Language and Cognition Laboratory
Department of Psychology, Columbia University

Methods report 2003.01
Original version, April 2, 2003

Playing sounds from E-Prime InLine scripts

Jim Magnuson
magnuson@psych.columbia.edu

Abstract
This brief report demonstrates how to play sounds from InLine events in E-Prime. This is useful when you wish to have control over sequences of stimulus events when, for example, you need to use InLine scripts to monitor for mouse clicks (e.g., in order to limit clicks to particular regions), or to monitor a serial connection (e.g., for eye tracker data). Two methods, one scruffy and one neat, are described.

Acknowledgments
Thanks to Sara Burgess at PST tech support.

Notes
1. The “neat” method described here employs techniques that may be vulnerable to changes in E-Prime in subsequent versions

Please direct questions and comments to Jim Magnuson.

Why you might play sounds inside InLine scripts

In our lab, we use E-Basic scripting in E-Prime InLine events in two methods that make precise control over sounds using E-Prime Sound events untenable:

1. Continuously monitoring for mouse clicks in order to constrain allowable responses to particular locations (see Methods Report 2003.02)
2. Continuously monitoring for serial input from an eye tracker (see Methods Reports 2003.03, 2003.04, and 2003.05).

If, for example, we want a sound to play on each trial 500 ms after the onset of a visual stimulus, but we want to monitor for mouse or serial data during that 500 ms, we cannot insert a Sound event to play the sound file. The obvious solution is to play the sound within the InLine script that is doing the monitoring.

With help from PST tech support, we eventually figured this out. Some people might be able to infer the process from the E-Basic help, but you would have to start with fairly deep understanding of E-Basic and stimulus events. The purpose of this report is to ensure that no one in our lab – and hopefully not in too many other labs – has to struggle with this.

Two methods for playing sounds from InLine scripts

The next two sections describe methods for playing sounds from scripts. In both methods, you need to have a global soundbuffer variable, and that variable must be initialized before you make reference to it. In the ***Scruffy*** method, this is accomplished by creating a Sound event under *unreferenced objects*. E-Prime automatically creates and initializes everything you need. In the ***Neat*** method, we explicitly code everything we need directly. In both cases, of course, you must enable the sound device and set the sound parameters appropriately for your stimuli (under the *Devices* tab when you open the *Experiment* window from the *Edit* menu).

The script for each method sets up a very simple ‘experiment’ in which a picture of an object is displayed, and 1000 ms later, the name of the object is played through the sound device. A third section describes a third script, based on the Neat script, that provides some basic timing information.

Scruffy method

This section describes the solution found in the accompanying E-Prime script, *scruffysound.es*. This is easy. Create a Sound event under 'Unreferenced e-objects' by dragging a SoundOut icon. Change the name to 'MySoundOut' (note that if you use a different name, you'll have to change the references in the script shown below). Set the default filename to be any existing file (necessary to get it to initialize properly). Now, when you generate the script, E-Prime generates everything needed to define and initialize the necessary variables.

There are 6 lines you must add to your InLine script to play sound.

```
Dim theSoundBuffer As SoundBuffer
Set theSoundBuffer = MySoundOut.Buffers(1)
theSoundBuffer.Filename = c.GetAttrib("Target") & ".wav"
theSoundBuffer.Load
theSoundBuffer.Play
Set theSoundBuffer = Nothing
```

However, if you put these all in the same inline script, you will add the variable amount of time it takes to load the sound (at least a few msecs, even for small sounds). You can preload the stimulus by adding an InLine prior to your stimulus events that does the loading (i.e., the 4 lines of code preceding the 'play' command). This is how it was done in *scruffysnd.es* (see Figure 1).

