Revolution AX Applications

Version 1.02

The contents in this document are highly confidential and should be handled accordingly.

RVL-06-0032-001-F
Released: October 17, 2008

Contents

Code Examples

Equations

Figures

Tables

Revision History

Version	Revision Date	Item	Description
1.02	2007/09/28	-	Complete revision to Chapter 3 (AUX Effects).
1.01	2006/12/26	-	Complete revision to match the Wii specification.
1.00	2006/03/01	-	First release by Nintendo of America Inc.

1 Overview

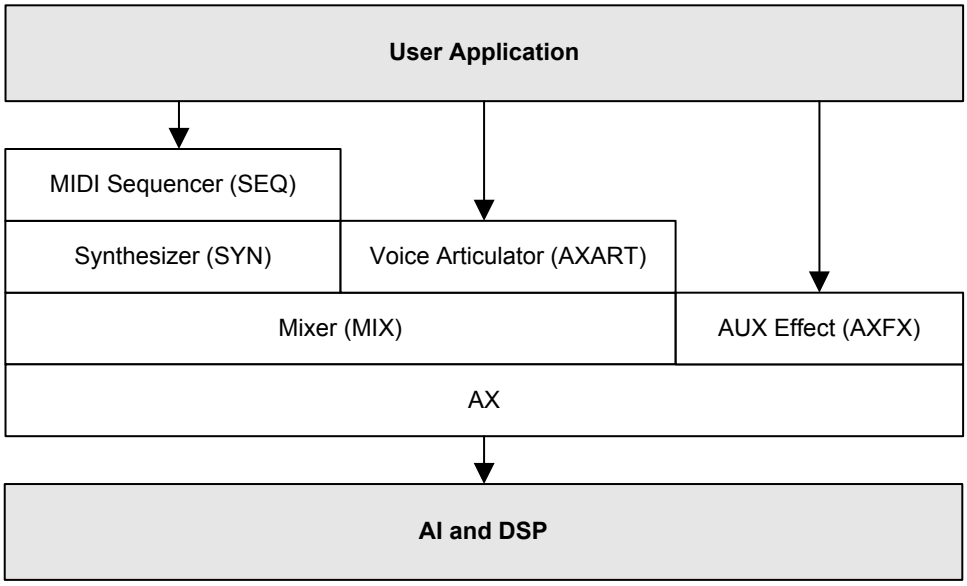
The AX library provides a low-level abstraction for the Revolution audio subsystem shown in Figure 1-1. The AX library was designed to provide flexible support for audio applications (for example, MIDI synthesizers, mixers, software audio streaming, and sound effect synthesizers).

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

- MIDI sequencer (SEQ)
- Synthesizer (SYN)
- Voice articulator (AXART)
- Mixer (MIX)
- AUX effects (AXFX)

This document provides an introduction to these libraries.

Figure 1-1 Applications for AX



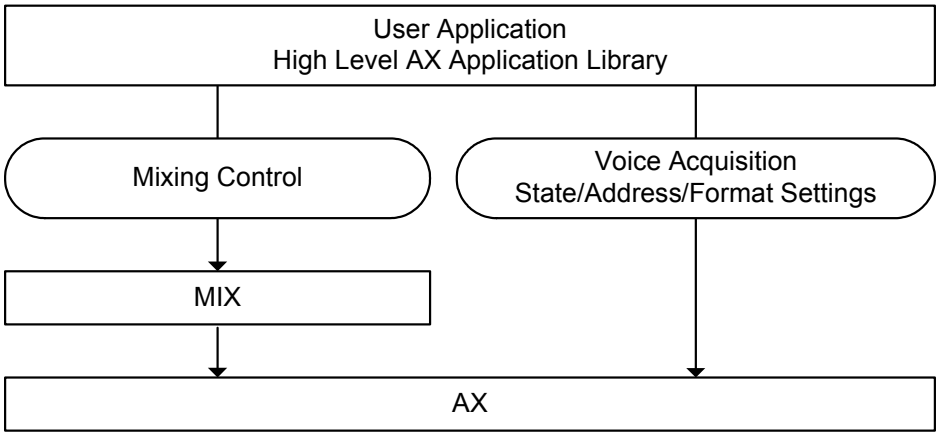
For detailed AX component descriptions, see “AX Applications” in the “Audio System” section of the *Revolution Function Reference Manual*.

2 Mixer

The AX mixer library (MIX) is located in the layer immediately above AX and controls the AX voice based on requests from the higher level AX application libraries and user applications. Parameters set in the MIX library (mixing control) are: input level, AUX send level, panning, muting, and fader. MIX library also performs an automatic volume ramping to prevent zip and pop effects when applying volume and pan changes to AX voice.

The MIX library also supports the remote controller speakers, where the fader and AUX send level may be set for each Wii Remote.

Figure 2-1 MIX API Layer



Applications will configure the individual mixing controls through the MIX API. The newly configured mixing control will be reflected into the AX voice when the audio frame callback is called.

For detailed API descriptions, see “Mixer” in the “Audio System” section of AX Applications in the *Revolution Function Reference Manual*.

2.1 Mixer Modes

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

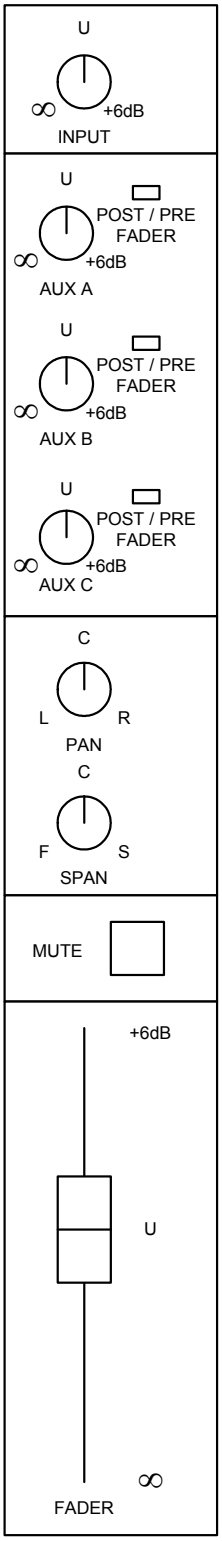
Table 2-1 Mixer Modes

MIX mode	Description	AX Mode
MIX_SOUND_MODE_MONO	Mono output mode. AX will be processed in the stereo mode, but the MIX library will ignore the left and right pan. This will result in AX output data that is mono (equivalent).	AX_MODE_STEREO
MIX_SOUND_MODE_STEREO	Stereo output mode. AX will also be processed in the stereo mode, and neither MIX nor AX will ignore the surround pan. For stereo applications, AX library mixes the generated surround sound signal equally to the left and right channels.	AX_MODE_STEREO
MIX_SOUND_MODE_SURROUND	Surround sound output mode. AX library will encode the surround sound signal to left and right channels.	AX_MODE_SURROUND
MIX_SOUND_MODE_DPL2	Dolby Pro Logic II output mode. This mode does not support Aux C.	AX_MODE_DPL2

2.2 Mixer Controls

2.2.1 Voice Mixing Controls

Figure 2-2 Mixer Channels



AUX supports post-fader mode and pre-fader modes.

The left to right pan is 0 ~ 127 (center is 64).

The front to back pan is 0 ~ 127 (center is 64).

Mute switch

0.1 dB unit fader

The MIX library supports AX_MAX_VOICES number of mixer channels in Figure 2-2.

In the MIX library, the volume parameters, such as input level and AUX send level, are managed in 0.1dB increments. For example, to set an AUX send level of -90.4dB, the MIX API will be specified as -904. Also, the individual volume parameters will be clamped to +6.0dB - -90.4dB at integration to AX voice.

2.2.1.1 Input Level (input)

The input control manages the input level applied to the input data (ADPCM/PCM sample). Applications can use this control to apply ADSR envelope and tremolo in addition to volume adjustments. 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 Aux*N* control manages the AUX send level for each bus. This control can also switch between the postfader and prefader modes. The following AUX send level is applied based on mode:

Equation 2-1 AUX Send Level

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

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

This control is set to post-fader mode/-96.0dB by default.

2.2.1.3 Pan (pan)

The pan control manages the left and right pan. Left is 0, center is 64, and right is 127. The pan value is converted as follows and reflected to the volume:

Equation 2-2 Pan Conversion

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

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

This control is set to 64 by default.

2.2.1.4 Surround Pan (span)

The span control manages the front and rear pan. Rear is 0, center is 64, and front is 127. The span value is converted as follows and reflected to the volume:

Equation 2-3 Span Conversion

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

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

This control is set to 127 by default.

