Computers & the Humanities 281

Audio Application: Dialog

In teaching languages, sound files are an invaluable tool. Providing spoken examples of speech in the target language is an integral part of teaching that language to students. Even instruction in other subject areas may employ sound as an effective enhancement. These notes are designed to walk you through the process of creating a simple application presenting a dialog between two individuals in a target language. The principles covered here can be easily extended to any situation where sound is needed.

In the Beginning...

The first step is to digitize the dialog. If you have a commercially produced dialog on tape or compact disc, you'll need permission from the publisher to digitize the dialog. If you're recording your own, be sure you do it in a professional recording studio with qualified help. In either case, you'll need proper equipment to edit the dialog into discrete items. This will allow us to provide the opportunity for the user to listen to the dialog one line at a time. Revolution recognizes files of these types: wav & midi (Windows), aiff (Macintosh), and au (Unix).

As for presenting the dialog, we'll need to give the user a textual representation in the target language of what they hear in the dialog itself. Again, the separate lines of the dialog need to be visually distinct from each other. It would also be useful to provide in the native language a textual equivalent for each line of the dialog. A "Play" button next to each line indicates to the user that each line of the dialog may be heard separately.

Sounding Off

Sound files in Revolution may be accessed in two ways. Like images, we can have the data for the sound reside in the stack itself, or we can arrange it so that the data for the sound reside in a discrete file and have the stack reference that file. Again, similar pros and cons exist: With the data as part of the stack, accessing the sound is simpler, but large sounds quickly increase the size of your stack. As an external file, the sound will not increase the size of the stack, but an absolute path is created, so if the sound file is ever relocated, the link is broken and the stack becomes inoperable (with regards to sound). The former method is ideal for a small number of short sounds, while the latter is preferable for stacks utilizing a large number of long sounds.

To make a sound file part of your stack, select **Audio File...** from the **Import as Control** menu under **File**. From this same menu you may also choose **All Audio Files in Folder...** to save time if all your sounds are located in a single folder and ready to go. In referencing an outside file, you need to provide the full file path to that file.

The Play's the Thing

We now need to script the buttons so they give the user access to the sound files. Whether the sounds are imported or referenced, the Revolution command `play` is employed to present those sounds. The following are valid forms of the play command:

```
play audioClip "mySound.wav"
play "Question 1"
play audioClip "/user/localHost/clips/ouch.aiff"
```

The Play command in Revolution starts the sound playing, then goes on, executing any other commands that may be given. This allows the stack author to program things to happen while the sound is playing (such as an animation). It also allows the user to have control before the sound is finished. If there is anything else to click, the user can do it.

The advantages and disadvantages of this are obvious. The user could conceivably move on in the stack, all while the sound is still playing. While this may be desirable in some cases, it is extremely detrimental in others. We can keep control of their actions within the script by using:

```
wait until the sound is "done"
```

With this in the script, users can't do anything until the sound finishes playing. However, the button may give a different visual impression with the way it is hilighted. This is a case where we want our script to control hilight to provide the user with a proper visual representation. With this in place we now have a proper play button for the purposes of our activity. This could safely be empolyed for each button in the dialog.

But wait. The idea of having essentially the same handler in several different buttons should strike you as sick and wrong. With a little bit of thought, the handler could be designed to be placed in the card (or a group) script and still work for all buttons concerned.

Do You Hear What I Hear?

In a dialog exercise it's always nice to have a capability to hear the entire dialog at once, rather than being forced to listen to it one statement at a time. To that end we could create a "Play All" button which would play each clip in sequence. This is a situation where we could use the `send` command and loop through buttons, sending a `mouseUp` to each in turn. Just make sure you leave unwanted buttons out of the loop appropriately.

This would also be an appropriate place to visually indicate to the user that the computer is busy. Something along the lines of `set the cursor to watch` would work just wonderfully. We also have to ensure that the user is unable to do anything while this is occurring. If the user clicks randomly around the card while the sounds are executing, all mouse events are recording and will execute appropriate handlers in turn. Including `put flushEvents(all) into temp` in the handler will cause all pending messages to be ignored, effectively rendering useless the user's impatient clicking.

It's All Greek

Now let's deal with the translation fields that accompany the dialog. In order for the activity to be productive, we don't want the translations to always be showing. In fact, let's turn them into pop-up fields. We could add an appropriate button for each line of dialog to display the translated text upon request.

What popup method would be best? Well, "click & click" requires unnecessary clicking, and the mouseEnter/mouseLeave combination is too "accidental," but the mouseDown/mouseUp pair appears the most desirable in this situation. Again, having the same handlers in several different buttons is bad form. Figure out a way around this.

It would probably be best to place the translation to pop up on top of the corresponding text in the target language. Just move them over and resize them accordingly. With respect to visual elements, it would be best if the properties of the fields and the texts were identical, with the exception of the font color, in order to tie the two together but still indicate a difference.

Bells and Whistles

As described up to this point, this dialog stack is quite functional, but also quite bland (vanilla ice cream compared to burnt almond fudge). You have several tools and skills and design principles in mind upon which you can draw to augment and extend beyond this meager example. Your assignment, then, is to create a dialog activity in a language of your choice.

1. Create a stack and entitle it ***YourName*--Dialog**. Give the stack an appropriate label (based on the language with which you are working).
2. Record phrases of a brief dialog between two persons (3-5 lines each person) in a target language. The PC lab has Sound Recorder on each computer (**Start:Programs:Accessories:Entertainment**). This would be on the PCs in the CLIPS lab as well. The Mac lab has Audacity, which should be on the Apple computers in the CLIPS lab.
3. Create a card with fields and buttons to show the dialog lines and allow users to play corresponding sound clips.
4. Include the capability to play the whole dialog, as well as individual lines. Also provide a way to view an English translation.
5. Use your creativity and experience with Revolution development to improve on the example created in class. Consider improvements to layout and presentation, options and flexibility, and visual elements as well as more efficient scripting.
6. Put a copy of your finished stack in the **Assignments** folder.

There you have it. As before, this is only one of a myriad of ways of approaching this type of activity. This is not necessarily the most effective or most efficient, but it works and works well. Hopefully, this will inspire your minds as to the possibilities that exist with Revolution and spark some creativity of your own.

Course Schedule
Main Page