
Channel 0	Channel 0	Channel 0	Channel 0

Data Packing for 8-Bit Stereo PCM

Sample 1	Sample 1	Sample 2	Sample 2
Channel 0	Channel 1	Channel 0	Channel 0
(left)	(right)	(left)	(right)

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:

Data Packing for 16-Bit Mono PCM

Sample 1	Sample 1	Sample 2	Sample 2
Channel 0	Channel 0	Channel 0	Channel 0
low-order	high-order	low-order	high-order
byte	byte	byte	byte

Data Packing for 16-Bit Stereo PCM

Sample 1	Sample 1	Sample 1	Sample 1
Channel 0	Channel 0	Channel 1	Channel 1
(left)	(left)	(right)	(right)
low-order	high-order	low-order	high-order
byte	byte	byte	byte

Data Format of the WAVE_FORMAT_PCM Samples

The data format and maximum and minimums values for PCM waveform samples of various sizes are as follows:

Sample Size	Data Format	Maximum Value	Minimum Value	Midpoint Value
8 bit	Unsigned	255 (0xFF)	0	128 (0x80)
16 bit	Signed	32767 (0x7FFF)	-32768 (0x8000)	0

Appendix E — Instrument Object Hierarchy and <DLS-form>

The following charts show how the instrument object hierarchies described in the DLS Device Architecture specification correspond to particular chunks within the <DLS-form> file.

Object Hierarchy: Level 1 Melodic Instrument	Corresponding <DLS-form> chunks
Melodic Instrument	<ins-list> instrument, <insh-ck> instrument header, <INFO-list>
Global	<lart-list> list of articulators
Connection Blocks	<art1-ck> articulator
Region (1..16)	<lrgn-list> list of regions, holding <rng-list> region chunks
Wave Link	<wlnk-ck> wave link
Wave Sample	<wsmp-ck> wave sample
Wave File	<wave-list> wave file
Wave Sample (optional)	<wsmp-ck> wave sample
Wave Data	<data-ck>

Object Hierarchy: Level 1 Drum Kit	Corresponding <DLS-form> chunks
Drum Kit	<ins-list> instrument, <insh-ck> instrument header, <INFO-list>
Region (1..128)	<lrgn-list> list of regions, holding <rng-list> region chunks
Wave Link	<wlnk-ck> wave link
Wave Sample	<wsmp-ck> wave sample
Wave File	<wave-list> wave file
Wave Sample (optional)	<wsmp-ck> wave sample
Wave Data	<data-ck>
Region Articulation	<lart-list> list of articulators
Connection Blocks	<art1-ck> articulator

References

- The MIDI Manufacturers Association, “DLS Protocol and Messaging Guidelines” (proposed, 1997)
- The MIDI Manufacturers Association, "General MIDI System Level 1" (adopted, 1994)
- The MIDI Manufacturers Association, “GM Developer Guidelines and Survey” (revised, 1998)

ERRATA for version 1.1

"Attenuation" vs. "Gain" (entire document)

The DLS Level 1 (version 1) document makes extensive reference to attenuation and the destination connection "DST_ATTEN". It was the original intent of the document to use positive units to reflect attenuation applied to the destination, and in fact, the example given beginning on p. 31 (???) makes this assumption.

However, due to an error that was overlooked in development, the MMA's Synth/Author DLS file development tool incorrectly implemented the attenuation field in the opposite sense, in that negative lScale values applied to DST_ATTEN will attenuate the signal. As a result, most DLS Level 1 synthesizers (designed using Synth/Author) were implemented in the same manner. By way of example, to apply 3dB of attenuation from the LFO to DST_ATTEN, the lScale value is 0xFFE20000, or -30 cB (centibels).

Rather than take a dogmatic approach to this problem and insist that the implementations in the field are wrong, the MMA has elected to take a more pragramatic approach and modify the document to reflect common practice. Given the negative sense of the lScale values, it seems logical to refer to them in terms of gain, rather than attenuation. This is reflected in the DLS Level 2 document, where the DST_ATTEN destination is now referred to as DST_GAIN, and all discussion refers to gain rather than attenuation.

Therefore, please bear in mind when reviewing the DLS Level 1 specification, that all units of attenuation should be thought of in terms of gain, and gain is therefore always limited to the range of 0dB to negative infinity.

Note Exclusivity (page 7)

In DLS Level 2, the behavior of the synthesizer when a note-exclusive event occurs has been specifically defined by addition of the EG_SHUTDOWNTIME parameter. For DLS Level 1 it was not stated but it is assumed and preferred that the behavior mimic that of the default for DLS Level 2, which is that the oscillator should shut down within 15ms or as quickly as possible without artifacts.

Envelope diagram (page 12) and segment descriptions (page 13-14)

The following diagrams and text replace the existing ones:

EG Attack Time

The attack time determines how long the attack segment lasts from the end of the Delay Time until the envelope hits peak level. When the output of the envelope is applied to gain, the Attack segment affects the output of the oscillator in a linear fashion with respect to amplitude. This requires a slightly more complex coupling to the DCA, since the envelope output will undergo a different transform during the attack segment than during all other segments.