2.2.1.5 Mute (mute)

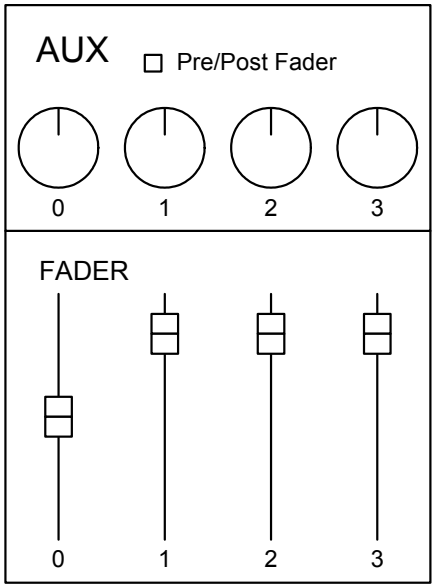
When the mute control is turned on, the input level will become 0 (-90.4dB). The input level will return to input control value when the mute control is turned off. The default value for this control is “off.”

2.2.1.6 Fader (fader)

The fader control manages the output level of all channels in the main bus (right, left, surround). This setting will also be applied to all AUX bus channels when the AUX bus is set to postfader mode. The default value is 0 dB for this control.

2.2.2 Wii Remote Speaker Control

Figure 2-3 Wii Remote Speaker Control



The MIX library supports Wii Remote speaker control shown in Figure 2-3 for each mixer channel.

2.2.2.1 Fader

The fader control manages the speaker output level for each Wii Remote.

2.2.2.2 AUX Send

The AUX control manages the AUX send level for each Wii Remote. This control also allows switching of a common pre-fader/post-fader mode for all Wii Remotes.

Note: As of 2006/12/26, the AUX bus processing for Wii Remote speakers is not yet implemented.

2.3 Mixer Application Notes

2.3.1 Volume Clamping

The MIX library clamps the individual volume between +6.0 dB and -90.4 dB and passes it to the AX library to accommodate the mixer specification of the DSP. Since the clamping will be performed automatically, the application does not need to consider this range when adjusting the fader and input controls. However, a careless adjustment of the individual controllers may cause the controller value to wrap around.

2.3.2 Volume Ramping

The mixer applies “per sample volume ramping” (process to gradually change volume) to mitigate pop or zip effects induced by rapid changes in volume. This volume ramping will be applied when the change in volume is greater than or equal to 96. If the volume change is smaller than 96, the sound volume will be set as-is. The new volume level set by the user will be ramped over a single audio frame and applied from the subsequent audio frame.

2.3.3 Suggested Input and Fader Control Use

When adjusting the volume, fewer DSP cycles are consumed when the adjustment is made through the input control instead of the fader control. Programmers can utilize the DSP efficiently by balancing the input control and fader control use per application.

The following sections discuss a few examples for using these controls.

2.3.3.1 Sound Effects and Audio Streaming

For most sound effects and audio streaming, it is possible to fix the fader control to 0 dB and use the input control to adjust the fader volume.

2.3.3.2 Music Synthesizer

With the music synthesizer, it is possible to fix the fader control to 0 dB and implement ADSR envelope, LFO volume modulation, channel volume, expression pedal volume, and master volume using only the input control.

2.3.3.3 3D Sound Applications

In 3D sound applications, movement of a sound source can be expressed through fine timing control of the pan control and span control. The distance attenuation of the volume can also be expressed through manipulation of the input control.

3 AUX Effects

The AUX Effects library (AXFX) provides various effects for the AuxA, AuxB, and AuxC buses in AX. Wii has added expansion effects that enable the implementation of effects that are more flexible and diverse, in addition to existing effects inherited from Nintendo GameCube.

Note: The implementation of all existing effects (excluding delay) has been changed to use expansion effects, while preserving API compatibility.

These existing and expansion effects include high-quality reverb, standard reverb, chorus, and delay.

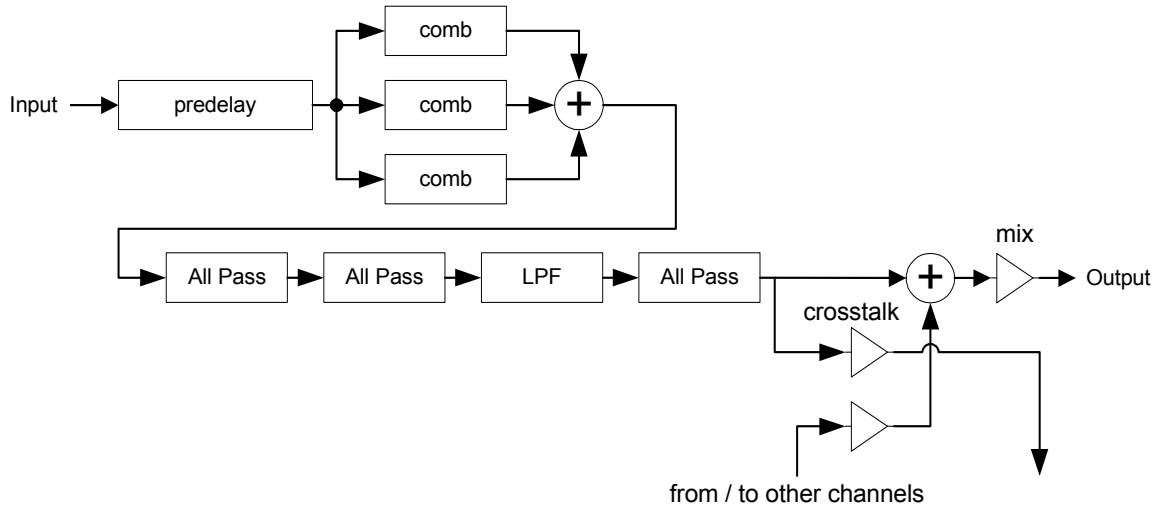
All effects have left, right, and Surround (left-front, right-front, left-rear, and right-rear for Dolby Pro Logic II) channels, all processed independently.

3.1 Effects

3.1.1 High Quality Reverb

The structure of the high-quality reverb is shown in Figure 3-1.

Figure 3-1 High-Quality Reverb



High-quality reverb takes the following parameters.

Table 3-1 High Quality Reverb Controls

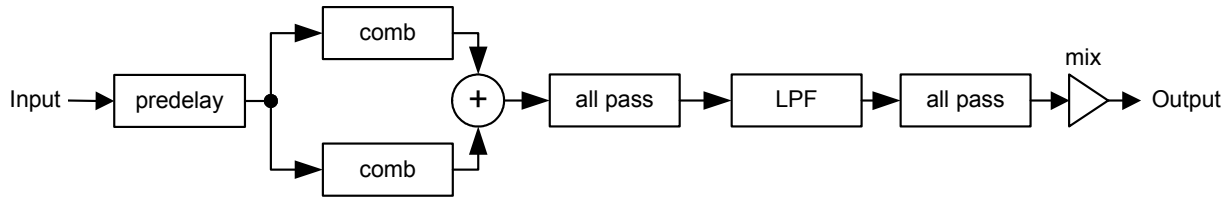
Parameter	Description
preDelay	The time delay before reverberation starts. A value of 0.0 or larger (in seconds) is specified. Set a large value for preDelay for a larger distance between the sound source and the reflective wall when simulating a large room.
time	The length of time until the reverberation attenuates. A value of 0.0 or larger (in seconds) is specified. A value of 0.01 seconds simulates a very small room, while a value of 10.0 seconds will simulate a cathedral or a stadium.
coloration	Modulates the all-pass filter coefficients. A value between 0.0 and 1.0 is specified. This value is used to simulate the acoustic properties of the reflective walls. In general, the smaller this value, the rougher the density of reverberations. The larger the value, the more detailed the density, but waveform interference may result in times where high-frequency waves sound louder than they should.
damping	Modulates high-frequency attenuation for reverberation by adjusting the low-pass filter (LPF) coefficients. A value between 0.0 and 1.0 is specified. The closer the value is to 0.0, the more pronounced low frequency components will be. Conversely, as damping approaches 1.0, high frequency components will remain without attenuation.
crosstalk	Sets the degree of cross-channel interaction. A value between 0.0 and 1.0 is specified. When a crosstalk value of 0.0 is specified, all interaction between the channels will be eliminated. When a value larger than 0.0 is specified, the reverberated signal will be applied to other channels as well.
mix	Output gain for the reverberation. A value between 0.0 and 1.0 is specified.

Note: In contrast with the Nintendo GameCube AXFX library, reverb here only outputs the reverberation, and does not output the original sound.