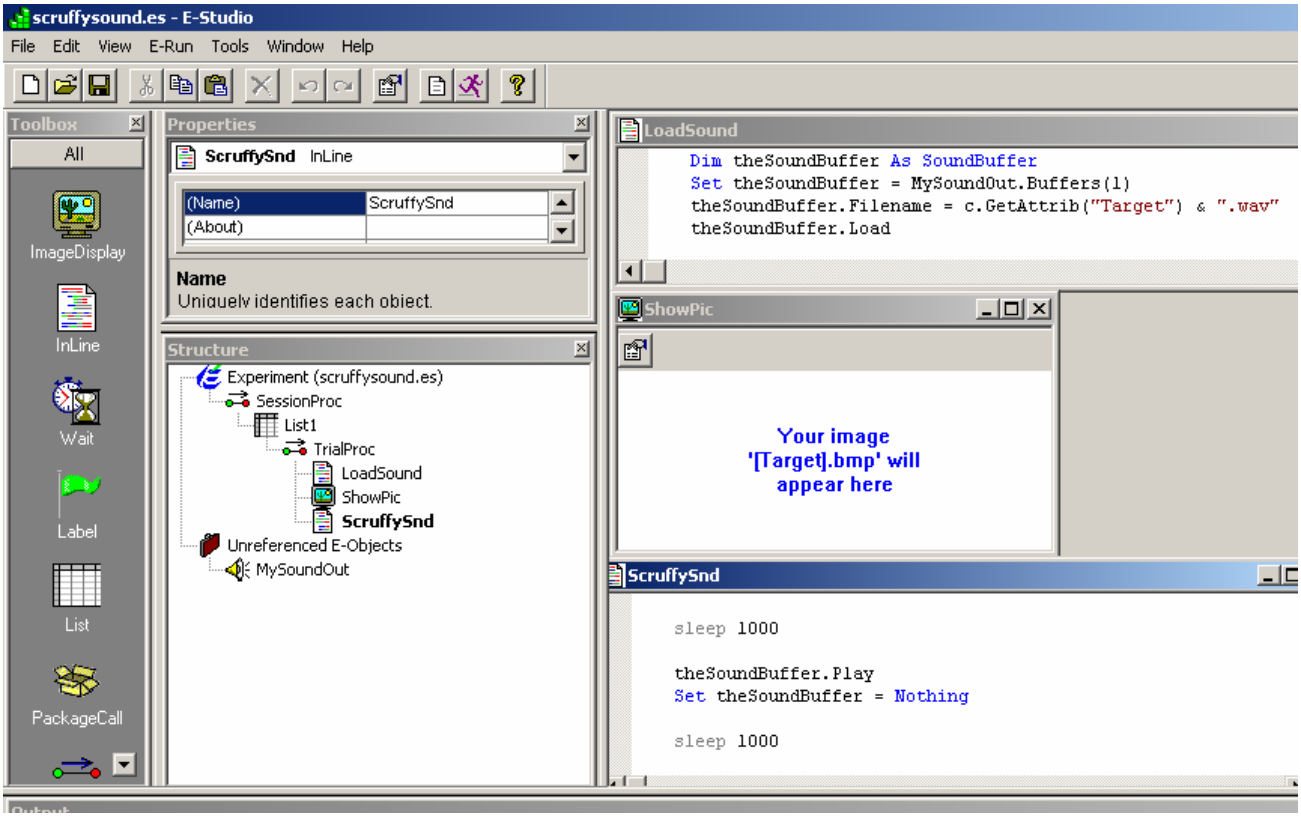

Figure 1: The scruffy solution.

Neat method

This section describes the solution found in the accompanying E-Prime script, *neatsound.es*. This method was developed by tracking down the appropriate code from the script generated by E-Prime for *scruffysound.es*, and putting it into the scripts directly. Why? Here are two reasons you might do it this way:

1. If leaving unreferenced objects lying around appeal to you aesthetically
2. If you are worried that some well-intentioned lab member might delete your unreferenced objects and disable your script

However, a word of warning is required: PST tech support told me that the Basic code for event types is among the aspects of E-Prime most likely to change in subsequent versions. This means that scripts using Basic code to generate things like sound objects may need updating when new versions of E-Prime are released.

This method requires the same 6 lines as the previous one (see the NeatSnd object in *neatsound.es*, also shown in Figure 2, and the code is reprinted on the next page). In addition, it requires you to declare three global variables under the *User* tab in the *Script* window (see Figure 2), and it requires that you initialize the variables (as in the MySoundOutCreate event (Figure 2). N.B.: the line:

```
MySoundOutSoundBuffer.Filename = "watch.wav"
```

may be confusing; you can set this to any existing file name; you need to do this in order to get it to initialize properly.

