Revolution

AX Applications

Version 1.00

"Confidential"

AX Applications

Version 1.00

Contents

Code Examples

Equations

Figures

Tables

Revision History

Revision No.	Date Revised	Items (Chapter)	Description
22-Mar-2006	3/22/2006	-	First release by Nintendo of America Inc.

1 Overview

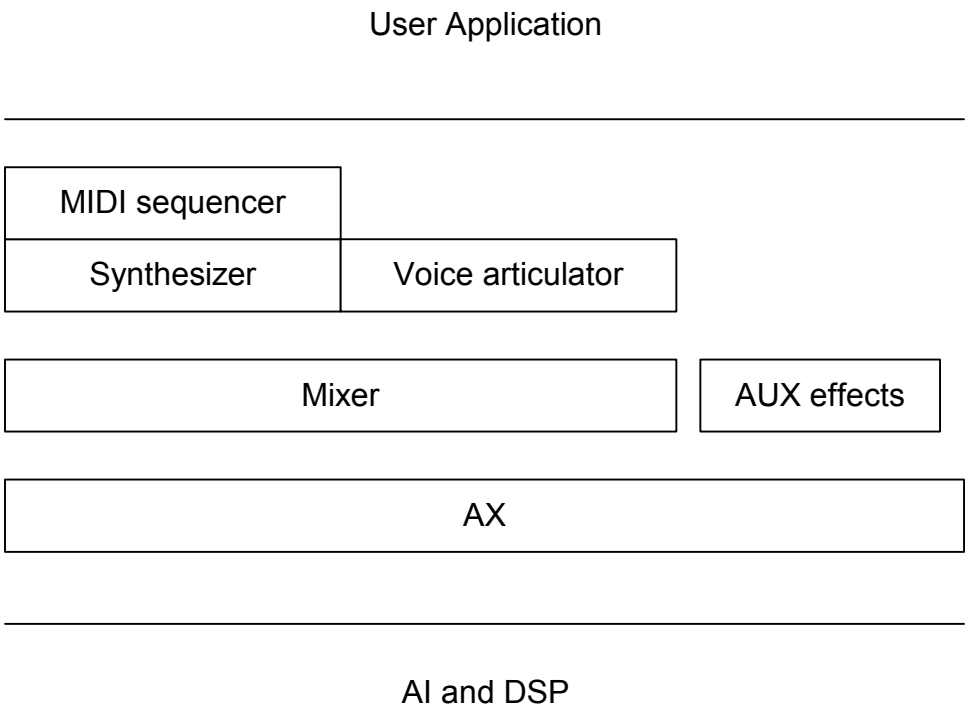
The AX library provides a low-level abstraction for the Revolution audio subsystem shown in Figure 1-1. This layer was designed to provide flexible support for custom audio applications-for example, MIDI synthesizers, mixers, audio streaming interfaces, and sound effect synthesizers.

Some of these applications have been implemented already and are provided as examples in the SDK. These applications include the following:

- MIDI sequencer (SEQ)
- Wavetable synthesizer (SYN)
- Voice articulator (AXART)
- Logical mixer (MIX)
- AUX effects library (AXFX)

This document provides an design overview about these libraries. Developers can examine and modify source code as needed.

Figure 1-1 Applications for AX

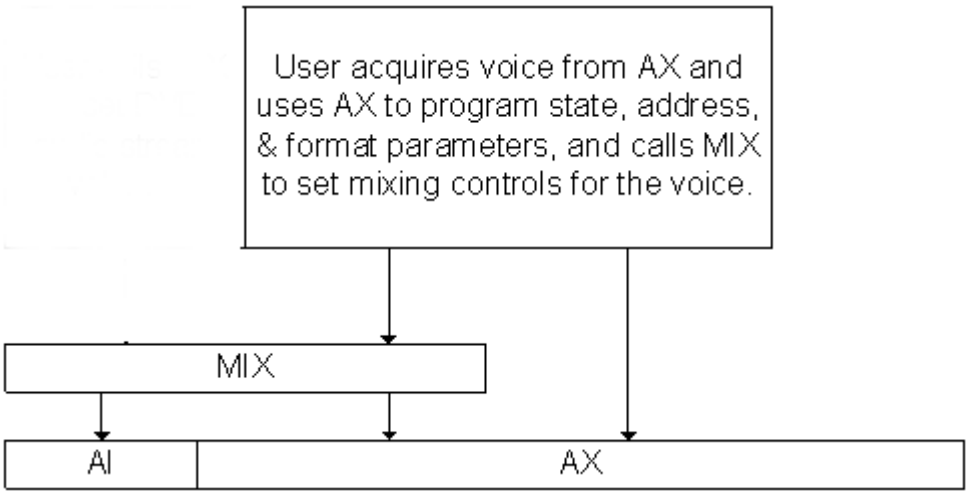


For detailed API descriptions, see the **AX Applications** section in the **Audio** section of the Revolution Reference Manual (HTML).

2 Mixer

As shown in Figure 2-2, the AX mixer application (MIX) is the first layer above AX; MIX provides other applications with logical mixer controls. MIX manages input, AUX send and return paths, panning, muting, and fader controls. MIX also computes volume ramping to prevent zip and pop effects when volume and pan changes are applied to a voice.

Figure 2-2Mixer API Layer



Volume attenuation is measured in 0.1 dB increments for voices. The user application may input absolute or relative values using the API. The mixer computes and updates new mixing values for each audio frame (3 ms). Mixing values can also be set directly to an AX.