3.1.2 Standard Reverb

The structure of the standard reverb is shown in Figure 3-2. The structure is simplified compared to high-quality reverberation, reducing the CPU load, but resulting in rougher reverberations. There are also fewer parameters to specify.

Figure 3-2 Standard Reverb



Standard reverb takes the following parameters:

Table 3-2 Standard Reverb Controls

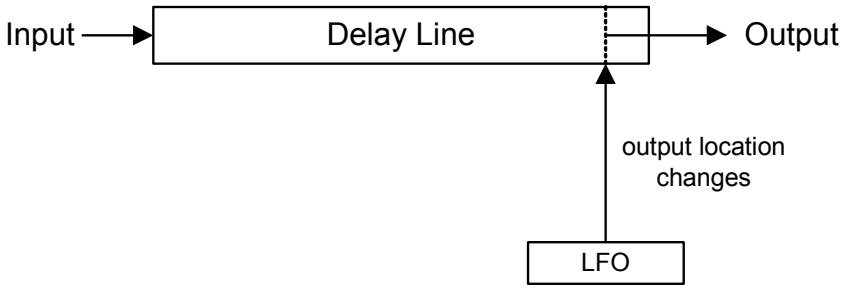
Standard Reverb Control	Description
preDelay	The time delay before reverberation starts. A value of 0.0 or larger (in seconds) is specified. Set a large value for preDelay for a larger distance between the sound source and the reflective wall when simulating a large room.
time	The length of time until the reverberation attenuates. A value of 0.0 or larger (in seconds) is specified. A value of 0.01 seconds simulates a very small room, while a value of 10.0 seconds will simulate a cathedral or a stadium.
coloration	Modulates the all-pass filter coefficients. A value between 0.0 and 1.0 is specified. This value is used to simulate the acoustic properties of the reflective walls. In general, the smaller this value, the rougher the density of reverberations. The larger the value, the more detailed the density, but waveform interference may result in times where high-frequency waves sound louder than they should.
damping	Modulates high-frequency attenuation for reverberation by adjusting the low-pass filter (LPF) coefficients. A value between 0.0 and 1.0 is specified. The closer the value is to 0.0, the more pronounced low frequency components will be. Conversely, as damping approaches 1.0, high frequency components will remain without attenuation.
mix	Output gain for the reverberation. A value between 0.0 and 1.0 is specified.

Note: In contrast with the Nintendo GameCube AXFX library, reverb here only outputs the reverberation and does not output the original sound.

3.1.3 Chorus

Chorus refers to the creation of a delay sound periodically modulated by a delay time. Its structure is shown in Figure 3-3.

Figure 3-3 Chorus



Chorus takes the following parameters:

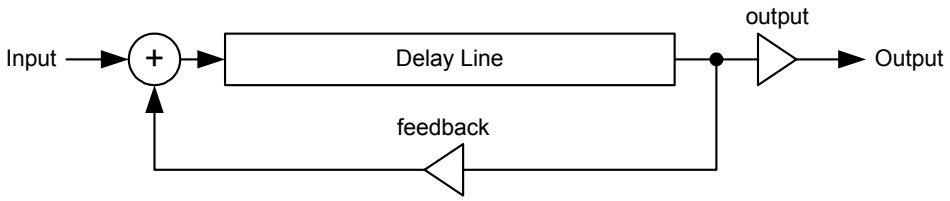
Table 3-3 Chorus Controls

Chorus Control	Description
baseDelay	The length of the delay line (input signal base delay). A value between 1 and 50 (milliseconds) is specified.
variation	The time variation range of the output position from the delay line (that is, the delay time) through LFO. A value between 0 and baseDelay (milliseconds) is specified. The output position changes time between the values (baseDelay - variation) and (baseDelay + variation). **Note:** This parameter is sometimes referred to as **chorus depth** in other documents.
period	The LFO variation cycle. A value between 500 and 10000 (milliseconds) is specified. **Note:** This parameter is typically referred to as **chorus rate** or **chorus speed** in other documents.

3.1.4 Delay

The structure of the delay is as shown in Figure 3-4.

Figure 3-4 Delay



Delay takes the following parameters.

Table 3-4 Delay Controls

Delay Control	Description
delay	The length of the delay line (that is, the input signal delay time). A value larger than 0 (milliseconds) is specified.
feedback	The feedback gain. A value smaller than 100 (percent) is specified.
output	The output gain. A value smaller than or equal to 100 (percent) is specified.

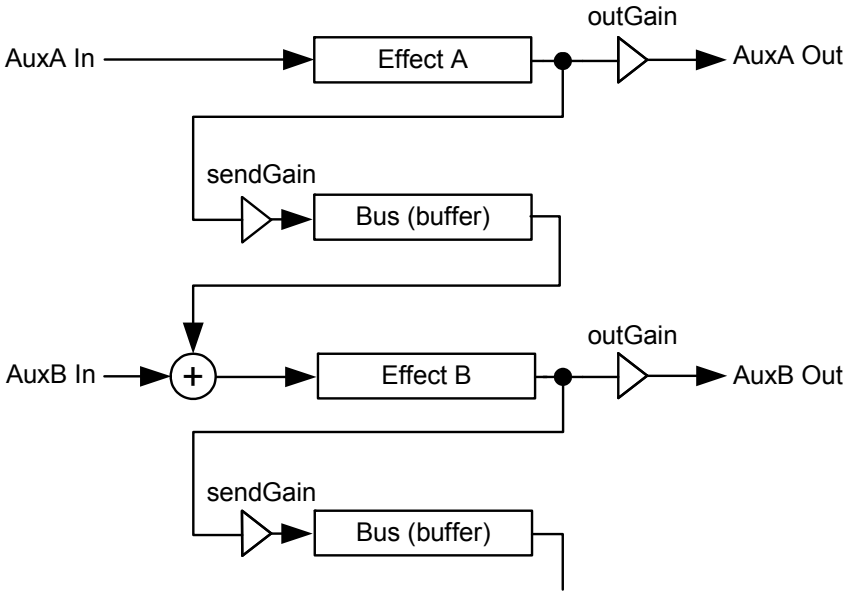
Note: The individual parameters must be configured for all channels, left, right, and Surround, in that order (and in the order of left-front, right-front, left-rear, and right-rear for Dolby Pro Logic II).

3.2 Expansion Effects

3.2.1 Effect Bus Send

Two types of parameters, busIn and busOut, are available for expansion effects. Their use enables processes such as applying reverberation on a chorus effect.

Figure 3-5 Effect Bus Send



As shown in Figure 3-5, each of the expansion effects, in addition to the existing output (Out) pass, now has a pass for sending output to the next stage's effect (Send).

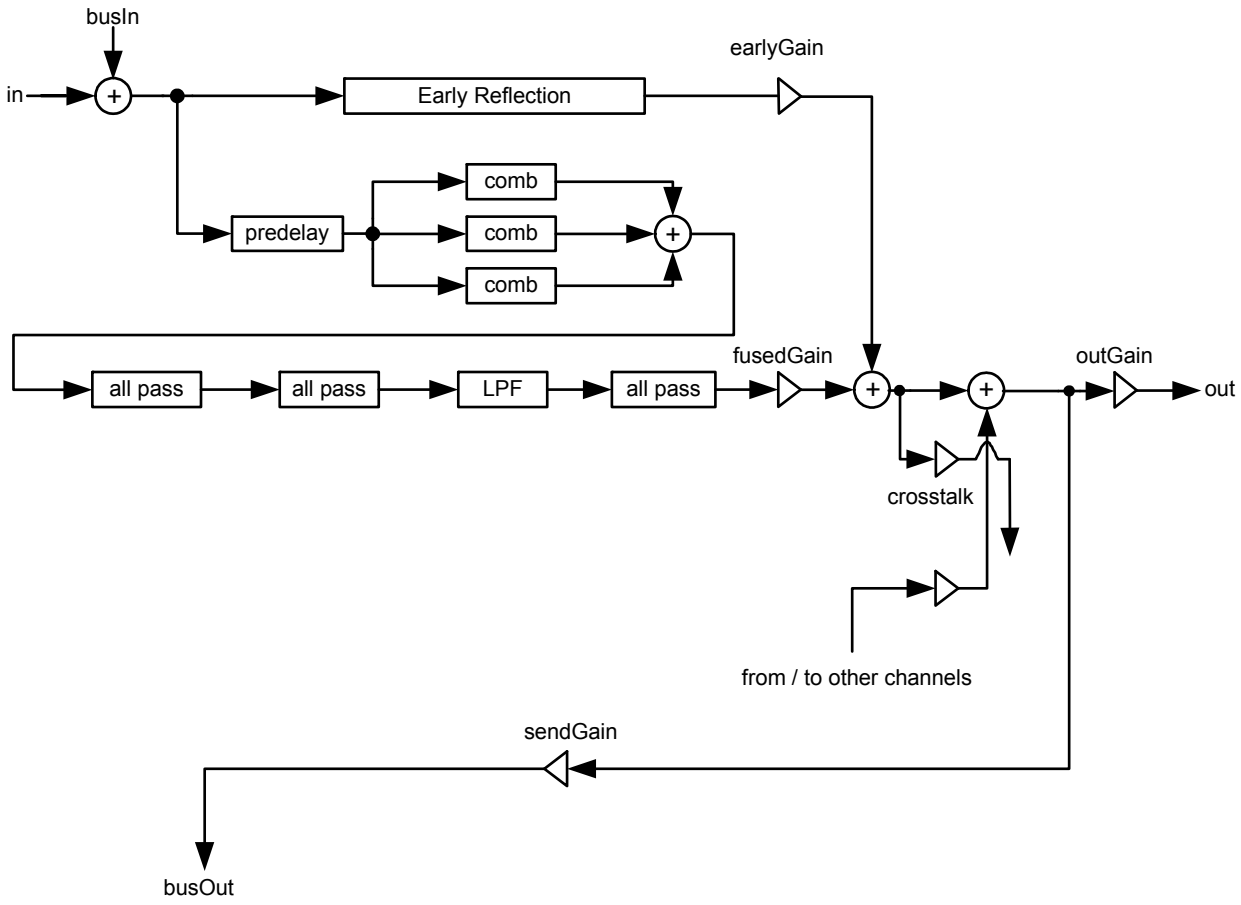
Note: Effect processing occurs in order from AuxA to AuxB to AuxC. As a result, when returning AuxC output to AuxA, the AuxC data is reflected in AuxA with a lag of one audio frame.

