Sound Recording

Prerequisites

Right now you need the svn version of orx in order to actually be able to record sound. However this is subject too change in the next couple of weeks. You can find it at:
http://orx.svn.sourceforge.net/viewvc/orx/trunk/

We will now present some basic recipes for common tasks and then cover the advanced settings and the whole API later.

Capturing audio data to a file

So let's say all you want to do is to capture some audio data from the default input device into a file. This is fairly easy and can be done like this:

```
  // Let's start recording:
  orxSound_StartRecording("sound.wav", orxTRUE, 44100, 1);
  // _zName = "sound.wav", that's the name of the file where the captured
audio samples will be written
  // Its extension determines which file format will be used for
compression.
  // _bWriteToFile = orxTRUE: we'll start writing to file immediately
  // _u32SampleRate = 44100: 44100 samples will be recorded per second
  // _u32ChannelRate = 1 : the audio data will be mono
```

A sound.wav file will be created and recorded to. The format depends on the extension, here it's WAV. The supported file extensions/formats are: WAV, CAF, VOC, AIFF, AU/SND, IFF/SVX. If the file extension is none of these, RAW (ie. uncompressed) samples will be written.

You can decide if you want to write the recorded data to the file with _bWriteToFile = orxTRUE/orxFALSE. We'll see later how to decide this on a per packet basis.

Passing 0 to _u32SampleRate and/or _u32ChannelNumber will use orx's default values for them. Defaults values are 44100Hz for the sampling rate and mono for the channel number.

Once you think you have captured enough data you can stop recording like this:

```
  orxSound_StopRecording();
```

Processing audio data

Let's now say you want to do some fancy processing of the audio data, before or instead of recording it to a file.

First we need an event handler, that will receive the raw audio samples:

```
static orxSTATUS orxFASTCALL SoundProcessingHandler(const orxEVENT *event)
{
  switch(event->eType)
  {
    case orxEVENT_TYPE_SOUND:
      switch(event->eID)
      {
        case orxSOUND_EVENT_RECORDING_PACKET:
          {
            orxSOUND_EVENT_PAYLOAD* payload =
(orxSOUND_EVENT_PAYLOAD*)event->pstPayload;
some_fancy_library_process(payload->stRecording.stPacket.as16SampleList,
payload->stRecording.stPacket.u32SampleNumber,
payload->stRecording.stPacket.fTimestamp);

            payload->stRecording.stPacket.bWriteToFile = (some_fancy_test) ?
orxTRUE : orxFALSE;
          }
          break;
        case orxSOUND_EVENT_RECORDING_START:
          some_fancy_library_initialize();
          break;
        case orxSOUND_EVENT_RECORDING_STOP:
          some_fancy_library_finalize();
          break;
        default:
          // do nothing
          break;
      }
    default:
      // do nothing
      break;
  }
  return orxSTATUS_SUCCESS;
}
```

We have to register this handler to the game engine:

```
  orxEvent_AddHandler(orxEVENT_TYPE_SOUND, SoundProcessingHandler);
```

Now we can start to capture audio samples:

```
  //now let's start recording:
  orxSound_StartRecording("sound.wav", orxFALSE, 0, 0);
```

Afterwards, the first event our handler will receive has a orxSOUND_EVENT_RECORDING_START ID. After that, each time new audio data is available, an event of orxSOUND_EVENT_RECORDING_PACKET ID will be created. You can access the raw audio data through the payload of the event. Beside the

audio data it also contains the number of samples that were captured and the time at which the current samples were recorded(in seconds since starting the application). More precisely it's the timestamp of the first sample in the current package. Each audio sample is represented by a 16bit integer value, ranging from -32768 to 32767, 0 being an audio output level of zero.

Once again, when we are done we stop the capturing with:

```
  orxSound_StopRecording();
```

Which will be followed by a orxSOUND_EVENT_RECORDING_STOP event.

Doing both: Capturing to a file and processing the audio data

So what if we want to do both, capturing the data to a file and process it at the same time? Well that's not a problem.

We can specify for each sound packet if it needs to be recorded or not by changing payload→stRecording.stPacket.bWriteToFile.

We can also alter the samples directly in the payload array (payload→stRecording.stPacket.as16SampleList). If we want to use less samples, we need also to update their number (payload→stRecording.stPacket.u32SampleNumber).

If we need more space for our samples, we can't reuse the array pointed by the payload. Instead we can populate our own array and update the payload pointer and the sample number accordingly.

NB: We can't use a stack allocated array for this as the array has to be valid till the next sound event we receive in our handler. If we allocated dynamically this array, we'll be in charge of deleting when receiving a future sound event so as to not leak any memory.

Advanced technique

If we need more advanced setting such as only analyzing sound blocks of a given size, we can make a local copy in a buffer till we receive the correct amoung of data. *NB: We can then process it, modify it if needed and ask for it to be written to file if needs be. One can't assume the number of samples sent by orx will ever be constant or sent at a constant time interval.*

API

Here is an overview of the API, until the doxygen documentation is updated:

```
/** Sound recording info
 */
typedef struct __orxSOUND_RECORDING_INFO_t
```

```
{
  orxU32    u32SampleRate;                    /**< The sample rate, e.g.
44100 Hertz : 4 */
  orxU32    u32ChannelNumber;                 /**< Number of channels,
either mono (1) or stereo (2) : 8 */

} orxSOUND_RECORDING_INFO;

/** Sound recording packet
 */
typedef struct __orxSOUND_RECORDING_PACKET_t
{
  orxBOOL   bWriteToFile;                     /**< Write recording to sound
file? : 4 */
  orxU32    u32SampleNumber;                  /**< Number of samples
contained in this packet : 8 */
  orxS16   *as16SampleList;                   /**< List of samples for this
packet : 12 */
  orxFLOAT  fTimeStamp;                       /**< Packet's timestamp : 16
*/

} orxSOUND_RECORDING_PACKET;

/** Sound event payload
 */
typedef struct __orxSOUND_EVENT_PAYLOAD_t
{
  const orxSTRING               zSoundName;   /**< Sound name : 4 */

  union
  {
    orxSOUND                   *pstSound;     /**< Sound reference : 8 */

    struct
    {
      orxSOUND_RECORDING_INFO   stInfo;       /**< Sound record info : 16 */
      orxSOUND_RECORDING_PACKET stPacket;     /**< Sound record packet : 24
*/
    } stRecording;
  };                                          /**< Recording : 24 */

} orxSOUND_EVENT_PAYLOAD;

/** Starts recording
 * @param[in]   _zName                          Name for the recorded
sound/file
 * @param[in]   _bWriteToFile                   Should write to file?
 * @param[in]   _u32SampleRate                  Sample rate, 0 for
default rate (44100Hz)
 * @param[in]   _u32ChannelNumber               Channel number, 0 for
```

```
default mono channel
 * @return orxSTATUS_SUCCESS / orxSTATUS_FAILURE
 */
extern orxDLLAPI orxSTATUS orxFASTCALL
orxSound_StartRecording(const orxCHAR *_zName, orxBOOL _bWriteToFile, orxU32
_u32SampleRate, orxU32 _u32ChannelNumber);

/** Stops recording
 * @return orxSTATUS_SUCCESS / orxSTATUS_FAILURE
 */
extern orxDLLAPI orxSTATUS orxFASTCALL
orxSound_StopRecording();

/** Is recording possible on the current system?
 * @return orxTRUE / orxFALSE
 */
extern orxDLLAPI orxBOOL orxFASTCALL
orxSound_HasRecordingSupport();
```

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/en/tutorials/audio/sound-recording

Last update: **2025/09/30 17:26 (14 months ago)**