For detailed API descriptions, refer to the **Mixer** section in the **Audio** section under AX Applications of the Revolution Reference Manual (HTML).

2.1 Mixer Modes

The MIX lib supports several mixer modes. The appropriate AX mode must be set in conjunction with the mixer mode. Table 2-1 includes descriptions about each mixer mode and lists the corresponding AX mode setting.

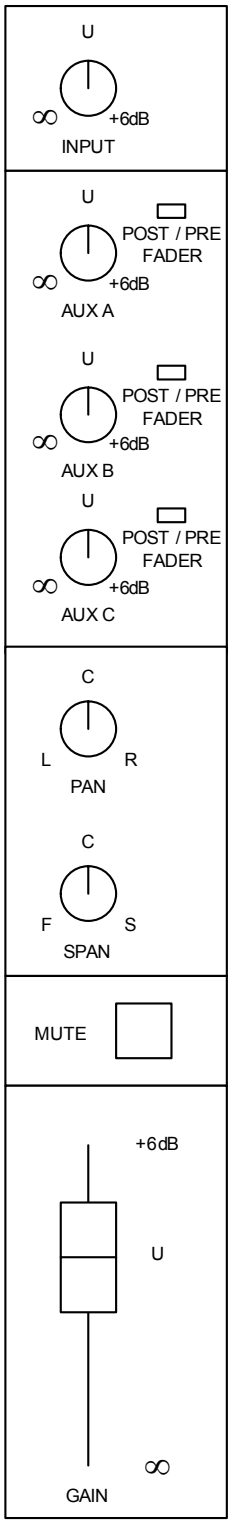
Table 2-1 Mixer Modes

MIX mode	Description	AX Mode
MIX_SOUND_MODE_MONO	For mono applications, the mixer ignores left and right. AX mixing is still in stereo.	AX_MODE_STEREO
MIX_SOUND_MODE_STEREO	For stereo applications, AX mixes the left and right panning Surround sound signal equally. For the mixer, this mode is the same as MIX_SOUND_MODE_SURROUND.	AX_MODE_STEREO
MIX_SOUND_MODE_SURROUND	For Surround applications, AX will encode the Surround sound signal. For the mixer, this mode is the same as MIX_SOUND_MODE_STEREO.	AX_MODE_SURROUND
MIX_SOUND_MODE_DPL2	For Dolby Pro Logic II applications. This mode does not support Aux C.	AX_MODE_DPL2

2.2 Mixer Controls

2.2.1 Voice Mixing Controls

Figure 2-3 Mixer Channels



The mixer supports AX_MAX_VOICES (64) channels of input.

AUX sends/receives support postfader and prefader modes .

Left to right panning is from 0 to 127 (64 = center).

Front to back surround panning from 0 to 127 (64 = center).

Mute switch.

Fader in .01 dB increments .

Volume attenuation is set/adjusted with 0.1dB increments, and pan values are 0 to 127. Gain greater than +6 dB and attenuation less than -90.4 dB will be clamped to +6.0 dB and -90.4dB, respectively.
(-90.4 dB = ∞)

2.2.1.1 Input

The input controls the gain applied to the input signal from the PCM sample. Applications can use this control to apply to the playback an ADSR envelope, tremolo, volume attenuation, or all of these effects. You can adjust attenuation with 0.1 dB precision. This control is set to 0 dB by default.

Note: Using input control takes fewer DSP cycles than fader and pan controls.

2.2.1.2 AuxA, AuxB and AuxC

The AUXn controls set the input gain that will be applied to the specified AUX send path. This control can be changed to postfader or prefader mode. Attenuation is adjusted with 0.1 dB precision.The default value is -90.4 dB in postfader mode for this control. The following AUX attenuation is applied:

post fader $atten_{dB}$ = AUX $atten_{dB}$ + fader $atten_{dB}$

pre fader $atten_{dB}$ = AUX $atten_{dB}$

2.2.1.3 Pan

The pan control distributes the input signal to the left and right channels of the main output. Left is 0, center is 64, and right is 127. The default value is 64 for this control.The following attenuation is applied:

left channel $atten_{dB}$ = -20 * log10(sqrt ((127 - pan) / 127))

right channel $atten_{dB}$ = 20 * log10(sqrt (pan / 127))

2.2.1.4 SPan

The SPan control distributes the input signal to front and Surround channels of the main output. Rear is 0, center is 64, and front is 127. The default value is 127 for this control. The following attenuation is applied:

surround channel $atten_{dB}$ = -20 * log10(sqrt ((127 - pan) / 127))

front channel $atten_{dB}$ = 20 * log10(sqrt (pan / 127))

2.2.1.5 Mute

The mute control sets to the input infinite attenuation. Unmuting the control restores input attenuation. The default value is unmute for this control.

2.2.1.6 Fader

The fader control attenuates output to the main left, right, and surround channels. Attenuation is set or adjusted in 0.1 dB increments. The default value is 0 dB for this control.

2.3 Mixer Application Notes

2.3.1 Volume Clamping

The mixer clamps the actual volume between -90.4 dB and +6.0 dB to accommodate the mixing range for the DSP application. User applications can set faders and input volumes beyond this range without any additional gain/attenuation. The decibel values are expressed as signed 32-bit integers; carefully consider adjustments as they may result in wrap around.