The Send target bus is a buffer, and memory must be allocated for it in advance. Buffers are required for each channel, and their size in bytes can be calculated as a product of sizeof(s32) and AX_IN_SAMPLES_PER_FRAME.

3.2.2 Expansion Effect High-quality Reverb

The structure of the expansion effect high-quality reverb is as shown in Figure 3-6.

Figure 3-6 Expansion Effect High-Quality Reverb



The expansion effect high-quality reverb takes the following parameters.

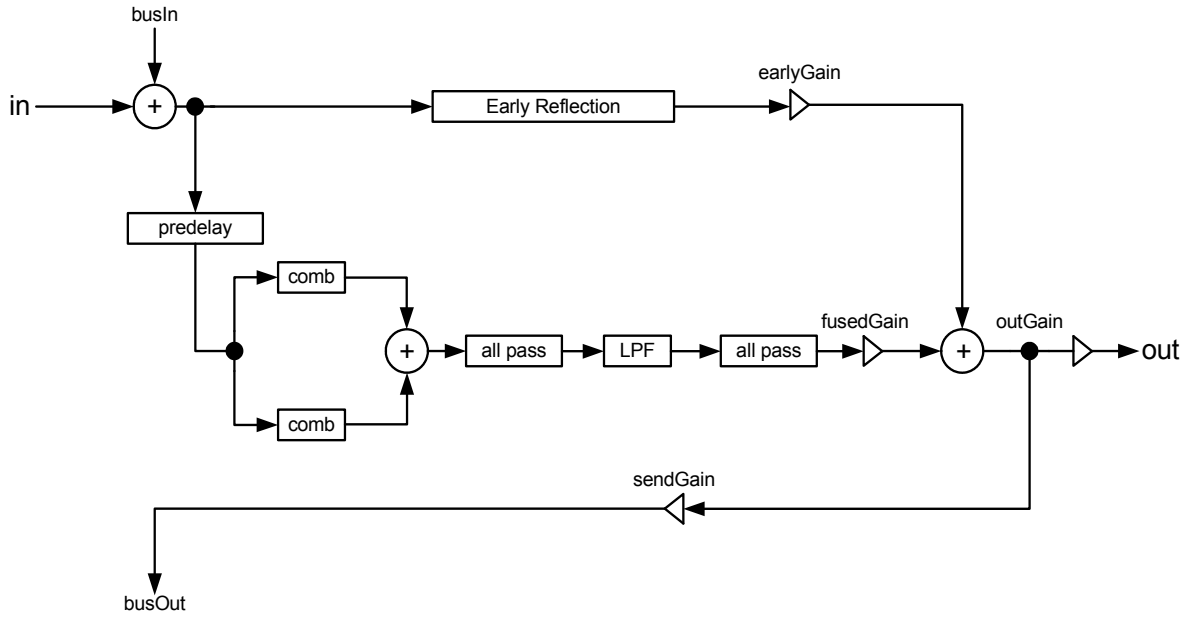
Table 3-5 Expansion Effect High-Quality Reverb Controls

Parameter	Description
earlyMode	The mode for the initial reflection (Early Reflection).
earlyGain	The mix gain for the initial reflection. A value ranging from 0.0 to 1.0 is specified.
preDelayTimeMax	The maximum value of the pre-delay for the end of the reverberation (that is, the maximum delay until reverberation begins). A value of 0.0 or more (in seconds) is specified. The pre-delay memory must be allocated according to this value.
preDelayTime	The pre-delay for the end of the reverberation (that is, the delay until the start of the reverberation). A value ranging from 0.0 to preDelayTimeMax (in seconds) is specified.
fusedMode	The mode for the end of the reverberation.
fusedTime	The reverberation time for the end of the reverberation. A value of 0.0 or more (in seconds) is specified.
coloration	Modulates the all-pass filter coefficients. A value between 0.0 and 1.0 is specified. This value is used to simulate the acoustic properties of the reflective walls. In general, the smaller this value, the rougher the density of reverberations. The larger the value, the more detailed the density, but waveform interference may result in times where high-frequency waves sound louder than they should.
damping	Modulates high-frequency attenuation for reverberation by adjusting the low-pass filter (LPF) coefficients. A value between 0.0 and 1.0 is specified. The closer the value is to 0.0, the more pronounced low frequency components will be. Conversely, as damping approaches 1.0, high frequency components will remain without attenuation.
fusedGain	The mix gain for the end of the reverberation. A value between 0.0 and 1.0 is specified.
crosstalk	Sets the degree of cross-channel interaction. A value between 0.0 and 1.0 is specified. When a crosstalk value of 0.0 is specified, all interaction between the channels will be eliminated. When a value larger than 0.0 is specified, the reverberated signal will be applied to other channels as well.
outGain	The output gain. A value between 0.0 and 1.0 is specified.
The following are effect bus send parameters.	
busIn	Specifies the input bus (buffer).
busOut	Specifies the output bus (buffer).
sendGain	The send gain. A value between 0.0 and 1.0 is specified.

3.2.3 Expansion Effect Standard Reverb

The structure of the expansion effect standard reverb is as shown in Figure 3-7.

Figure 3-7 Expansion Effect Standard Reverb



The expansion effect standard reverb takes the following parameters.

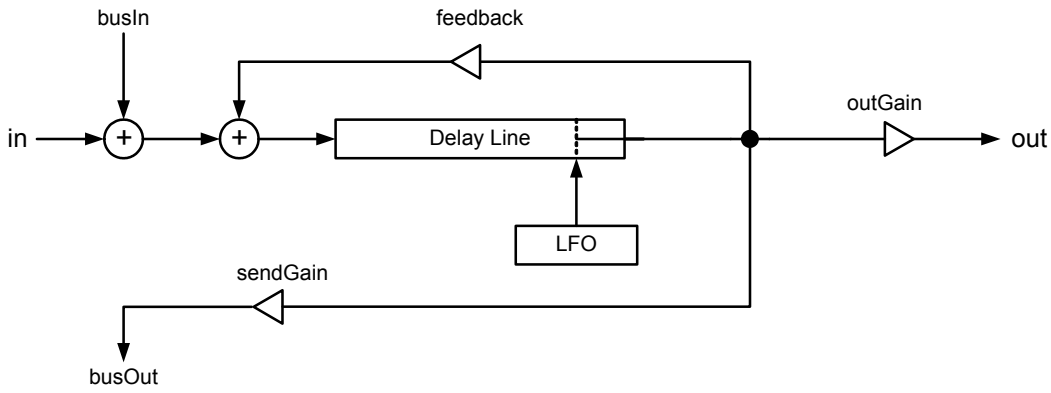
Table 3-6 Expansion Effect Standard Reverb Controls

