

Ryan O'Neill
May 05, 2005
CS197
Final Report

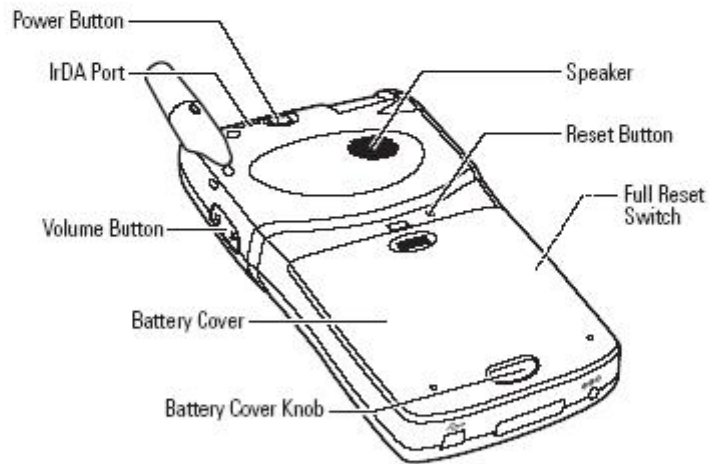
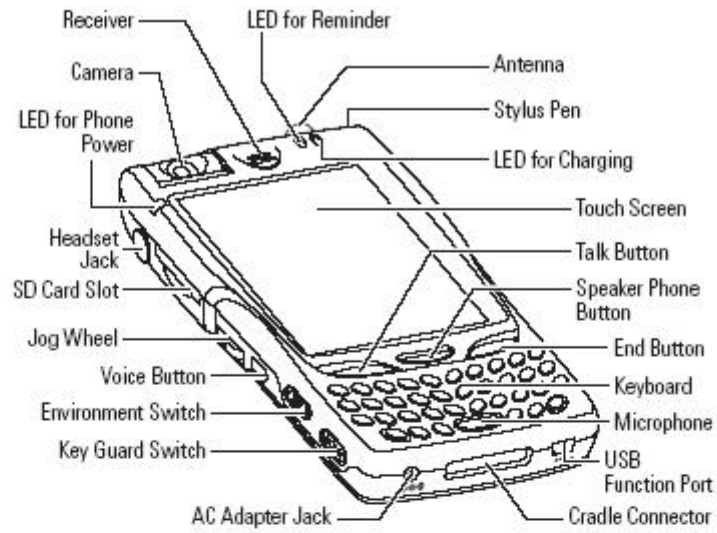
This final project uses two Sprint Pocket PC Mobile Phones. One is the Hitachi G1000 Pocket PC 2002 phone. This phone has a built-in swivel camera and an infrared (IrDA) port. The other is the Audiovox PPC6600, a Pocket PC 2003 phone, which also has an infrared port (and a camera, although it is not used for this project). The goal of this project was to have the two phones talk to each other. One phone, the Hitachi G1000, acts as the host, and the other phone, the PPC6600, acts as the client. The two phones communicate via the Infrared protocol. What is transmit between the two phones are settings for the G1000 camera and for when it should take a picture.

The Hitachi G1000 is a Sprint Pocket PC Mobile Phone that was released in June 2003. It is a Sprint only mobile phone, working only on the CDMA 1900 network. The processor is an Intel Xscale PXA255 400 Mhz (an ARM processor). The screen on the G1000 is full color and measures 240 by 320 pixels. It contains 32 MB of RAM and 64 MB of ROM and an MMC/SD expansion slot. The camera is VGA 640 x 480 pixels with a 4x zoom.



Hitachi G1000 Pocket PC 2002 Mobile Phone

Your PCS Phone's Features



Hitachi G1000 Hardware Diagram

The Sprint version of the Audiovox PPC6600 is a dual band CDMA 800 / 1900 Mobile phone and was released around October 2004. Unlike the G1000, the PPC6600 is available on other mobile networks including Verizon. The PPC6600 contains an Intel PXA 263 400 Mhz processor and runs Microsoft Pocket PC 2003 as the operating system. The screen is the same size as that of the G1000. It also has an IrDA port and a camera.