2.3.2 Volume Ramping

The mixer applies per sample volume ramping to mitigate pop or zip effects induced by rapid changes in volume. If a change in the volume multiply value is greater than 160, this ramping will be applied at any mixing point. Any requests with a change in value that is less than 160 will be set directly. The volume level that the user requests will be applied on the subsequent audio frame to allow time for ramping.

2.3.3 Suggested Input and Fader Control Use

Adjusting the input control takes fewer DSP cycles than using the fader and pan controls. Programmers can utilize these controls in the most efficient manner possible; the goal is to balance the load between DSP and CPU cycles per application. Using the input controls take fewer DSP cycles than the fader; however, the CPU would have to calculate attenuation. The following sections discusses a few methods for using these controls.

2.3.3.1 Sound Effects and Audio Streaming

Usually, sound effects and audio streaming set the fader to 0 dB and use the input control to fade volume.

2.3.3.2 Music Synthesizer

While setting the fader to 0 dB and using the pan control based on the MIDI specification, a music synthesizer can calculate attenuation for ADSR envelope, LFO volume modulation, MIDI channel volume, expression pedal volume, and master volume for the input control. However, depending on the application type and if the MIDI master volume, channel volume, and expression pedal rarely changes, you can apply the total sum of these attenuations to the fader to save CPU cycles during playback on the synthesizer.

2.3.3.3 3D Sound Applications

When a sound source moves through space, 3D sound applications can access the pan and span controls frequently. The input control can be modulated as the sum of volume per distance and volume attenuation.

3 AUX Effects

The AX AUX Bus Effects library (AXFX) provides processing for effects on the send and return paths for AuxA and AuxB. These effects include high-quality and low-quality reverb, chorus, and delay. All effects have discrete left, right, and Surround channels.

3.1 High Quality Reverb

This library provides high quality reverb that is appropriate for music and sound effects. Reverb controls are described below in Table 3-2.

Table 3-2 High Quality Reverb Controls

Reverb Control	Description
Predelay	Predelay is a value between 0.0 and 0.1 seconds that specifies the time delay before reverberation starts. This parameter is especially useful for large rooms because it simulates the latency for reflected sound waves. A larger value corresponds to a greater distance between the sound emitter and reflecting surfaces.
Reverb time	Reverb time is a value between 0.01 and 10.0 seconds that specifies the length of time until the reverberation attenuates. For example, a value of 0.01 seconds specifies a very small room and a value of 10.0 specifies a cathedral or stadium.
Coloration	Coloration is a value between 0.0 and 1.0. This value modulates the all-pass filter coefficients of the algorithm and can be used to simulate the acoustic properties of the surfaces in a room.
Damping	Damping is a value between 0.0 and 1.0 that modulates the high frequency attenuation of the algorithm. If damping is 0.0, lower frequencies are predominant and the reverberation becomes more pronounced. As damping approaches 1.0, high frequencies are predominant and the reverberation becomes less pronounced.
Crosstalk	Crosstalk is a value between 0.0 and 1.0 and describes the interaction level between channels. Channels with a value of 0.0 are strictly independent; in other words, the reverberated signal will only be applied to the channel that it was derived from. Channels with a value of 1.0 contributes its reverberated signal to itself and the other channels.
Mix	Mix is a value between 0.0 and 100.0 that specifies the reverberated signal level as a fraction of the output. If the value is 0.0, only the original signal will be audible. If the value is 100.0, only the reverberated signal will be audible.

3.2 Standard Reverb

The standard reverb effect uses fewer CPU cycles than the high-quality reverb. Standard reverb may be sufficient for most applications. This effect has the controls shown in Table 3-3.

Table 3-3 Standard Reverb Controls

Standard Reverb Control	Description
Predelay	Predelay is a value between 0.0 and 0.1 seconds that specifies the time delay before reverberation starts. This parameter is especially useful for large rooms because it simulates the latency for reflected sound waves. A larger value corresponds to a greater distance between the sound emitter and reflecting surfaces.
Reverb time	Time is a value between 0.01 and 10.0 seconds that specifies the length of time until the reverberation attenuates. For example, a value of 0.01 seconds specifies a very small room while a value of 10.0 specifies a cathedral or stadium.
Coloration	Coloration is a value between 0.0 and 1.0. This value modulates the all-pass filter coefficients of the algorithm and can be used to simulate the acoustic properties of the surfaces in a room.
Damping	Damping is a value between 0.0 and 1.0 that modulates the high frequency attenuation of the algorithm. If damping is 0.0, lower frequencies are predominant and the reverberation becomes more pronounced. As damping approaches 1.0, high frequencies are predominant and the reverberation becomes less pronounced.
Mix	Mix is a value between 0.0 and 1.0 that specifies the reverberated signal level as a fraction of the output. If the value is 0.0, only the original signal will be audible. If the value is 1.0, only the reverberated signal will be audible.

3.3 Chorus

The chorus effect is intended for general MIDI type synthesizer applications where a chorus is expected on the AuxB bus. The chorus has the controls shown in Table 3-4.

Table 3-4 Chorus Controls

Chorus Control	Description
Base delay	The value of base delay is between 5 and 15 ms. This value specifies the length of the time delay before the input signal is mixed back into itself.
Variation	The length of the base delay varies over time. The delay value varies between 0 and 5 ms that specifies the maximum amount of time that the base delay can vary. If the base delay equal to its boundary values (5 or 15 milliseconds), this parameter is ignored because the base delay can't vary. **Note:** This parameter is typically referred to as **chorus depth** in other products.
Period	If variation is greater than zero, the variation of the delay is continuous and periodic. Period is a value between 500 and 10000 milliseconds that specifies the period of the variation. **Note:** This parameter is typically referred to as **chorus rate** or **chorus speed** in other products.