Parameter	Description
earlyMode	The mode for the initial reflection (early reflection).
earlyGain	The mix gain for the initial reflection. A value ranging from 0.0 to 1.0 is specified.
preDelayTimeMax	The maximum value of the pre-delay for the end of the reverberation (that is, the maximum delay until reverberation begins). A value of 0.0 or more (in seconds) is specified. The pre-delay memory must be allocated in conjunction with this value.
preDelayTime	The pre-delay for the end of the reverberation (that is, the delay until the start of the reverberation). A value ranging from 0.0 to preDelayTimeMax (in seconds) is specified.
fusedMode	The mode for the end of the reverberation.
fusedTime	The reverberation time for the end of the reverberation. A value of 0.0 or more (in seconds) is specified.
coloration	Modulates the all-pass filter coefficients. A value between 0.0 and 1.0 is specified. This value is used to simulate the acoustic properties of reflective walls. In general, the smaller this value, the rougher the density of reverberations. The larger the value, the more detailed the density, but waveform interference may result in times where high-frequency waves sound louder than they should.
damping	Modulates high-frequency attenuation for reverberation by adjusting the low-pass filter (LPF) coefficients. A value between 0.0 and 1.0 is specified. The closer the value is to 0.0, the more pronounced low frequency components will be. Conversely, as damping approaches 1.0, high frequency components will remain without attenuation.
fusedGain	The mix gain for the end of the reverberation. A value between 0.0 and 1.0 is specified.
outGain	The output gain. A value between 0.0 and 1.0 is specified.
The following are effect bus send parameters.	
busIn	Specifies the input bus (buffer).
busOut	Specifies the output bus (buffer).
sendGain	The send gain. A value between 0.0 and 1.0 is specified.

3.2.4 Expansion Effect Chorus

The structure of the expansion effect chorus is as shown in Figure 3-8.

Figure 3-8 Expansion Effect Chorus



The expansion effect chorus takes the following parameters.

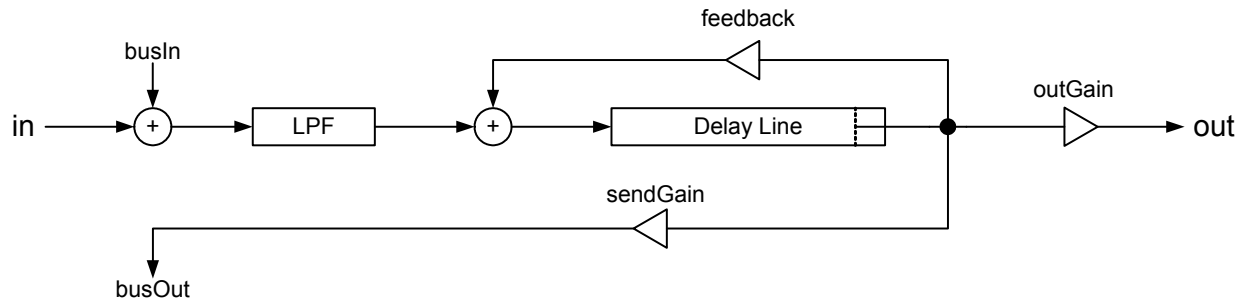
Table 3-7 Expansion Effect Chorus Controls

Parameter	Description
delayTime	The length of the delay line (that is, the input signal's basic delay time). A value between 0.1 and 50.0 (in milliseconds) is specified.
depth	The degree of variation by which LFO modifies the output position from the delay line (that is, the delay time). Specified as a ratio of delayTime, with a value ranging from 0.0 to 1.0.
rate	The LFO frequency. A value between 0.1 and 2.0 (in Hertz) is specified.
feedback	The feedback gain. A value between 0.0 and 1.0 is specified. This is used primarily to increase the flange effect when using the chorus as a flanger effect (to produce a powerful swell of sound, called a jet sound). It can also be used for unique delay effects when a large delayTime value is specified.
outGain	The output gain. A value between 0.0 and 1.0 is specified.
The following are effect bus send parameters.	
busIn	Specifies the input bus (buffer).
busOut	Specifies the output bus (buffer).
sendGain	The send gain. A value between 0.0 and 1.0 is specified.

3.2.5 Expansion Effect Delay

The structure of the expansion effect delay is as shown in Figure 3-9.

Figure 3-9 Expansion Effect Delay



The expansion effect delay takes the following parameters.

Table 3-8 Effect Delay Controls

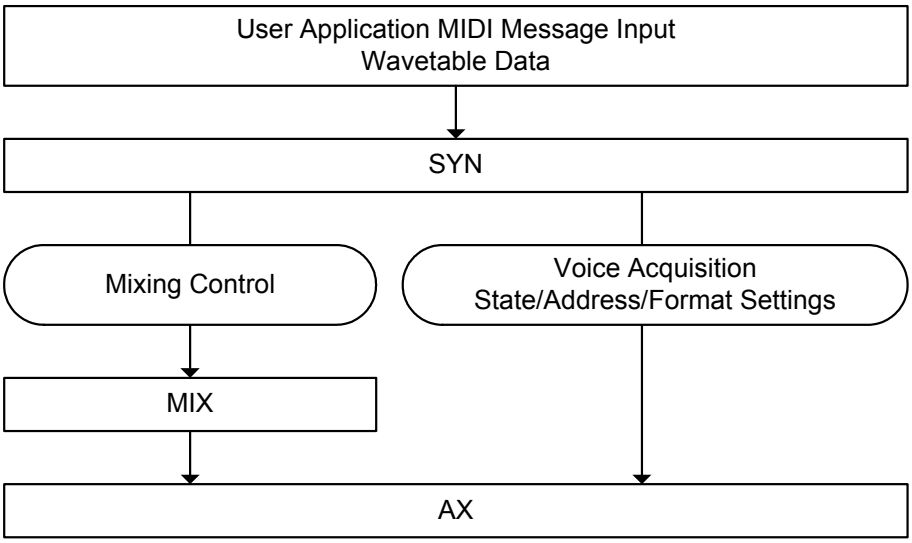
Parameter	Description
maxDelay	The length of the delay line (that is, the input signal's basic delay time). A value larger than 0.0 (in milliseconds) is specified. Memory for the delay line must be allocated according to this value.
delay	The delay time (that is, the delay time for the input signal). A value from 0.0 to maxDelay (in milliseconds) is specified.
feedback	The feedback gain. A value between 0.0 and 1.0 is specified.
lpf	Modulates the cutoff frequency for the low-pass filter (LPF). A value from 0.0 to 1.0 is specified. The smaller the value, the lower the cutoff frequency.
outGain	The output gain. A value between 0.0 and 1.0 is specified.
The following are effect bus send parameters.	
busIn	Specifies the input bus (buffer).
busOut	Specifies the output bus (buffer).
sendGain	The send gain. A value between 0.0 and 1.0 is specified.

Note: In existing versions of delay, parameters had to be specified for each channel, but this has been changed in the expansion effect delay effect such that the parameters are shared by all channels.

4 Synthesizer

The synthesizer library (SYN) accepts the MIDI message input by the higher level AX application library and user application, and the wave table information registered to synthesizer. The library then uses this information to control the mixer and AX voice, producing music.

Figure 4-1 SYN API Layer

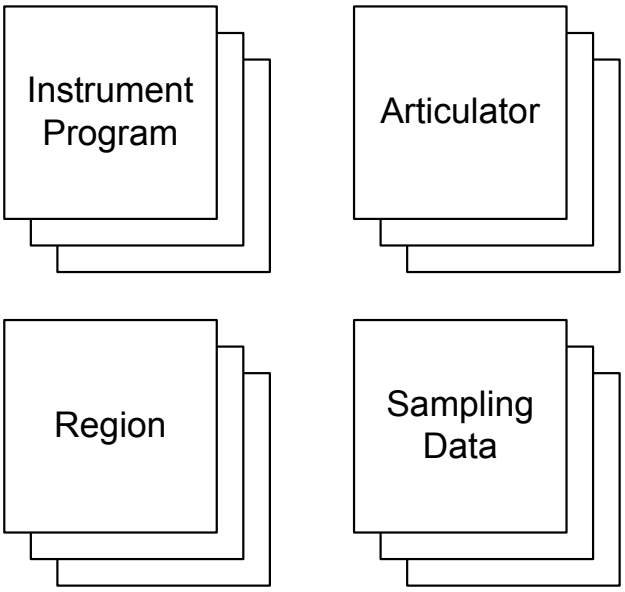


For detailed API descriptions, see “Synthesizer” under “AX Applications” in the “Audio System” section of the *Revolution Function Reference Manual*.

4.1 Wavetable

A wavetable is a lookup table used by the synthesizer to determine playback parameters for individual notes. Specifically, the wavetable contains tables for instrument programs, regions, articulators, and sampling data. The following sections explain general wavetable concepts; implementation for this specific synthesizer is covered in section 4.3 Synthesizer Application Notes.

