Project 3: Walkie-Talkie

You will write a multithreaded program for your Pocket PC that implements an audio walkie-talkie. In Project 1, you programmed on top of the libraries of the code base. In this project you will program on top of components and messages. The difference is that components are active – they contain threads whose priority you can manipulate. The messages can be communicated not only within your program, but across the network.

Before you start

You should read the handout "Introduction To Windows CE Programming for Pocket PC" if you haven't already. You should also carefully read "Pocket PC Code Structure". You will be using the following components and features of the code base:

- Messages and MessageQueues.
- Audio component (sound input and output)
- Communication component (UDP and Bluetooth connectivity)
- Router component (Maybe)
- Video component (video input and output) – for extra credit

Getting and installing the class code base

You'll need to update your code base:

```
cd ~/real-time/code/ppc
svn update
```

The update process causes your code base to be changed to reflect what's in the repository. If your code and the repository code can't be reconciled, subversion will indicate a conflict exists and make you fix it. Note that even if no conflicts exists, subversion can and will modify your checked out code tree. If this concerns you, you should make a backup copy first.

Note that there was an issue with the cygwin installation on the tlab machines (DOS versus Unix line endings). You may want to check out the whole code tree again to avoid this problem.

You will notice that you now have a series of component_* library projects, as well as a new application project, example_component, that shows how to incorporate the components into an application.

Adding your project and building

Similar to the first project, you will want to add a new project, "walkie-talkie", to your workspace. It should be dependent on all of the component libraries. There are a fairly

large number of include paths, library paths, and libraries that you will need to incorporate. You should copy these from example_component.
As with the first project, you are creating a simple hello-world application. You should now add the following lines to your walkie-talkie.cpp:

```
#include "component_video.h"
#include "component_audio.h"
#include "component_discovery.h"
#include "component_comm.h"
#include "component_route.h"
```

If your program compiles and links, you have the configuration correct.

All communication in this project is via UDP packets sent to port 5199.

How will this walkie-talkie work?

The user interface of your walkie-talkie will be very simple. There will be a simple dialog box to choose the IP address of the companion walkie talkie.[1] Once this is set, the walkie-talkie will be in "Hold" mode. You'll tap on the screen to switch to "Record" mode and again to switch to "Play" mode. Taping again will land you back in "Hold."

In hold mode, you will do nothing.

In record mode, you'll acquire audio data using the audio component. The audio component will hand you PPCAudioMessages. You'll set the address and other fields of these messages and pass them to the communication component to be transmitted.

In play mode, you'll wait for PPCAudioMessages from the communication component and play them back using the audio component.

A helper app

We will give a binary that implements the walkie-talkie on a desktop windows computer. You can communicate to that from your pocket pc. However, your implementation should also be compatible with the implementations of your classmates.

[1] The instructions in the following assume that your PPC is on the CS wireless network in 1890 Maple. On main campus, NUIT runs the wireless network and requires that you use a VPN. Although it is possible to set up the PPCs with the VPN, the receiving walkie-talkie must effectively run as a server, which creates several issues with the VPN. The upshot is that you're welcome to try to get this working on main campus, but you'll have an easier time of it in 1890 Maple.

One thing you could try in main campus is to use 802.11 ad hoc networking. If you configure your PPC in this mode, it is capable of talking directly to another PPC (or other device) also running in ad hoc mode. The default mode ("access point") requires that all communication go through an access point – a wireless bridge or router.

A deeper understanding

In this lab, we sample audio at a rate of 8 KHz, generating an 8-bit sample every 1/8000 of a second. This is sufficient to cover frequencies from 0 to 4 KHz (the voice range) with reasonable accuracy. It also means we generate 8 KB of data every second. We do not use any kind of lossy or lossless compression of the audio. For comparison, raw high quality audio is typically 24-bit samples taken at a rate of 96 KHz, covering 0..48 KHz well and generating about 288 KB of data every second; or 1-bit samples taken at 2.4 MHz and generating about 300 KB of data every second.

A UDP packet can be up to about 64 KB in size. However, such large packets will be fragmented over many smaller packets in most systems. For example, the maximum packet size on 802.11 is about 1500 bytes.

This is a very simple streaming audio application. In particular, it makes no attempt to find appropriate buffering levels or to recover from loss, corruption, or reordering in any way. It also makes no attempt at flow or congestion control. Why is this important?

You are using UDP. The network provides only best-effort delivery of UDP packets. That means that the packets you send may arrive corrupted, out of order, or not arrive at all. You may wonder why we're not using TCP, which has none of these problems. The problem is that our packets have decreasing value (and eventually zero value) with time. TCP treats all outstanding data with the same importance. Hence, we could easily be in a situation of wasting time retransmitting data that has ceased to be useful.

The walkie-talkie could add a sequence number to each packet sent to assure that they play back in order.

While the audio library does double-buffering[2] to avoid glitches locally, the walkie-talkie does not attempt to buffer audio to deal with transient network conditions. Imagine if, instead of playing every message as it arrives, we buffered the messages and played them back, say, 10 seconds later. This would let us paper over even a 10 second network glitch. The downside is that we would also introduce a 10 second delay.[3] Another question with such a buffer is just how deep it should be – this varies depending on network conditions and developing a reasonable feedback control algorithm[4] to adjust it is a challenging flow control problem.

Congestion control is a serious challenge with streaming audio. Unlike with data traffic, we can't simply slow down. We have to transform the audio to be of lower quality.

[2] "double buffering" just means "have two buffers. write in one while the machine plays/draws/etc the other. " If you write faster than the machine plays back, then it will always have a new buffer ready to be played back when it is done with the present one.

[3] By default, there is a 1 second record delay and a 1 second playback delay in the audio component. This can be reduced, but not my much. Unfortunately, the Windows wave API has very high latency and, at least at the present time, the lower latency directx sound API is not yet available on the PPC.

[4] i.e., make the buffer deeper when it underruns and shallower when it overruns.

The dialog box

We do not want you to spend an inordinate amount of time building a dialog box for this project. Here's the easiest way to do it: Create a dialog resource and then draw the dialog box. Name the fields (textboxes, etc) reasonably. Instantiate your dialog box in the same way as the about dialog box is instantiated in the hello world code. The only difference is that you will want to use Get and SetDialogItem to exchange information between your dialog controls and variables in your program.

If you have trouble, ask for help. This is not the place to get stuck. Also, initially, you can just hard-code the IP address.

Setting priorities and ordering queues

You will note that each component exposes its threads so that you can change their priority. At minimum, you will have three active threads in this application, audio recording, audio playback, and the user interface thread. You should play with thread priority to see its effects.

The audio and communication component both use first-come-first-served queues internally. You may also find it interesting to change these to priority queues and then use different ways to assign the priorities. For example, your priority could be the deadline of the message.

We hope to have available for you a utility that shrinks the available resources to make the effects of priorities and queue ordering more obvious.

For the ambitious (Extra Credit)

Extend your code to use the video component as well as the audio component. This would give you a video phone. One of the challenges in this is to fragment the video data across multiple messages since the UDP packet size limit is only 64K. A more important challenge (which you don't have to address), is keeping audio and video synchronized.