3.4 Delay

The delay effect is a standard delay that has feedback and output control. Delay has the controls shown in Table 3-5.

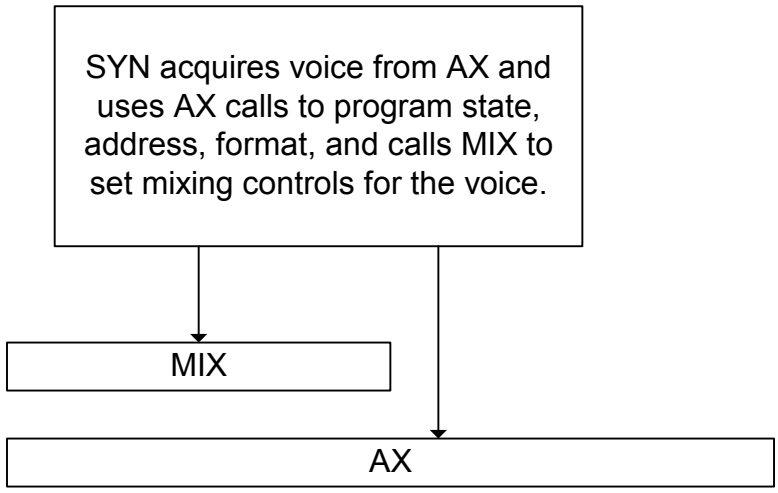
Table 3-5 Delay Controls

Delay Control	Description
Delay	Delay is a value between 10 and 5000 milliseconds that specifies the time delay before the input signal is mixed back into itself. **Note:** Delay values must be specified for the left, right, and surround channels (in that order).
Feedback	Feedback is a value between 0 and 100 that specifies the percentage of the delayed signal that will be mixed with the input signal.
Output	Output is a value between 0 and 100 that specifies the percentage of the delayed signal that will be mixed with the output signal.

4 Synthesizer

The AX synthesizer application (SYN) accepts MIDI input and then applies this input to a wavetable to produce music. This application relies on the AX and MIX APIs.

Figure 4-4 Functional Relationship of SYN to MIX and AX

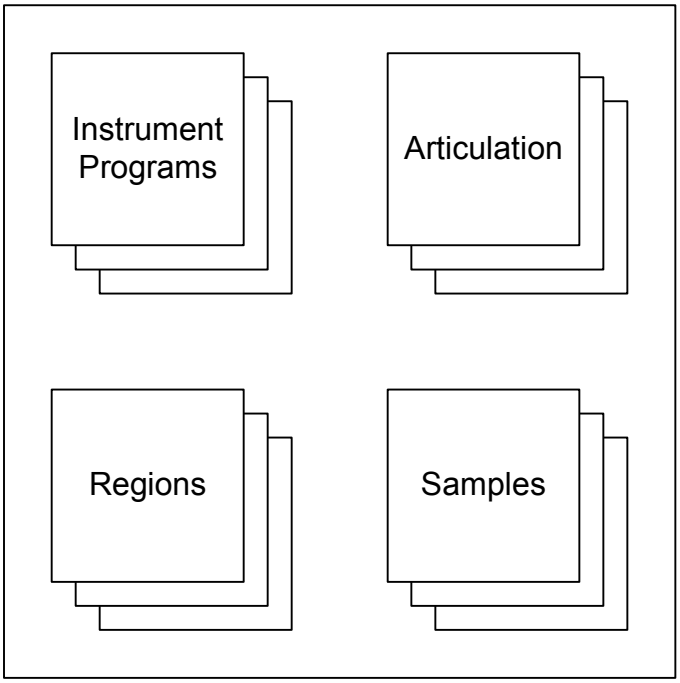


For detailed API descriptions, refer to **Synthesizer** under **AX Applications** in the **Audio** section of the Revolution Reference Manual (HTML).

4.1 Wavetable

A wavetable is a lookup table used by the synthesizer to determine playback parameters for MIDI events. Specifically, the wavetable contains tables for instrument programs, regions, articulators, and sampling data. The following sections explain general wavetable concepts-application methods for this specific synthesizer is covered in Section 4.3, Synthesizer Application Notes, on page 22.

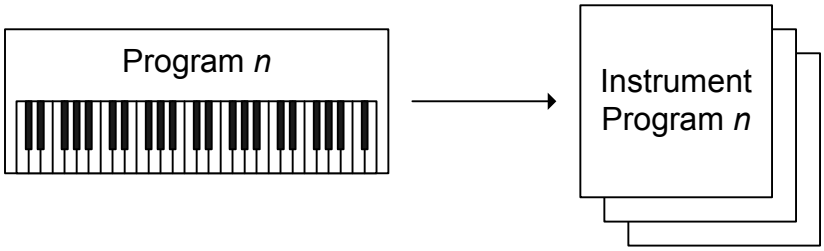
Figure 4-5 Wavetable Concepts



4.1.1 Instrument Programs

Instrument programs specify the wavetable objects that correspond to MIDI program numbers. Typically, MIDI events will program each of the 16 MIDI channels to access a particular instrument. There are two types of instrument programs: melodic and percussion.