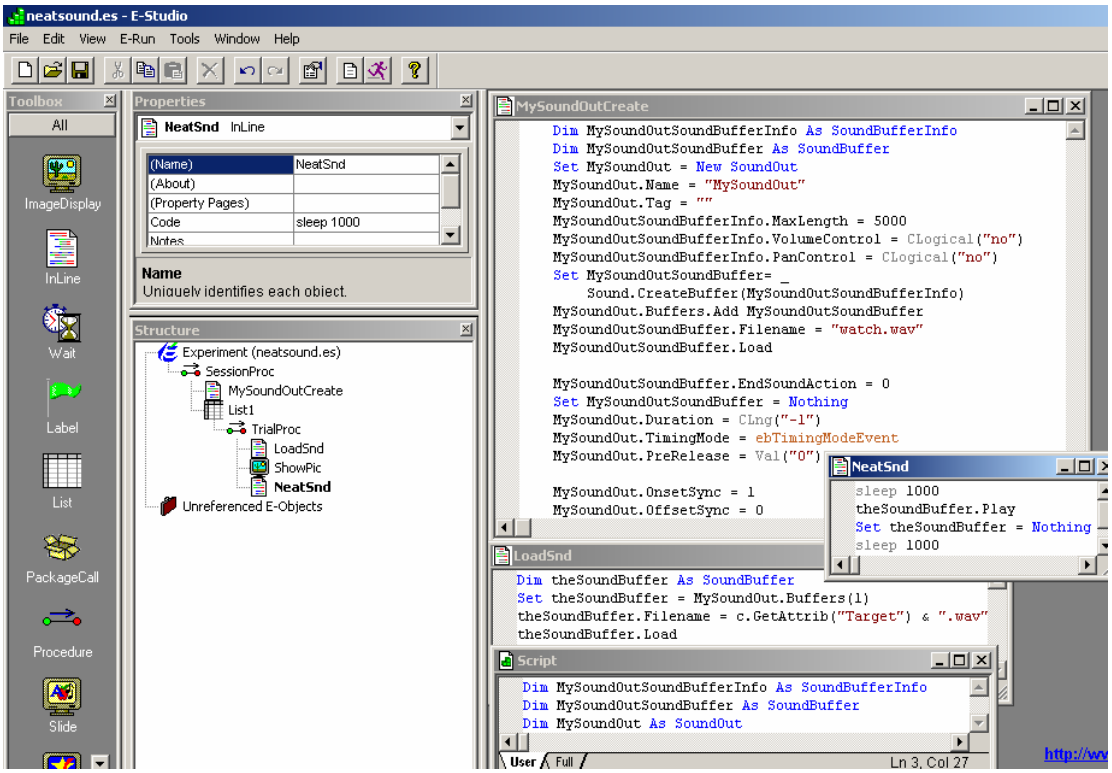

Figure 2: The neat solution.

Here, more legibly, are the three bits of code from Figure 2.

User area of script window

```
Dim MySoundOutSoundBufferInfo As SoundBufferInfo
Dim MySoundOutSoundBuffer As SoundBuffer
Dim MySoundOut As SoundOut
```

MySoundOutCreate

```
Dim MySoundOutSoundBufferInfo As SoundBufferInfo
Dim MySoundOutSoundBuffer As SoundBuffer
Set MySoundOut = New SoundOut
MySoundOut.Name = "MySoundOut"
MySoundOut.Tag = ""
MySoundOutSoundBufferInfo.MaxLength = 5000
MySoundOutSoundBufferInfo.VolumeControl = CLogical("no")
MySoundOutSoundBufferInfo.PanControl = CLogical("no")
Set MySoundOutSoundBuffer= Sound.CreateBuffer(MySoundOutSoundBufferInfo)
MySoundOut.Buffers.Add MySoundOutSoundBuffer
MySoundOutSoundBuffer.Filename = "watch.wav"
MySoundOutSoundBuffer.Load

MySoundOutSoundBuffer.EndSoundAction = 0
Set MySoundOutSoundBuffer = Nothing
MySoundOut.Duration = CLng("-1")
MySoundOut.TimingMode = ebTimingModeEvent
MySoundOut.PreRelease = Val("0")

MySoundOut.OnsetSync = 1
MySoundOut.OffsetSync = 0
```

LoadSnd

```
Dim theSoundBuffer As SoundBuffer
Set theSoundBuffer = MySoundOut.Buffers(1)
theSoundBuffer.Filename = c.GetAttrib("Target") & ".wav"
theSoundBuffer.Load
```

NeatSnd

```
sleep 1000

theSoundBuffer.Play
Set theSoundBuffer = Nothing

sleep 1000
```

Neat method – plus timing

This section describes the solution found in the accompanying E-Prime script, *neatsoundplustime.es*.

