Sound Buses

Sounds and Music can be assigned to a bus. This allows you to alter aspects of the audio, like the volume, without affecting audio assigned on another bus. Buses are like object groups in many respects.

Default Project

Create a project using Orx's init script.

Demo music for download

drums.ogg

bass-drums.ogg

Switching music

In this tutorial we're going to work through a scenario where we want to switch between two tracks. The tracks are identical in length, but have a different amount of instruments playing. The idea is that if you reach a dramatic part of the game, the more intense track will switch or fade in from the other.

Buses are perfect for this. Let's start by defining the two pieces of music, and giving them separate buses:

```
[NormalMusic]
Music       = drums.ogg
Loop        = true
Bus         = normalbus
KeepInCache = true
Volume      = 0.8

[DramaticMusic]
Music       = bass-drums.ogg
Loop        = true
Bus         = dramaticbus
KeepInCache = true
Volume      = 0.0
```

Download the two music tracks above and pop them into your data/sound folder.

Make a `MusicObject` and Add both music tracks to it:

```
[MusicObject]
SoundList = NormalMusic # DramaticMusic
```

Create the `MusicObject` in your init() function:

```
orxObject_CreateFromConfig("MusicObject");
```

Compile and run. The drum track will be playing. Actually both tracks are playing, however you'll only hear the drum track as it's volume is 0.8, while the bass/drums track is volume 0.0.

We can use the sound API to affect an audio bus, ie: volume changes, attenuation, pitch etc.

The function we need to change the volume of the bus is:

```
orxSound_SetBusVolume (orxU32 _u32BusID, orxFLOAT _fVolume)
```

Setting the volume value is simple enough, but what is the bus ID? Being a `orxU32` type, how do I get this? Remember above when we defined the music sections?

```
[NormalMusic]
Bus         = normalbus

[DramaticMusic]
Bus         = dramaticbus
```

The two sections were given a Bus id, and that was a string. We can retrieve the internal ID of this string name using:

```
orxString_GetID (const orxSTRING _zString)
```

from the string API.

Therefore if we wanted to mute the `NormalMusic` currently playing, we could do it with:

```
orxSound_SetBusVolume(orxString_GetID("normalbus"), 0.0);
```

And if we wanted to set the `DramaticMusic` to full volume, we could do it with:

```
orxSound_SetBusVolume(orxString_GetID("dramaticbus"), 1.0);
```

Multiple Sounds on one Bus

Multiple sounds can be assigned to the same bus. If each sound has it's own volume, using `orxSound_SetBusVolume` will set each sound to the same value. For example:

```
[MyMusic]
Music = somemusic.ogg
Bus = music
```

```
[MySFX1]
Sound = somesound1.ogg
Bus = sfx

[MySFX2]
Sound = somesound2.ogg
Bus = sfx
```

Then, to set the bus volume (if you used the Orx console):

`Sound.SetBusVolume sfx 0.4`

This will set all the MySFXs to 40% volume.

Bus hierarchy

Buses can have a parent/child relationship too. Let's say we have:

```
[SFX1]
Bus = main_sfx

[SFX2]
Bus = child_sfx
```

If you set up the relationship using the Orx Console:

`Sound.SetBusParent child_sfx main_sfx`

Or in code with:

```
orxSound_SetBusParent (orxString_GetID("child_sfx"),
orxString_GetID("main_sfx"))
```

You can then halve the volume with: `Sound.SetBusVolume main_sfx 0.5`.

This will halve both sounds because the child is connected to the parent `main_sfx`.

If you only applied the volume change to the `child_sfx`:

`Sound.SetBusVolume child_sfx 0.5`

Then only the child would be affected, not the parent.

Conclusion

This is only just a taste of what buses can be used for. Remember to check the sound API for other features.