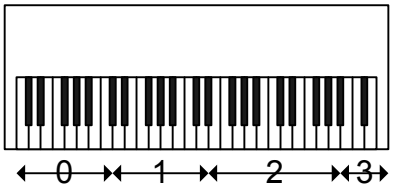
Figure 4-6 Instrument Programs



4.1.1.1 Melodic Instruments

As shown in Figure 4-7, melodic instruments contain *n* key regions to handle incoming MIDI key-on events for specific keys. Typically, the regions are used to match PCM samples with the intended key pitch for a particular instrument. Each contains descriptors that represent the lowest and highest key for that instrument region-the synthesizer is responsible for locating the correct region for the key. Each region also contains pointers to articulation and sampling data.

Figure 4-7 Melodic Key Regions

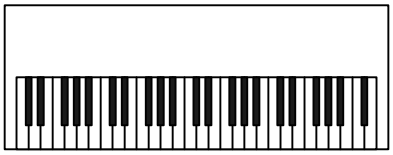


Region 0 key 0 - *a*	Region 1 key *a* - *b*	Region 2 key *b* - *c*	Region 3 key *c* - 127

4.1.1.2 Percussive Instruments

Unlike melodic instruments, percussion instruments contain a region for each key as shown in Figure 4-8. Percussion instruments typically play a unique sample for each key. Each region also contains pointers to articulation and sampling data. In addition to data (as found in melodic instrument regions), percussion instrument regions also specify a key group. Only one instance of a key group can be played at a time. For example, only one sample between the closed, open, and pedal hi-hat keys should be heard at a time because these keys belong to the same key group.

Figure 4-8 Percussive Key Regions



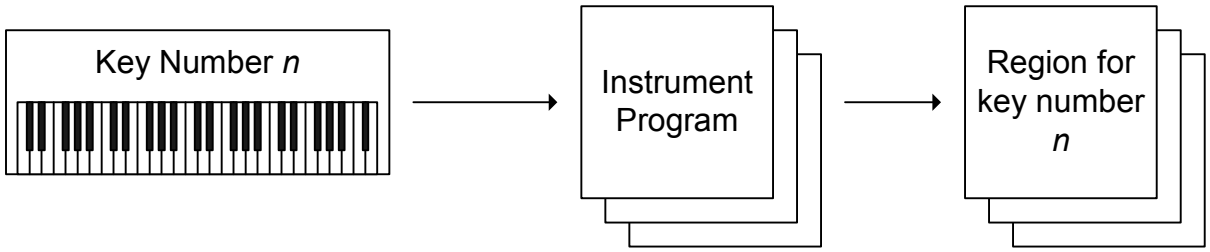
Region 0 key *a*	Region 1 key *b*	Region 2 key *c*	Region 3 key *d*

4.1.2 Regions

A region specifies the playback method for a particular key of an instrument. Specifically, these playback methods are listed below:

- **Low key:** Lowest key that uses the region.
- **High key:** Highest key that uses the region.
- **Normal key:** Key that yields no sample rate conversion for playback, also known as a **unity note**.
- **Key group:** Key yields single instance for a group.
- **Fine tune:** Fine tuning for pitch, usually in cents (1/100).
- **Attenuation:** Attenuation to apply to the sample.
- Pointer to articulator
- Pointer to sample data

Figure 4-9 Regions for Key Numbers

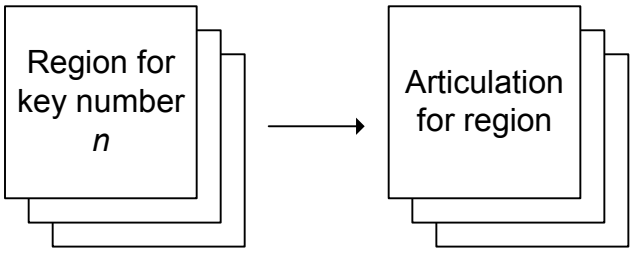


4.1.3 Articulators

Articulators define the playback characteristics for a note. Typically, an articulator describes the following elements:

- LFO start delay, frequency, pitch, and volume
- Volume envelope, attack time, decay time, sustain volume, and release time
- Pitch envelope, attack time, decay time, sustain pitch, release time, and maximum pitch
- Pan (absolute value pan of percussion instruments)

Figure 4-10 Articulation for Regions

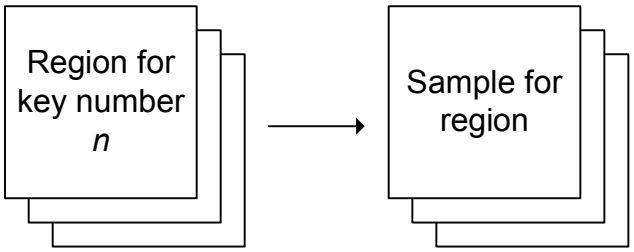


4.1.4 Sampling

Sampling data is used to store information about the particular sound sampling data. Types of sample data include:

- Sampling and base frequency for the sampling
- Sampling data format
- Sampling data size
- Sampling data location in a pool
- Loop data, start and length

Figure 4-11 Sampling Data for Regions



4.2 File Format

In this section, the specific file format for the AX synthesizer application is explained. Each instance of the synthesizer uses a WT and PCM file. These files are created for setting the DLS 1.0 files to be accepted.

4.2.1 File Creation

Wavetable data is created for the synthesizer using the program `DLS1WT.exe`. `DLS1WT.exe` accepts DLS 1.0 files for input. These files can be created by third party DLS editing and preview tools. `DLS1WT.exe` outputs a WT file that contains wavetable data and a PCM file of sampling data.

