Project 1: Getting Acquainted With Windows CE Programming

The purpose of this project is to become acquainted with low-level programming on Windows CE using Embedded Visual C++. You will create a small program that takes a picture using the video camera when the user taps on the screen, and displays what it sees on the screen.

Before you start

It is essential that you read the handout "Introduction To Windows CE Programming for Pocket PC" before starting this project. If you don't, the project will seem much more daunting than it actually is. We are only asking you to write about 100 lines of code.

Getting and installing the class code base

In this class, you will be adding programs to a code base that we have provided for you. For the most part, this code base exists to simplify your life with very high-level APIs for services such as Zigbee, Bluetooth and IP communication, audio input and output, video input, and more.

In this project, you will add a subproject to the tree.

The code tree exists in a Subversion version control system. To check out a copy, you will do this (from cygwin):

```
cd ~
mkdir real-time
svn ssh:[we will tell you this in class]  code
```

After you do this, you will have a directory code with a subdirectory ppc that contains the pocket pc code base. The subdirectory mote contains the mote code base. You should now read the handout "Pocket PC Code Structure" to make sense of what's in the ppc subdirectory.

Adding your project and building

To open the Pocket PC codebase, do

```
cd ~/real-time/code/ppc
cmd /c PPC.VCW
```

Or , equivalently, open the ppc folder and double-click on PPC.VCW. This will open Embedded Visual C++ (EVC++) and open the pocket pc workspace. The workspace contains a large number of projects.

Now create a new project:

1. Right-click on "Workspace 'PPC' in the workspace window (file view) and select "Add New Project To Workspace…"
2. A dialog box presenting you with a number of different types of projects will be displayed. Choose the first option, "WCE Pocket PC 2003 Application". For project name, choose "myvideo". For location, choose "$HOMEDIR\real-time\code\ppc\apps\myvideo", where $HOMEDIR is your home directory. Note the addition of "apps" to the end of the directory. Choose "add to current repository", no dependencies, and select both CPU types.
3. At the next dialog, choose "A Typical 'Hello World' Windows Application", then Finish and OK.

You will now see a new project "myvideo" in the workspace and it will be in bold, indicating that it is the default project. Now:

4. Go to the project menu and select "dependencies". Make the "myvideo" project dependent on the "video" project. This means that EVC++ will build "video" before "myvideo".
5. Right click on "myvideo" and choose settings. In the Configuration box, choose "multiple configurations" and then select both ARM configurations. The ARM (or "Intel XScale") is the microprocessor that is inside a pocket pc.
6. Goto the C/C++ tab and to category "preprocessor". In the "Additional include directories dialog", type "../../libs/video" – this is needed so that EVC++ can find the "video.h" include file that you'll be using.
7. Goto the Link tab and category "input". In the object/library modules section, add "video.lib" and "SDIOCam.lib". In the additional library path section, add "../../libs/video/ARMV4Dbg, ../../libs/SDIOCam". This is needed so that the linker can find the video library (our wrapper code) and the SDIOCam library (the lower-level interface to the camera).

You are now almost ready to build your first project.

8. Place your pocket pc into the dock and wait for active sync to connect to it. A "guest partnership" is fine.
9. Open "Mobile Device" from "My Computer" and then open "My Pocket PC". This is the root directory of your pocket pc and is where we will deploy our app.
10. Copy code/libs/SDIOCam/SDIOCam.dll, code/libs/BTAccessMobile/*.dll and code/libs/Bluetooth/ARMV4Dbg/Bluetooth.dll to this directory. These are shared libraries that we will be using during the project. You only need to do this once.
11. From the EVC++ WCE configuration toolbar, choose "POCKET PC 2003", "Win32 (WCE ARMv4) Debug" and "POCKET PC 2003 DEVICE". This tells EVC++ that you want to build for the ARM processor using both the

basic Windows CE APIs and the Pocket PC 2003 APIs. It also tells it that you want to deploy your program right into the attached Pocket PC

Now, go to the Project menu and select “build myvideo.exe”. This will compile your hello world program and download it to the pocket pc.

You can now run your program using the Execute option (CTRL-F5) from the Project menu. You should see the PPC pop up an empty window with the words “Hello World” on it. The menu should work and will let you exit the program or get an about box.

You can also run the program under the debugger by using the Debug option (F5). With this option, you can set breakpoints and observe what’s going on just like with the debugger running on a regular windows program.

What do I do now?

You should first figure out how to determine when the user taps on the screen. There are a number of messages that windows can deliver. The one you want to process is “WM_LBUTTONDOWN”. Initially, you probably want to just change the text that is displayed when this message arrives and then send a message to yourself to make your redraw the screen (“WM_PAINT”, which can be sent using InvalidateRect()).

You should now read and understand the documentation for the video API presented in the file code/ppc/libs/video/video.h. In particular, you should understand and use the following functions:

InitVideo(HWND);
RequestPictureFrame(uint32, uint32);
DeinitVideo();

Think about when you need to call each of those functions…

You will need to handle the following messages:

PICTURE_FRAME_READY

What should you do when this message arrives?

Also note that the window procedure of your main window needs to include

VIDEO_CASE_DISPATCH()

in order to properly forward internal messages that the video library processes. If you decide to be clever, (or use MFC), you must assure that these messages are delivered or else you’ll never get a picture back.

The hardest part of this is to figure out how to get the data in the Frame that PICTURE_FRAME_READY message delivers onto the screen. Your first inclination will no doubt be to use SetPixel(). That's a good place to start, but it's pretty slow. Also notice that the picture size is of a different size than the screen. How do you change the size of the picture?

Do you need to draw the picture onto the screen only once?

A faster way to draw the picture onto the screen is to use CreateBitmap to create a bitmap from it and then copy it to the screen using BitBlt or StretchBlt.

At the end of this project, you should be able to click on the screen to take a picture, have the picture persist on the screen even after changing to another application and back, and be able to take a new picture at any time.

For the ambitious

Extend your code so that you take a movie.

Extend your code so that you stream your picture (or your movie) over the network to another pocket pc or some other computer and display it there. For this, do not worry about buffering or playback glitches. You will see later in the class the various tradeoffs involved here.