Playing Sounds

Generating audio in the Torque engine is a snap. There are very few hoops to jump through when setting up and playing sound effects.

Descriptions

While it may seem confusing at first, playing a sound requires establishing two objects in memory – an `AudioDescription` and an `AudioProfile`. (Yes, I know profile and description mean the same thing. Tell it to the people who created Torque.)

Let's start with descriptions, since in many ways they are a parent to our profiles. An `AudioDescription` is a set of properties that determines how sounds are played *in general*. That means that several sounds can share the same `AudioDescription`. This will become clear when we look at some code, which we'll do in just a moment.

First, we need to establish how our sound will be used within Torque. Will it be localized to our computer or accessible across a network? The answer to this question will determine which keyword we use to create our sound. Localized audio is created using the `new` keyword, whereas networked audio is created using the `datablock` keyword, like so:

```
// Create a localized sound effect, which
// can only be heard on your computer.
new AudioDescription(localized_3D_sound){
      volume = 1;
      isLooping = false;
      is3D = true;
      referenceDistance = 1;
      maxDistance = 10;
      type = $DefaultAudioType;
};

// Create a networked sound effect, which
// can be heard by multiple computers.
datablock AudioDescription(networked_3D_sound){
      volume = 1;
      isLooping = false;
      is3D = true;
      referenceDistance = 1;
      maxDistance = 10;
      type = $DefaultAudioType;
};
```

Descriptions, whether created via `new` or `datablock`, take one argument – the name of the description. In the first description, it's "localized_3D_sound," and in the second, it's "networked_3D_sound."

Let's look at the properties within these descriptions. First we have `volume`, which is set to 1, or 100% of the actual volume of the sound file in question. Values greater than 1 will increase the volume of the sound, while values less than 1 will decrease it.

Next, the property `isLoop` determines if the sound in question loops, or plays one time. Music will probably loop, so a value of true is appropriate. On the other hand, explosions should only play one time, so a value of false is appropriate.

The `is3D` property determines whether the sound is ambient or specific to a particular source. For example, the sound made when you click on a button in a popup window is ambient – it exists outside of the game world. Music is also ambient (unless tied to a source in-game, such as a phonograph or radio). For these effects, we'd set the `is3D` property to false. On the other hand, a door opening, glass shattering and dynamite exploding are sounds tied to a particular source. Their volume and stereo position may change depending on where they happen relative to the player. In this way, the `is3D` property should be set to true.

The next two properties, `referenceDistance` and `maxDistance`, are only relevant if our sound is a 3D effect. The first property, `referenceDistance`, establishes the distance at which the audio is played at full volume. In this case, the sound effect will be played at full volume as long as it occurs within 1 world unit (about 3 meters) of the player. `maxDistance` indicates at what point the volume drops to zero. In this case, the audio will not be heard if it occurs more than 10 world units from the player.

Finally, we have the `type` property, which indicates what channel the audio will play from. Torque has three built-in channels, but others can be created. Our example description uses `$DefaultAudioType`, which is channel 0. Two other channels are `$GuiAudioType` (channel 1), used for the graphical user interface; and $SimAudioType (channel 2), used for in-game mission objects.

Profiles

Notice how neither of these descriptions reference a particular file? That's what our `AudioProfile` is for. An `AudioProfile` essentially specifies which sound to play, and which `AudioDescription` to play it with.

Profiles, like descriptions, also fall victim to the stigma of being either localized or networked, and so we again need to determine if we use the `new` or `datablock` keyword. Let's looky codey:

```
new AudioProfile(explosion){
     filename = "demo/data/sounds/explosion.ogg";
     description = "localized_3D_sound";
};
```

Profiles accept one argument – the name of the profile – which in this case is "explosion."

Profiles have two properties, `filename` and `description`. `Filename` is simply the path to our intended sound file. `Description` is a reference to an

`AudioDescription`, in this case "localized_3D_sound," which we created a few pages ago.

Simple enough!

Playing the Sound

Now that we know how to prepare audio for playing, it's time to investigate methods for actually playing it. As it turns out, we have a few tools in our Torque tool belt.

Playing Sounds Manually

The easiest way to play a sound is by calling the `alxPlay()` function. We do so like this:

```
%obj = alxPlay(explosion_2D);
```

Wow, that was ridiculously easy! `alxPlay`() takes one argument, the name of the `AudioProfile` for our sound. It returns an ID number, which we store in `%obj`. This only works, however, for localized, ambient sounds. Networked or 3D sounds must be played some other way.

Playing Sounds through the GUI

It is also possible to link sounds to GUI controls. Again, these sounds must be localized, ambient sounds. Let's take a look:

```
// The following commands can be executed via
// the console, or planted in functions
// elsewhere.
GuiButtonProfile.soundButtonOver = "over_2D";
GuiButtonProfile.soundButtonDown = "press_2D";
```

Both of these properties (`soundButtonOver` and `soundButtonDown`) reference the `GuiButtonProfile`, which is responsible for the behavior of default buttons in Torque. Assigning an `AudioProfile` to either of these properties will cause the sound to play whenever the mouse moves over the button, or presses the button, respectively.

Now, because these commands are assigned to GuiButtonProfile, they will play for every button assigned that profile. If you want to assign a sound to a particular button, then that button should be assigned a unique GUI profile. This is beyond the scope of this tutorial – a topic for another day, so to speak.

Playing Sounds through Special Effects

Finally, sounds may be attached to particular special effects in the Torque Game Engine. This is accomplished via the `soundProfile` property of certain datablocks. These sounds, unlike the other methods, may be either localized or networked, and ambient or specific.

The execution of these sounds varies across datablock type. Let's look at a few examples:

```
datablock ProjectileData(arrow_shaft){
      // Here we would normally list all of this
      // datablock's properties, however we are
      // really only interested in…
      sound = "whistle";
};
```

Here, we create an arrow as a projectile. As the arrow travels through the air, it generates a whistling sound.

```
datablock ExplosionData(bloody_arrow_hit){
      // Here again we'd normally list all of this
      // datablock's properties, however we are
      // really only interested in…
      soundProfile = "messy_thump";
};
```

Here, we create an explosion – actually, an explosion of gore when an arrow strikes its victim (yuck!). When the explosion occurs, it plays a messy "thump" sound, indicated by the `soundProfile` property.

How Do I Turn This Thing Off?!

The other side of the audio coin is stopping sound in its tracks. This is particularly important when playing music.

Remember when we called `alxPlay()`, and how we stored the sound's ID number in a variable? Well, we need to reference that ID when we want to stop our sound, which looks like this:

```
alxStop(%obj);
```

Of course, if we're playing music, we might want to store its ID number in a global variable, like `$music_ID` or something catchy.