4.2.2 WT File

The WT file contains optimized wavetable lookup data that will be used by the synthesizer application at runtime. The format of this file is shown in Code 4-1. The definitions for these data objects can be found in `wt.h`.

4.2.2.1 File header

Code 4-1 Wavetable File Header

```
typedef struct WTFILEHEADER
{

    u32 offsetPercussiveInst;
    u32 offsetMelodicInst;
    u32 offsetRegions;
    u32 offsetArticulations;
    u32 offsetSamples;
    u32 offsetLoopContext;

    // data ...

} WTFILEHEADER;
```

The file header contains file offsets to data structures These offsets are described in Table 4 1.

Table 4-6 WT File Offsets

Offset	Description
`u32 offsetPercussiveInst`	Byte offset to an array of 128 `WTINST` data structures. These instruments are intended for percussion programs on MIDI channel 10.
`u32 offsetMelodicInst`	Byte offset to an array of 128 `WTINST` structures. These instruments are intended for melodic programs on all MIDI channels except channel 10.
`u32 offsetRegions`	Byte offset to first `WTREGION` structure in an array of unknown size. These are regions indexed by the `keyRegion` members found in `WTINST`.
`u32 offsetArticulations`	Byte offset to first `WTART` structure in an array of indefinite size. These are articulators indexed by `WTREGION`.
`u32 offsetSamples`	Byte offset to first `WTSAMPLE` structure in an array of indefinite size. These are sample data indexed by `WTREGION`.
`u32 offsetLoopContext`	Intended to specify byte offset to loop context data for ADPCM-compressed sampling data. (*Not used at this time.*)

4.2.2.2 Melodic and Percussion Instruments

Code 4-2 WTINST

```
typedef struct WTINST
{

    u16 keyRegion[128];

} WTINST;
```

The WT file contains an array of 128 `WTINST` structures that store the key region index for 128 melodic instruments or 128 percussion instruments of the corresponding MIDI program number. This file format supports up to 65,534 region descriptors. 0xFFFF is used to specify if there is no region for that note.

`u16 keyRegion[128]` is the region index for all 128 keys of the given instrument.

4.2.2.3 Regions

Code 4-3 WTREGION

```
typedef struct WTREGION
{

    u8  unityNote;
    u8  keyGroup;
    s16 fineTune;
    s32 attn;
    u32 loopStart;
    u32 loopLength;
    u32 articulationIndex;  // articulation index to reference
    u32 sampleIndex;        // sample index to reference

} WTREGION;
```

The WT file contains an array of `WTREGION` structures of an indefinite size. Each region is referenced by one or more corresponding key(s) of an instrument. Table 4-7 describes the different regions:

Table 4-7 Wavetable Regions

Region	Description
`u8 unityNote`	The key number that yields no change in pitch from original sample frequency.
`u8 keyGroup`	Member of single key instance key group. Range is 1 – 15, 0 equals none.
`s16 fineTune`	Fine tuning for this note in cents, 1 = 1cent.
`s32 attn`	Base attenuation for this note, 0x00010000 = 0.1dB.
`u32 loopStart`	Sample count for loop start.
`u32 loopLength`	Total number of samples for loop, including loop points.
`u32 articulationIndex`	Index to articulation for this region in WT file.
`u32 sampleIndex`	Index to sample data for this region in WT file.

4.2.2.4 Articulators

Code 4-4 WTART

```
typedef struct WTART
{
    // LFO
    s32 lfoFreq;
    s32 lfoDelay;
    s32 lfoAtten;
    s32 lfoPitch;
    s32 lfoMod2Atten;
    s32 lfoMod2Pitch;

    // EG1
    s32 eg1Attack;
    s32 eg1Decay;
    s32 eg1Sustain;
    s32 eg1Release;
    s32 eg1Vel2Attack;
    s32 eg1Key2Decay;

    // EG2
    s32 eg2Attack;
    s32 eg2Decay;
    s32 eg2Sustain;
    s32 eg2Release;
    s32 eg2Vel2Attack;
    s32 eg2Key2Decay;
    s32 eg2Pitch;

    // pan
    s32 pan;

} WTART;
```

The WT file contains an array of `WTART` structures of an indefinite size. Each articulator is referenced by one or more region(s) of an instrument. Each articulator is specified in Table 4-8:

Table 4-8 Wavetable Articulators

Articulator	Description
`s32 lfoFreq`	LFO frequency, expressed in delta per 5 ms audio frame for a 64-step sine wave (0x00010000 = 1).
`s32 lfoDelay`	LFO start delay, expressed as the number of 5 ms audio frames prior to starting the LFO.
`s32 lfoAtten`	The maximum volume of LFO attenuation is 1.0 (0x00010000 = 0.1dB).
`s32 lfoPitch`	The maximum LFO pitch in cents is 1.0 (0x00010000 = 1cent).
`s32 lfoMod2Atten`	LFO attenuation to apply to modulation wheel (0x00010000 = 0.1dB). $Attenuation_{dB}$ = `lfoMod2Atten` * (modulation wheel / 128)
`s32 lfoMod2Pitch`	LFO pitch to apply to modulation wheel (0x00010000 = 1cent). $Pitch_{Cents}$ = `lfoMod2Pitch` * (modulation wheel / 128)