However, there are some major differences between the two phones. The PPC6600 runs Pocket PC 2003, whereas the G1000 runs Pocket PC 2002. The PPC6600 has built in Bluetooth technology. It has four times the amount of memory. It also supports faster networks, and has SDIO built-in, whereas the G1000 requires an installed driver.

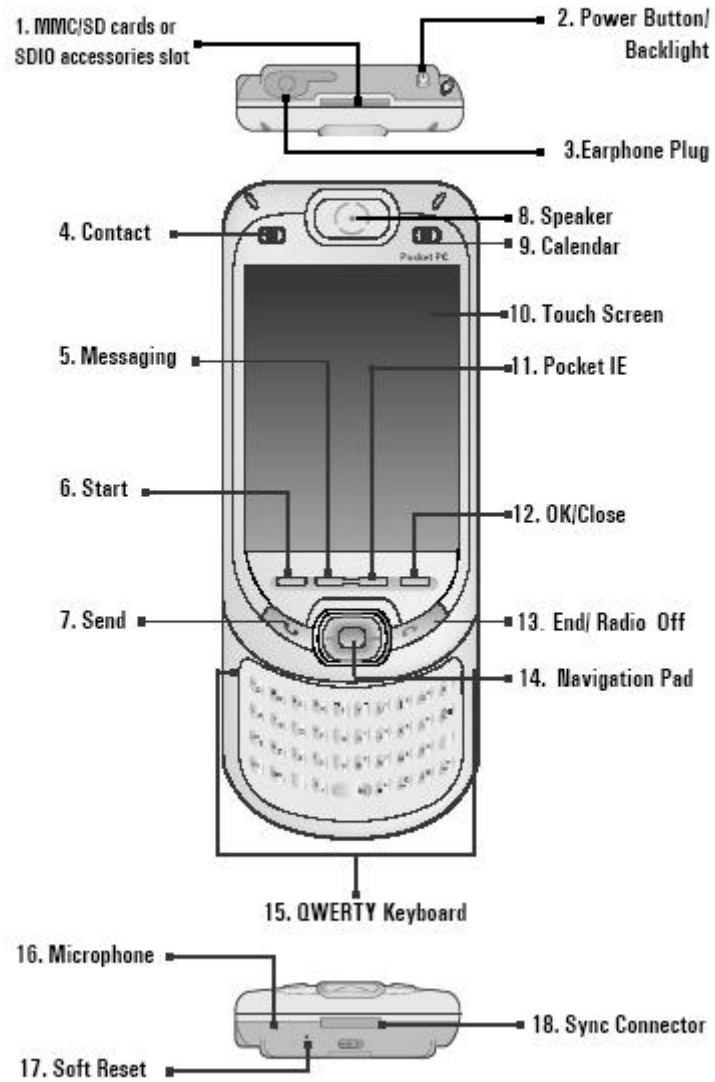
Most importantly, the imprint of the PPC6600 is much smaller than the G1000. First, the PPC6600 keyboard pulls out from the back, eliminating the real estate taken up by the keyboard on the G1000. Second, the G1000 has a sizable antenna that sticks out, where the PPC6600 does not have one sticking out at all. Aside from being shorter length wise, the PPC6600 is also shorter width and height wise as well.