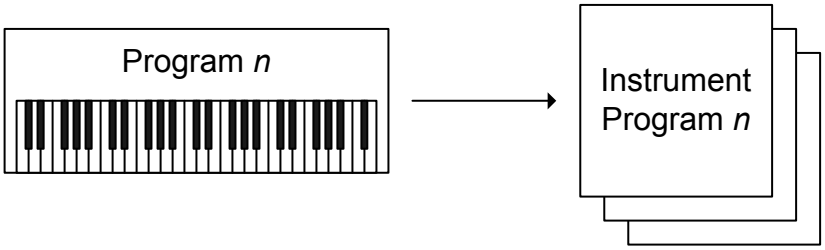
Figure 4-2 Wavetable Concepts



4.1.1 Instrument Programs

When a MIDI program number is allocated to a specific channel through MIDI messages, an instrument program corresponding with the program number will be referenced. Various wave table objects will be associated with the individual instrument programs and referenced at the time of synthesis. Instrument programs come in two types: melodic and percussion.

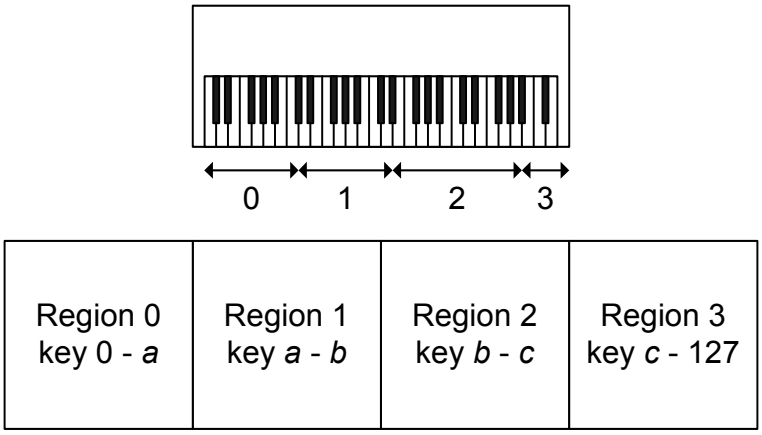
Figure 4-3 Instrument Programs



4.1.1.1 Melodic Instruments

As shown in Figure 4-4, individual melodic instruments typically do not have a region for each note/on-message specified key number, but instead are composed of *n* key regions that can each assume a range of keys. Each region will contain descriptors representing the lowest and the highest keys to specify the key range. Regions will also contain pointers to articulators and sampling data. When the MIDI message specifies a program number and a key number, the synthesizer will produce music using the appropriate sampling data after selecting the corresponding instruments and region.

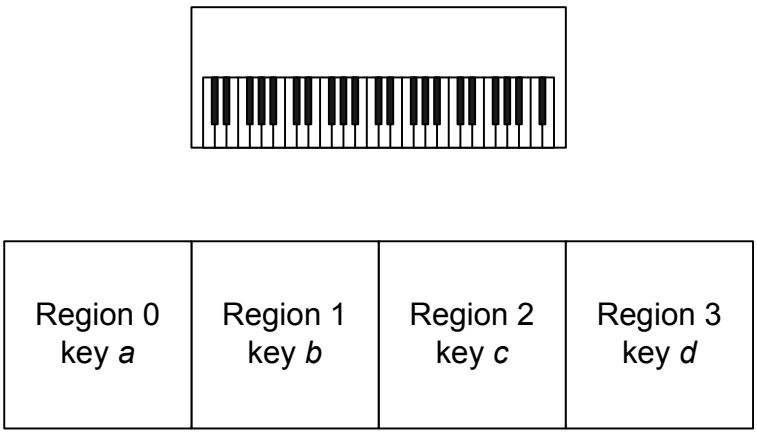
Figure 4-4 Melodic Key Regions



4.1.1.2 Percussive Instruments

Percussion instruments typically play a unique sample for each key. Unlike melodic instruments, percussion instruments contain a region for each key as shown in Figure 4-5. Each region will also contain a “key group” data, in addition to pointers to articulators and sampling data, that is also contained in melodic instrument regions. Key groups are used for exclusive playback of samples. For example, more than one closed, open, or pedal hi-hat key cannot be played at the same time. To achieve this, the same key group is assigned for these data.

Figure 4-5 Percussive Key Regions

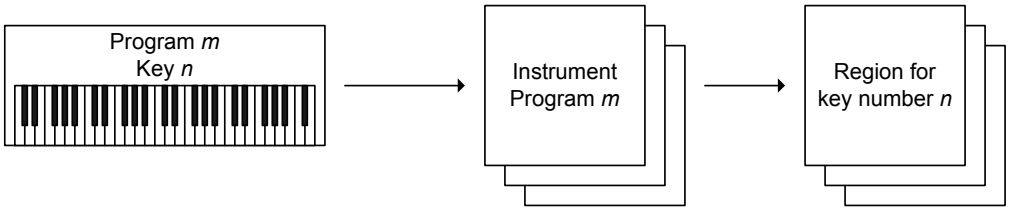


4.1.2 Regions

When the MIDI message specifies a program number and a key number, the synthesizer will produce sounds using the corresponding region information. A region will have the following parameters:

- **Low key:** Lowest key of the region.
- **High key:** Highest key of the region.
- **Normal key:** Baseline key of the region. This is also known as a **unity note**.
- **Key group:** Key group.
- **Fine tune:** Fine tuning value for the pitch.
- **Attenuation:** Volume adjustment value.
- Pointer to the articulator.
- Pointer to the sample data.

Figure 4-6 Region Specification

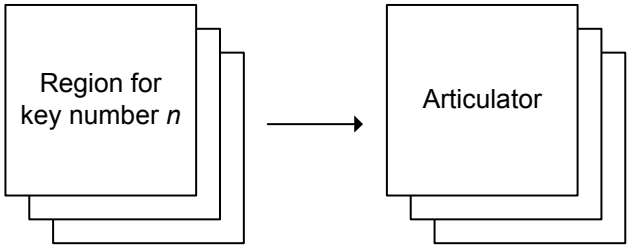


4.1.3 Articulators

Articulators define elements that add characteristics to the music during music production. Typically, an articulator will have the following parameters:

- LFO parameter (start delay, frequency, pitch modulation range, and volume modulation range)
- Volume envelope parameter (attack time, decay time, sustain volume, and so on)
- Pitch envelope parameter (attack time, decay time, sustain pitch, and so on)
- Pan for percussion instruments

Figure 4-7 Regions and Articulators

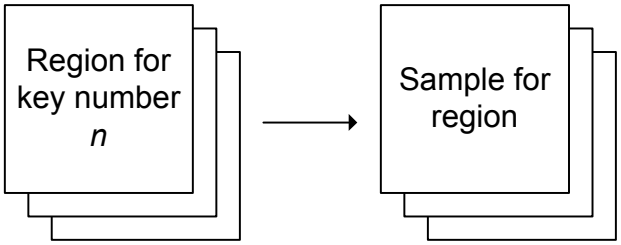


4.1.4 Sampling Data

Sampling data will typically have the following parameters:

- Sampling frequency
- Format
- Data size
- Offset address
- Loop information (loop start and loop length)

Figure 4-8 Regions and Sampling Data



4.2 File Format

This section will explain the file format of the wave table data that is used by the SYN library. Wave table data is composed of a WT and PCM file. These files are created from a DLS 1.0 file.

4.2.1 File Creation

Wave table data used by the SYN library is created using the `dls1wt.exe` program that is included in the SDK. `dls1wt.exe` accepts DLS 1.0 files as an input. These DLS 1.0 files can be created through commercially available DLS editors. `dls1wt.exe` outputs a WT file containing an instrument program with sample data information and a PCM file. The PCM file is a collection of only the sampling data.

4.2.2 WT File

The WT file is a lookup table, optimized so that the SYN library will access the runtime. The format for the WT file is defined in `/include/revolution/wt.h`. The WT file format is explained in this section.

4.2.2.1 File Header

Code 4–1 WT File Header

```
typedef struct WTFILEHEADER
{

    u32 offsetPercussiveInst;
    u32 offsetMelodicInst;
    u32 offsetRegions;
    u32 offsetArticulations;
    u32 offsetSamples;
    u32 offsetAdpcmContext;

    // data ...

} WTFILEHEADER;
```

Code 4–1 shows the file header of the WT file. An offset for each data structure is stored here.