Articulator	Description
`s32 eg1Attack`	Scale for ADSR volume envelope attack time. $Time_{Sec}$ = pow(2, scale / 1200 * 0x00010000)
`s32 eg1Decay`	Scale for ADSR volume envelope decay time.
`s32 eg1Sustain`	Sustain volume for ADSR volume envelope, in decibels (0x00010000 = 0.1dB).
`s32 eg1Release`	Decibel change per 5 ms audio frame for ADSR volume envelope release stage (0x00010000 = 0.1dB).
`s32 eg1Vel2Attack`	Scale to add to ADSR volume envelope attack time. Scale = `eg1Vel2Attack` * (key velocity / 128).
`s32 eg1Key2Decay`	Scale to add to ADSR volume envelope decay time. Scale = `eg1Key2Decay` * (key number / 128).
`s32 eg2Attack`	Same as `eg1Attack` for ADSR pitch envelope.
`s32 eg2Decay`	Same as `eg1Decay` for ADSR pitch envelope.
`s32 eg2Sustain`	Pitch to sustain for ADSR pitch envelope (0x00010000 = 1cent).
`s32 eg2Release`	Cent change per 5-millisecond audio frame for ADSR pitch envelope (0x00010000 = 1cent).
`s32 eg2Vel2Attack`	Same as `eg1Vel3Attack` for ADSR pitch envelope.
`s32 eg2Key2Decay`	Same as `eg1Key2Decay` for ADSR pitch envelope.
`s32 eg2Pitch`	Relative pitch at max for ADSR pitch envelope (0x00010000 = 1cent).
`s32 pan`	Absolute panning for percussion instruments (0 = left, 64 = center, 127 = right).

4.2.2.5 Samples

Code 4-5 WTSAMPLE

```
typedef struct WTSAMPLE
{
    u16 format;     // ADPCM, PCM16, PCM8
    u16 sampleRate; // Hz
    u32 offset;     // offset in samples from beginning of PCM file
    u32 length;     // length of sample in samples
    u16 adpcmIndex; // ADPCM index to determine if in ADPCM mode

} WTSAMPLE;
```

The WT file contains an array of `WTSAMPLE` structures of an indefinite size. Each set of sampling data is referenced by one or more instrument region(s). Table 4-9 describes the data members:

Table 4-9 Wavetable Samples

Sample	Description
`u16 format`	Format for samples. Supports: ADPCM(WT_FORMAT_ASPCM), 16-bit PCM(WT_FORMAT_PCM16), and 8-bit PCM(WT_FORMAT_PCM8).
`u16 sampleRate`	Base sampling rate for specified sampling data measured (unit of hertz).
`u32 offset`	Offset to sampling data from beginning of PCM file (unit of sample).
`u32 length`	Sampling data length (unit of sample).
`u16 adpcmIndex`	Index for ADPCM data decode information array.

4.2.3 PCM File

The PCM file contains all the sampling data used by the WT file. All sampling data offsets, formats, loop points, size, and sample rates are stored in the WT file, which is loaded into ARAM by the user application. Afterwards, the synthesizer instance is then initialized with the ARAM offset in bytes.

4.3 Synthesizer Application Notes

4.3.1 MIDI Bank Support

The synthesizer application (SYN) ignores bank MSB and LSB for program changes. MSB and LSB are always assumed to be 0. Nonzero MSB and LSB bank instruments on are not supported by `dlsiwt.exe` or SYN.

4.3.2 MIDI Event Support

The synthesizer application (SYN) supports MIDI note off, note on, program change, pitch wheel, and some controller events. SYN does not support polyphonic after-touch, channel after-touch, and system real-time system events.

4.3.3 MIDI Controller Support

SYN will store all MIDI controller settings set by calling `SYNMidiInput()`. The controller settings in Table 4-10 are supported:

Table 4-10 MIDI Controller Settings

Controller Setting	Purpose
`0x01`	Modulation wheel.
`0x06 & 0x26`	Data entry (for pitch wheel range setting).
`0x07`	Channel volume.
`0x0A`	Pan.
`0x0B`	Expression.
`0x40`	Hold pedal.
`0x5B`	Effects, 1 depth (reverb).
`0x5C`	Effects, 2 depth (chorus).
`0x64 & 0x64`	Registered parameter number (for pitch wheel range).
`0x78`	All sound off.
`0x7B – 0x7F`	Only for all note off.

4.3.4 Calling SYNMidiInput()

User applications that call `SYNMidiInput()` should disable interrupts if called from outside the AX audio frame callback. Typically, a sequencer would run inside the AX audio frame callback and therefore would not need to disable interrupts. MIDI input events are buffered by the synthesizer and will execute during `SYNRunAudioFrame()`, thus ensuring that the ADSR envelope runs in proper sequence after a voice starts for a note.

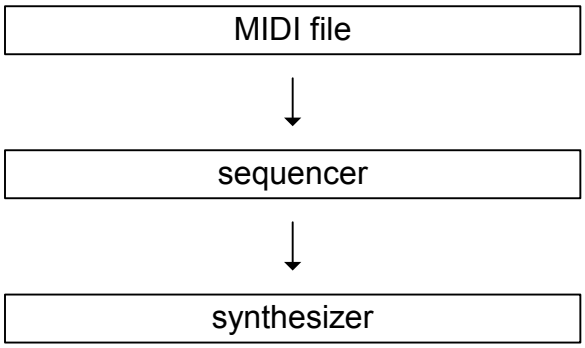
4.3.5 Shutting Down a Synthesizer

Prior to calling `SYNQuitSynth()`, the user application should call `SYNGetActiveNotes()` to ensure that no other notes are still running. In general, it takes some additional time for voices to complete the ADSR release phase after the MIDI note off event; during this time the note is still active.