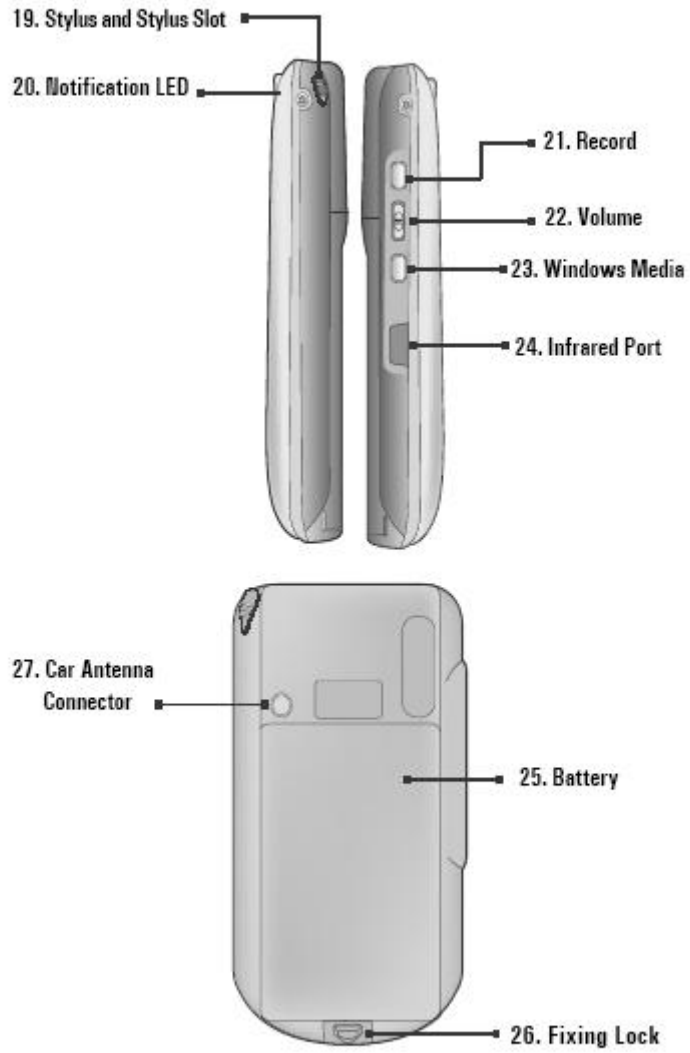
Audiovox PPC6600 with keyboard slid out

1.2 Your Sprint PCS Vision Smart Device

Front, Top and Bottom Side Panel Components



Back, Left and Right Side Panel Components



Section 1: Getting Started 7

The hardware components of interest for this project are the infrared port and camera on the G1000, and the infrared port on the PPC6600. The Camera Host has to be a Hitachi G1000. The Camera Host program makes use of the G1000's camera API that was released by Hitachi. The pocket pc running the Camera Client program can be any Pocket PC 2002 or 2003 device; here it is the PPC6600.

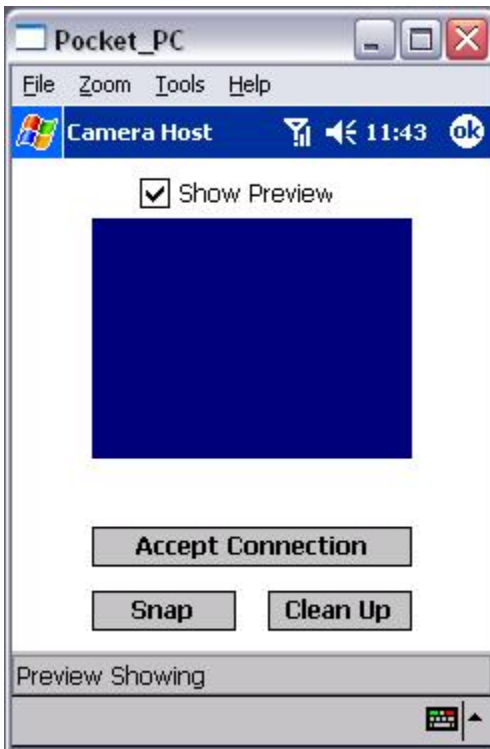
Microsoft's Visual Studio.NET 2003 was used to develop both the Camera Host and Camera Client programs. The same copy of Visual Studio was installed on two different computers. To the first computer, the G1000 was hooked up via a usb cradle. The PPC6600 was connected via a usb cradle to the other computer.

Because the PPC6600 is a Pocket PC 2003 phone, some other tools need to be installed before Visual Studio is installed on that machine. Visual Studio.NET 2003 cannot build applications for a Pocket PC 2003 device by default, only Pocket PC 2002. So, Microsoft's embedded Visual C++ version 4.0 (with patches) and Microsoft Pocket PC 2003 SDK were installed beforehand. This allows a developer access to the Pocket PC 2003 emulator and the ability to debug on the device itself.