When the envelope output is applied to any other destination, the effect is linear with respect to the units used at that particular destination. For example, when applied to pitch, the envelope will always affect the pitch of the oscillator in a linear fashion with respect to units in Pitch Cents, which is exponential with regard to frequency.

If the attack time is set to zero, the envelope generator should immediately go to peak output. Likewise, any controls to which the envelope is routed should also go to full output. For example, if the attack time of the Volume EG is set to zero, the amplitude of the DCA at the time the first sample is output must be at full scale so that any transients in the sample are faithfully reproduced.

EG Velocity to Attack Scaling

The MIDI velocity scales the duration of the attack segment. Since all times are described in Time Cents, which are logarithmic units, the effect of adding values in Time Cents is multiplicative in actual time. The formula is as follows:

*Attack_Timetimecents = Nominal_Attack_Timetimecents + (Velocity / 128) * Attack_Scalartimecents*

Where *Nominal_Attack_Time* is the attack time constant set by the articulation data, *Velocity* is the MIDI Note-On velocity, and *Attack_Scalar* is the *lScale* value from the *EG Velocity to Attack Time* connection.

Decay Scaling

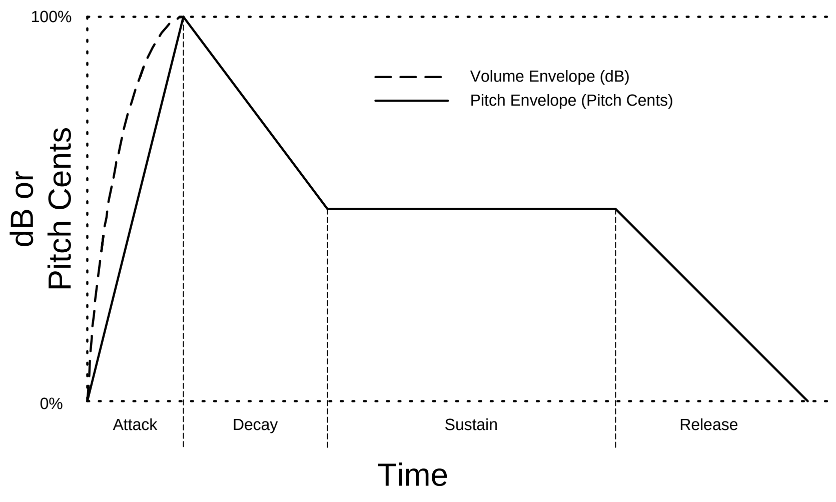
The MIDI note number scales the duration of the decay segment. For example, increasing the note number could decrease the duration of the decay, just as a piano note decays slower for lower notes and faster for higher notes. The formula is as follows:

$$Decay\_Time_{timecents} = Nominal\_Decay\_Time_{timecents} + (Note\_Number / 128) * Decay\_Scalar_{timecents}$$

Where *Nominal_Decay_Time* is the decay time constant set by the articulation data, *Note_Number* is the MIDI note number, and *Decay_Scalar* is the lScale value from the Mod EG Note to Decay connection.

EG Sustain Level

The Decay segment ends when the Sustain Level is reached and the note sustains at this level until the Note-Off event (or Note-On event with a velocity of 0) is received. Raising the Sustain Level will shorten the Decay segment and lengthen the Release segment, while lowering it will have the opposite effect. Sustain is defined as a percentage of the envelope peak in 0.1% increments. Note that Sustain Level is in ordinal units, the actual range is determined by the *lScale* setting of the connection and the unit type is determined by the units used by the destination node. Thus the sustain level as applied to CONN_DST_GAIN is in Gain units (dB), and similarly when applied to CONN_DST_PITCH is in Pitch Cents.



DLS Level 1 Envelope Diagram (corrected)

Response to Reset All Controllers (Page 15)

The AMEI/MMA standard response to Reset All Controllers includes reseting Expression (CC#11). This Specification (and subsequent DLS specifications) incorrectly states that Expression should **not** be reset. By polling manufacturers of DLS devices it was discovered that everyone resets Expression regardless. Therefore this Specification (and all subsequent DLS Specifications) is amended.

<wsmp-ck> description (Page 46)

Only the following item is changed:

fulOptions Specifies flag options for the digital audio sample. Current options are:

F_WSMP_NO_TRUNCATION 0x0001

This option specifies that a synthesis engine is not allowed to truncate the bit depth of the sample if it cannot synthesize at the bit depth of the digital audio. If the NO_TRUNCATION bit is set, the device is not allowed to truncate the data to a bit depth less than that of the original sample. Note that, if truncation is disallowed, the wave may take up more memory in the device and thus may fail to download due to memory constraints. If the bit is clear, the synthesis engine may truncate the bit depth, if required.

F_WSMP_NO_COMPRESSION 0x0002

This option specifies that a synthesis engine is not allowed to use compression in its internal synthesis engine for the digital audio sample. If the NO_COMPRESSION bit is set, the device is not allowed to compress the data. Note that, if compression is disallowed, the wave may take up more memory in the device and thus may fail to download due to memory constraints. If the bit is clear, the synthesis engine may compress the digital audio samples, if required.