5 MIDI Sequence

The MIDI sequence application (SEQ) takes standard MIDI type 0 and type 1 byte streams. It applies the MIDI events to the AX SYN application (described in the previous chapter).

Figure 5-12 MIDI File, Sequencer, and Synthesizer



For detailed API descriptions, refer to the **Sequence** topics under **AX Applications** in the **Audio** section of the Revolution Reference Manual (HTML).

5.1 Sequence Features

SEQ exposes logical controls to the user application for runtime manipulation of MIDI sequences.

5.1.1 State Control

The user can set one of the following states for any tracks in a MIDI sequence.

Table 5-11 Track State Controls

State	Description
Stop	Stops the sequence playback and rewinds each track to the beginning of the track.
Run	Starts the sequence playback from the current track position and stops at the end of the track.
Run looped	Automatically rewinds and continues playback from the end of the track.
Pause	Stops the sequence playback but does not rewind current track position.

5.1.2 Tempo Control

The user application can set the current tempo to and retrieve it from any track in a sequence. Tempo is expressed in floating point BPM (120.0 = 120 BPM).

5.1.3 Volume Control

The user application can set the current volume to and retrieve from any track in a sequence. Volume is expressed as a signed fixed point number in decibels (0x00010000 = 0.1 dB).

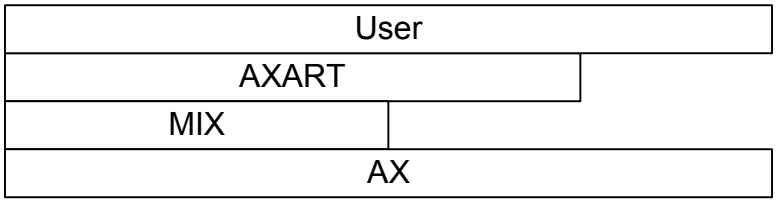
5.1.4 Controller Event Callback Interface

The user application may register callbacks for specific MIDI controller events. These events can then be inserted into a MIDI file to notify the user of playback progress.

6 Voice Articulator

The voice articulator library (AXART) applies articulations to AX voices in real-time. Any number of articulators can be applied to a single voice to achieve volume, pan, pitch, LFO, envelope, and 3D sound effects. Articulators applied to a voice can be interactively controlled to alter the associated sound as it plays. AXART relies on the AX and MIX libraries.

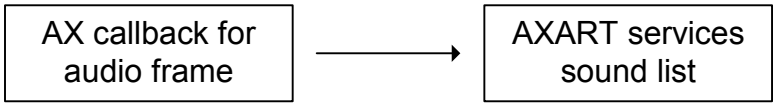
Figure 6-13 AXART API Layers



6.1 AXART and AX

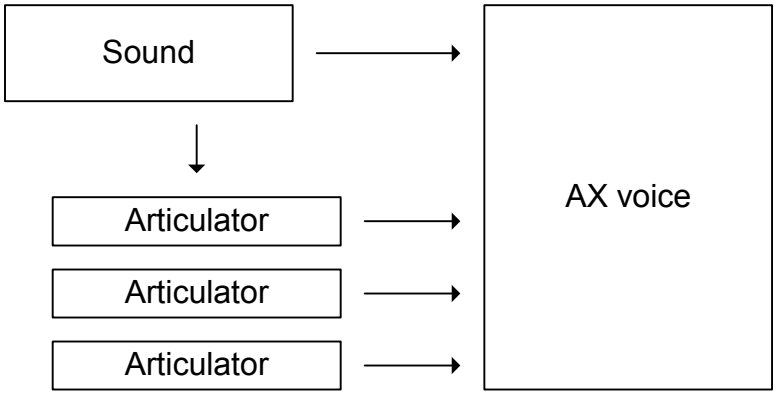
AXART uses a sound list that utilizes the AX audio frame callback.

Figure 6-14 AX Callback and AXART



Each sound added to the list has a pointer to its associated AX voice and a list of articulators.

Figure 6-15 Sound, Articulators and AX Voice



The articulators for each sound have specific functions and will alter pitch, pan, volume and ITD settings as specified by the user. The user can change articulator parameters at runtime.

6.2 Articulator Types

Table 6-12 shows the articulators and their voice parameter usage methods. Refer to the Revolution Reference Manual for details about each articulator.

Table 6-12 Articulators and Voice Parameter Usage

Articulator Type	SRC Type	SRC Ratio	ITD	Volume	Pan	Span	AuxA	AuxB	LFO
AXART_3D	X	X	X	X	X	X			
AXART_PANNING					X	X			
AXART_ITD			X						
AXART_SRCTYPE	X								
AXART_PITCH		X							
AXART_PITCH_ENV		X							
AXART_PITCH_MOD		X							X
AXART_VOLUME				X					
AXART_AUXA_VOLUME							X		
AXART_AUXB_VOLUME								X	
AXART_VOLUME_ENV				X					
AXART_AUXA_VOLUME_ENV							X		
AXART_AUXB_VOLUME_ENV								X	
AXART_VOLUME_MOD				X					X
AXART_AUXA_VOLUME_MOD							X		X
AXART_AUXB_VOLUME_MOD								X	X

6.3 LFOs

AXART provides several common LFO shapes:

- Sine
- Square
- Saw
- Reverse Saw
- Triangle
- Noise

The user may implement additional shapes. See the Revolution Reference Manual for details about LFOs.