The Camera Host program was written using Visual Studio .NET 2003 and the .NET Compact Framework in C#. The project contains five important files. The first file (FormMain.cs) is the main form of the project that the user sees when running the program. The second file (G1000CamLib.dll) is a dynamic link library provided by Hitachi that encapsulates the camera API for the G1000 phone. The third file (CameraWrapper.cs) is a static class wrapper for accessing the functions of the camera API contained in the dll. The fourth and fifth files are for saving the current picture number in an xml settings file.

The main form of the Camera Host program contains visual controls. There is a checkbox to toggle the camera preview window, and the preview window itself. There are also three buttons. One button (“Snap”) is for taking a picture directly using the G1000 camera. Another button (“Clean Up”) is for terminating the camera. Most applications that are run on Pocket PCs do not have an actual close button, only a minimize button. They are never really unloaded from memory. The Camera Host application uses a real close button. Because of the nature of programs for the pocket pc, the user must hit the clean up button before exiting, in order to reuse the program without resetting the device. The third button (“Accept Connection”) is for accepting an infrared command from the other pocket pc. When this button is pressed, a blocking call occurs where the Camera Host listens for the Camera Client to send it a message. The message can be a settings message or a take a picture message.

Unfortunately, settings and snap need to be two separate messages, because of the way the camera works. When the settings are changed, the camera needs a few seconds to readjust and redisplay the preview properly. Trying to do both the settings and snap in one shot caused the pictures to all be black because the camera did not have enough time to readjust to the new settings.



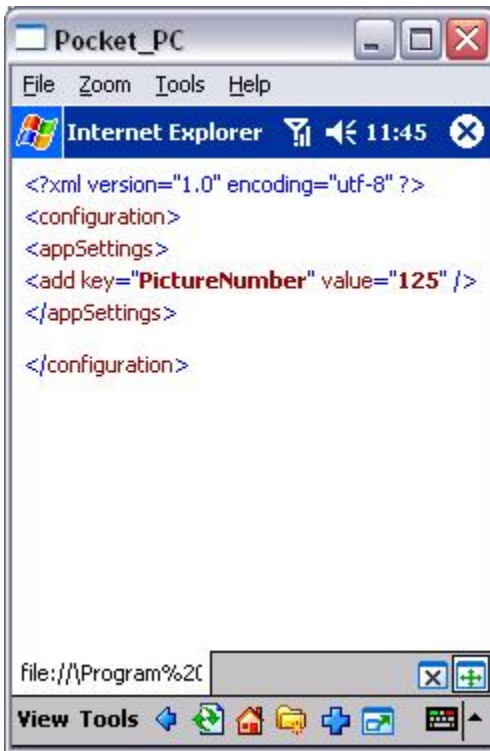
A Screenshot of the Camera Host Program

(Blue area indicates where Preview Pane is located)

The CameraWrapper.cs file is a class that contains all static methods. Each of the methods in the class corresponds to a function in the camera API dll. Within the .NET Framework, most code is managed code. Managed code can only access classes written for the .NET Compact Framework. In order to access the underlying hardware of the device, unmanaged code has to be used (written in usually either embedded Visual C++ or embedded Visual Basic, both of which are available free of charge from Microsoft). To tie in unmanaged code into a .NET Compact Framework project, a technique called Platform Invoke (P/Invoke) is used.

The settings file does not hold the camera settings information. The only piece of information it holds is the picture number. This is held in a file so that each picture taken winds up with a unique filename. To use the system clock or other resource to get a

unique identifier would have required another P/Invoke call because of the nature of the Compact Framework. The unique filename is important because when trying to save a new picture with an old filename, the old picture seemed to come back from the dead within the same running instance.