Table 4-1 WT File Header Members

Offset	Description
`u32 offsetPercussiveInst`	Byte offset to an array of percussive instrument `WTINST` structures.
`u32 offsetMelodicInst`	Byte offset to an array of melodic instrument `WTINST` structures.
`u32 offsetRegions`	Byte offset to an array of `WTREGION` structures.
`u32 offsetArticulations`	Byte offset to an array of `WTART` structures.
`u32 offsetSamples`	Byte offset to an array of `WTSAMPLE` structures.
`u32 offsetAdpcmContext`	Byte offset to an array of `WTADPCM` structures.

4.2.2.2 Melodic and Percussion Instruments

The WT file contains the WTINST structure arrays for melodic instruments and percussion instruments. Each array will have 128 entries with each array element corresponding to the MIDI program number.

Code 4–2 WTINST

```
typedef struct WTINST
{

    u16 keyRegion[128];

} WTINST;
```

The WTINST structure represents each instrument that comprises the wave table. The member of this structure is a table of an index to the region information (WTREGION) that corresponds to each key. Each region index will take a value between 0 and 65,534 (65535 is used when the region is not specified).

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 will wait for an array of region information structures (WTREGION). The size of the array will vary for each wave table. The key number and the region information will not always correspond one-to-one, and more than one key number accesses the same region information. Members of the WTREGION structure are described in Table 4-2.

Table 4-2 WTREGION Members

Region	Description
u8 unityNote	The key number that will be the basis of the corresponding region.
u8 keyGroup	Key group number. Range is 1-15, and 0 indicates no group specification.
s16 fineTune	Fine tuning value for the pitch (0x0001 = 1 cent).
s32 attn	Volume (0x00010000 = 0.1dB).
u32 loopStart	Loop start position by sample unit.
u32 loopLength	Total number of samples in the loop.
u32 articulationIndex	Index to the articulator information (WTART).
u32 sampleIndex	Index to the sampling data information (WTSAMPLE).

4.2.2.4 Articulator

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 waits for an array of articulator information structures (`WTART`). The size of the array varies for each wave table. Each `WTART` structure is accessed by more than one region information (`WTREGION`). The members of the `WTART` structure are described in Table 4-3.

Table 4-3 WTART Members

Articulator	Description
`s32 lfoFreq`	LFO frequency. The LFO is calculated using a sine wave divided into 64 steps per period. lfoFreq is a Δ (delta) step value to be added every audio frame (3 ms) (0x00010000 = 1 step).
`s32 lfoDelay`	LFO start delay time, expressed as the number of audio frames (3ms).
`s32 lfoAtten`	LFO volume amplitude (0x00010000 = 0.1dB).
`s32 lfoPitch`	LFO pitch amplitude (0x00010000 = 1cent).
`s32 lfoMod2Atten`	Per unit change in LFO volume through modulation wheel (0x00010000 = 0.1dB, $Attenuation_{dB}$ = `lfoMod2Atten` * (modulation wheel / 128)).
`s32 lfoMod2Pitch`	Per unit change in LFO pitch through modulation wheel (0x00010000 = 1cent, $Pitch_{Cents}$ = `lfoMod2Pitch` * (modulation wheel / 128)).

Articulator	Description
`s32 eg1Attack`	Volume envelope attack time. (unit Time Cents, $Time_{Sec}$ = pow(2, eg1Attack / (1200 * 65536)))
`s32 eg1Decay`	Volume envelope decay time. (unit Time Cents, $Time_{Sec}$ = pow(2, eg1Decay / (1200 * 65536)))
`s32 eg1Sustain`	Sustain volume for volume envelope (0x00010000 = 0.1dB).
`s32 eg1Release`	Volume change per audio frame (3 ms) in volume envelope release stage (0x00010000 = 0.1dB).
`s32 eg1Vel2Attack`	Scaling applied to volume envelope attack time. ($Scale_{TimeCents}$ = `eg1Vel2Attack` * (key velocity / 128)).
`s32 eg1Key2Decay`	Scaling applied to volume envelope decay time. ($Scale_{TimeCents}$ = `eg1Key2Decay` * (key number / 128)).
`s32 eg2Attack`	Pitch envelope attack time. (unit Time Cents, $Time_{Sec}$ = pow(2, eg2Attack / (1200 * 65536)))
`s32 eg2Decay`	Pitch envelope decay time. (unit Time Cents, $Time_{Sec}$ = pow(2, eg2Decay / (1200 * 65536)))
`s32 eg2Sustain`	Sustain pitch for pitch envelope (0x00010000 = 1cent).
`s32 eg2Release`	Pitch change per audio frame (3 ms) in pitch envelope release stage (0x00010000 = 0.1dB).
`s32 eg2Vel2Attack`	Scaling applied to pitch envelope attack time. ($Scale_{TimeCents}$ = `eg2Vel2Attack` * (key velocity / 128)).
`s32 eg2Key2Decay`	Scaling applied to pitch envelope decay time. ($Scale_{TimeCents}$ = `eg2Key2Decay` * (key number / 128)).
`s32 eg2Pitch`	Pitch envelope attack level (0x00010000 = 1cent).
`s32 pan`	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 sampling data information structures (`WTSAMPLE`). The size of the array varies for each wave table. Each `WTSAMPLE` structure is accessed by more than one region information (`WTREGION`). The members of the `WTSAMPLE` structure are described in Table 4-4.

Table 4-4 WTSAMPLE Members

Sample	Description
`u16 format`	Format. The following formats are supported: ADPCM (WT_FORMAT_ADPCM) 16-bit PCM (WT_FORMAT_PCM16) 8-bit PCM (WT_FORMAT_PCM8)
`u16 sampleRate`	Sampling frequency (in Hz).
`u32 offset`	Offset to sampling data from beginning of PCM file (sample units).
`u32 length`	Length (sample units).
`u16 adpcmIndex`	Index for the array of ADPCM data decode information (WTADPCM).

4.2.2.6 ADPCM Data Decode Information

Code 4–6 WTAPCM

```
typedef struct WTADPCM
{

   // values to program at start
   u16     a[8][2];           //  coef table a1[0], a2[0], a1[1], a2[1]....
   u16     gain;              //  gain to be applied (0 for ADPCM, 0x0800 for PCM8/16)
   u16     pred_scale;        //  predictor / scale combination (nibbles, as in hardware)
   u16     yn1;               //  y[n - 1]
   u16     yn2;               //  y[n - 2]

   // loop context
   u16     loop_pred_scale;   //  predictor / scale combination (nibbles, as in hardware)
   u16     loop_yn1;          //  y[n - 1]
   u16     loop_yn2;          //  y[n - 2]

} WTADPCM;
```

The WT file contains an array of ADPCM data decode information structures (`WTADPCM`). The size of the array varies for each wave table. Each `WTADPCM` structure is accessed from sampling data information (`WTSAMPLE`). The members of the `WTADPCM` structure are described in Table 4-5.

Table 4-5 WTADPCM Members

ADPCM	Description
`u16 a[8][2]`	ADPCM coefficient table.
`u16 gain`	Gain. 0 for ADPCM, 0x0800 for PCM16, and 0x0100 for PCM8.
`u16 pred_scale`	Frame information.
`u16 yn1, yn2`	History data.
`u16 loop_pred_scale`	Frame information for loop.
`u16 loop_yn1, loop_yn2`	History data for loop.

4.2.3 PCM File

The PCM file stores all the sampling data registered to the WT file. The offset information for each sampling data in the WT file will be initialized to PCM file load destination as an address. This address is passed as an argument to the synthesizer instance initialization function `SYNInitSynth`.

4.3 Synthesizer Application Notes

4.3.1 MIDI Bank Support

The SYN library does not support MIDI controller bank select (0x00 and 0x20). Also, the data conversion tool for SYN library, `dls1wt.exe`, converts only the bank 0 data, even if the original DLS file is composed of multiple banks.

4.3.2 MIDI Message Support

The SYN library supports the MIDI messages listed in Table 4-6.

Table 4-6 Supported MIDI Messages

MIDI Message	Description
`0x8n`	Note off
`0x9n`	Note on
`0xBn`	Control change
`0xCn`	Program change
`0xEn`	Pitch wheel

4.3.3 MIDI Controller Support

SYN library supports the MIDI controllers shown in Table 4-7.

Table 4-7 Supported MIDI Controllers

Controller Number	Description
0x01	Modulation wheel
0x06 & 0x26	Data entry (set pitch wheel range)
0x07	Volume
0x0A	Pan
0x0B	Expression
0x40	Hold pedal
0x5B	Effect send level (AuxA)
0x5C	Effect send level (AuxB)
0x62 & 0x63	Data entry disabled
0x64 & 0x65	Data entry enabled
0x78	All sounds off
0x79	Reset all controllers
0x7B – 0x7F	All notes off (same as all sounds off)

4.3.4 Calling SYNMidiInput()

Applications are required to disable interrupts when `SYNMidiInput()` is being called outside of the AX audio frame callback. MIDI messages input through `SYNMidiInput()` will be buffered into a synthesizer instance and will be executed when `SYNRunAudioFrame()` is called.