Figure 3 highlights what changes we make to do this. First, we add some global *Long* variables to handle the timing records to the user area of the script window (I know; not all of them need to be global – so much for the "neatness").

Then, add an InLine to read the clock right before the picture is displayed.

Finally, we add a few lines to *NeatSnd* so we can measure how much time elapses between the clockreads, and to print summaries in the Output window (via the Debug.Print method). From the output window in Figure 3, you can see that 1008 ms elapsed from the time the ShowPic event was called to when the next clockread happened **on every trial**. Without preloading (i.e., with the 4 lines from *LoadSnd* in *NeatSnd*), the latency is variable, but usually 1013 or 1012 msecs. Thus, we can chalk the 8 extra msecs observed here to the synching of the picture to to refresh synching (this test was done on a monitor with a refresh rate of 120 hz). Preloading is saving about 5 msecs. However, N.B. that this does not guarantee that the sound actually played 8 ms after the request; this is when E-Prime asked it to be played. Verifying the actual timing of the sound event is beyond the scope of this report (neat method: oscilloscope; scruffy method: audio and/or video tape and lots of trials to estimate variability in timing).

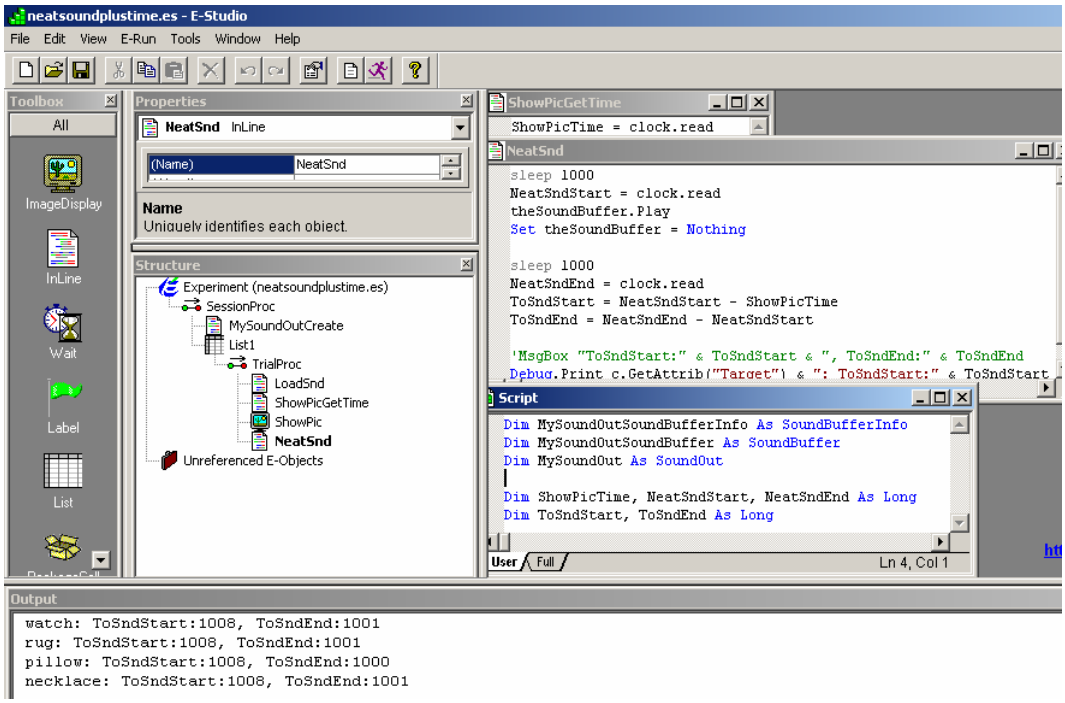

Figure 3: The neat solution plus basic timing.