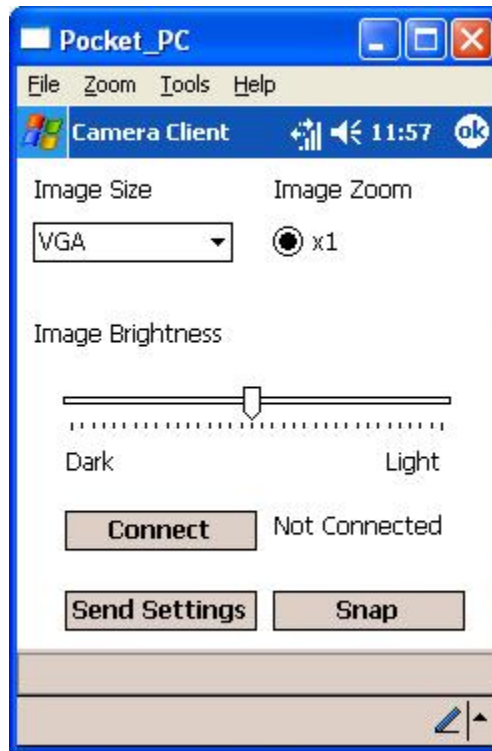
Settings.xml file containing PictureNumber

The Camera Client program only has a main form and some controls to change the settings of the camera and talk with the host program. A combobox allows the user to change the Size of the Image. Radio buttons allow the user to change the zoom of the image. A trackbar allows the user to change the brightness of the image.

When the Infrared ports of the host and client are in sight of each other, the client can connect (if the host is accepting a connection at the time). When connected, the client can send the new settings the camera should have or tell the camera to take a picture. Once the message is sent, the client disconnects immediately. So in order to

send another message, the host must be accepting a connection again and the client must reconnect. The reason for this limitation is functional, (as stated above) because when the settings change the camera needs time to readjust before taking a picture again.

Having the client need to reconnect gives it enough time.



Camera Client Screenshot

While being developed in Visual Studio .NET 2003, both applications used the same development and deployment methods. For development, the Solution Configuration was set to “Debug” and the Deployment Device was set to “Pocket PC Device”. Emulators could not be used for this project as the infrared hardware on both phones and the camera on the G1000 needed to be used by the programs. Building of the programs was done through the Build->Build Solution menu item. Then to debug the program, it was loaded and run on the actual device by choosing the Debug->Start menu item. Once the development of the applications was finished, the Solution Configuration

was switched from Debug to Release and the projects were rebuilt. Next, Cab files were created for the projects and Build->Deploy Project was chosen so that the programs were loaded on the devices.

The original scope of this project was much different than how it turned out. The idea at first was to have a Zilog Z8 development board be the one to talk to a G1000 camera over IrDA and have it control the picture taking. Because of the unfamiliarity with Visual Studio .NET 2003 and the .NET Compact Framework and the time it took to get up to speed, the Z8 portion of the project had to be replaced. A great deal of time was spent studying up on the .NET Compact Framework through various resources including “.NET Compact Framework Programming with C#” by Paul Yao and David Durant.

Developing using the camera in the G1000 also took much more time than was planned on. It took a while to figure out that the pictures could not be properly taken if the preview window is being displayed at the same time. Also, the camera cannot be properly shutdown if the preview window is being displayed. If the camera is not properly shutdown, then to run the program again and debug it, the phone must be taken out of the cradle, soft reset, stuck back in the cradle, resynchronized with the desktop machine, and the project rebuilt and reloaded onto the phone. This became a very tedious process, forcing over fifty soft resets of the phone.

Another original intention was to have the settings sent at the same time the message to take a picture was sent and have the settings change and the picture taken. This turned out to not be possible; because when the settings change the camera needs time to readjust. When the two were done in succession, all the pictures came out black, because the camera was not ready yet to take a picture.

The main goals of this project was to learn more about Visual Studio .NET, the .NET Compact Framework, and developing applications for Microsoft Pocket PCs. Those goals were accomplished with the working CameraHost and CameraClient programs. However, the goal of having two vastly different pieces of hardware communicate over a standard protocol was not accomplished due to time constraints and lack of prior knowledge.