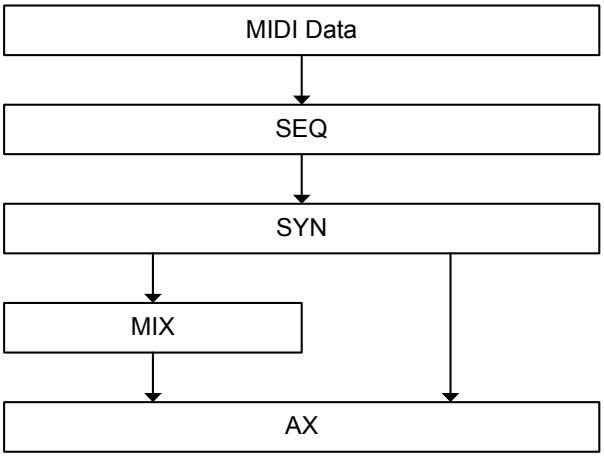
4.3.5 Shutting Down a Synthesizer

Applications need to guarantee that there are no active notes before shutting down the synthesizer with `SYNQuitSynth()`. Before shutting down, call `SYNGetActiveNotes()` to check the availability of active notes. Typically, notes will stay active until the ADSR envelope release phase is completed, even if the note-off message is input.

5 MIDI Sequencer

The MIDI sequencer library (SEQ) is an AX application library layer above the SYN library. The SEQ library processes the standard MIDI type 0 and type 1 data. The MIDI message analyzed through the SEQ library is passed to the SYN library.

Figure 5-1 SEQ API Layer



For detailed API descriptions, see “Sequencer” under “AX Applications” in the “Audio System” section of the *Revolution Function Reference Manual*.

5.1 Sequencer Features

SEQ library provides an API for the application to control the MIDI sequencer in runtime.

5.1.1 State Control

The user can set one of the following states in a MIDI sequencer.

Table 5-1 State Controls

State	Description
Stop	Stops the playback of MIDI data. The current position (playback position) will return to the beginning of the MIDI data.
Run	Starts the MIDI data playback from the current position. When the current position reaches the end of MIDI data, the playback will be stopped.
Run looped	Starts the MIDI data playback from the current position. When the current position reaches the end of MIDI data, the playback will resume from the beginning of the data.
Pause	Stops the MIDI data playback. The current position will not return to the beginning.

5.1.2 Tempo Control

A user application can set the tempo for any track through an API. Tempo will be expressed as a floating point number in BPM units (120.0 = 120 BPM).

A tempo can also be obtained from any track in the MIDI data. **However, the new tempo will then only be applied to that track. Be aware of this when using type 1 MIDI data.**

5.1.3 Volume Control

The user application can set the master volume for the sequence through an API. Volume will be expressed as a signed fixed point number in dB units (0x00000001 = 0.1 dB).

Also, the channel volumes can be obtained from MIDI data using control-change messages.

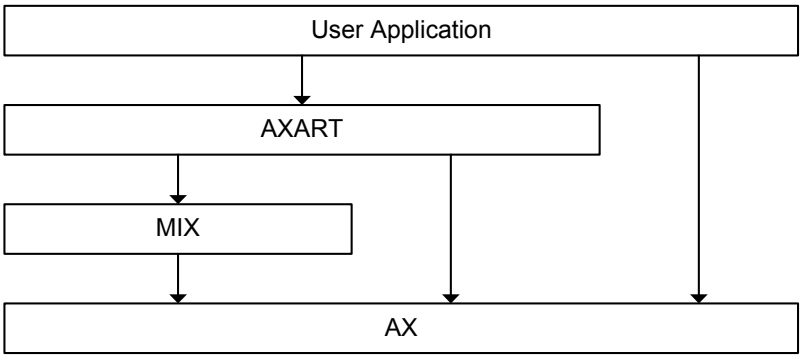
5.1.4 Controller Event Callback Interface

The user application may register callbacks to specific MIDI controllers. Any controller, 0 ~ 127, may be used as the destination, regardless of whether any function (such as a modulation wheel) is assigned. These events can then be inserted into a MIDI data to notify the user application of the current playback progress, for example.

6 Voice Articulator

The voice articulator library (AXART) applies articulators to AX voices, adding effects such as volume, pan, pitch, LFO, envelope, and 3D sound effects. Multiple articulators may be applied to a same voice.

Figure 6-1 AXART API Layers

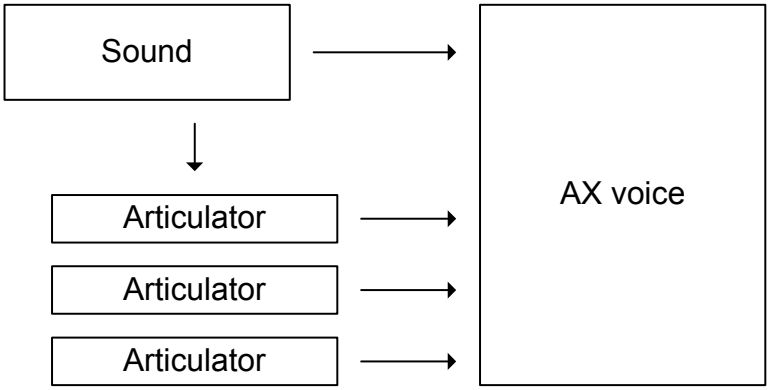


For detailed API descriptions, see “Articulator Library” under “AX Applications” in the “Audio System” section of the *Revolution Function Reference Manual*.

6.1 AXART and AX

The AXART library will control the articulator through the sound list, which sets a sound as a node. Each sound will have a pointer to its associated AX voice and a list of applied articulators.

Figure 6-2 Sound, Articulators and AX Voice



The articulators will each have a specific function and will set volume, pan, pitch, and other settings, following the user direction. Also, the user can change the articulator parameters at runtime.

The AXART library will run the articulators for all sounds in the sound list using the AX audio frame callback.

6.2 Articulator Types

Table 6-1 shows the articulator types and the voice parameters set by each articulator (indicated by ×).

Table 6-1 Articulators and Voice Parameter Support (1)

Articulator Type	SRC Type	SRC Ratio	ITD	Volume	Pan	SPan
AXART_3D	×	×	×	×	×	×
AXART_PANNING					×	×
AXART_ITD			×			
AXART_SRCTYPE	×					
AXART_PITCH		×				
AXART_PITCH_ENV		×				
AXART_PITCH_MOD		×				
AXART_VOLUME				×		
AXART_AUXA_VOLUME						
AXART_AUXB_VOLUME						
AXART_AUXC_VOLUME						
AXART_VOLUME_ENV				×		
AXART_AUXA_VOLUME_ENV						
AXART_AUXB_VOLUME_ENV						
AXART_AUXC_VOLUME_ENV						
AXART_VOLUME_MOD				×		
AXART_AUXA_VOLUME_MOD						
AXART_AUXB_VOLUME_MOD						
AXART_AUXC_VOLUME_MOD						
AXART_LPF						
AXART_FADER						
AXART_RMT						
AXART_RMT_FADER						
AXART_RMT_AUX_VOLUME						

Table 6-2 Articulators and Voice Parameter Support (2)

Articulator Type	AuxA	AuxB	AuxC	LFO	LPF	Fader	RMT	RMT Fader	RMT Aux
AXART_3D									
AXART_PANNING									
AXART_ITD									
AXART_SRCTYPE									
AXART_PITCH									
AXART_PITCH_ENV									
AXART_PITCH_MOD				×					
AXART_VOLUME									
AXART_AUXA_VOLUME	×								
AXART_AUXB_VOLUME		×							
AXART_AUXC_VOLUME			×						
AXART_VOLUME_ENV									
AXART_AUXA_VOLUME_ENV	×								
AXART_AUXB_VOLUME_ENV		×							
AXART_AUXC_VOLUME_ENV			×						
AXART_VOLUME_MOD				×					
AXART_AUXA_VOLUME_MOD	×			×					
AXART_AUXB_VOLUME_MOD		×		×					
AXART_AUXC_VOLUME_MOD			×	×					
AXART_LPF					×				
AXART_FADER						×			
AXART_RMT							×		
AXART_RMT_FADER								×	
AXART_RMT_AUX_VOLUME									×

6.3 Low Frequency Oscillators (LFOs)

AXART library supports the following common LFO shapes:

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

The user may also add LFO shapes not listed above.