Appendix 4: PDA

1. What is PDA?

Personal digital assistants (PDAs) are handheld computers that were originally designed as personal organizers, but became much more versatile over the years. PDAs are also known as pocket computers or palmtop computers. PDAs have many uses: calculation, use as a clock and calendar, playing computer games, accessing the Internet, sending and receiving E-mails, video recording, typewriting and word processing, use as an address book, making and writing on spreadsheets, use as a radio or stereo, recording survey responses, and Global Positioning System (GPS). Newer PDAs also have both color screens and audio capabilities, enabling them to be used as mobile phones (Smartphone), web browsers, or portable media players. Many PDAs can access the Internet, intranets or extranets via Wi-Fi, or Wireless Wide-Area Networks (WWANs).One of the most significant PDA characteristics is the presence of a touch screen.

1.1 Architecture

Many PDAs run using a variation of the **ARM architecture** . This encompasses a class of **RISC microprocessors** that are widely used in mobile devices and embedded systems, and its design was influenced strongly by a popular 1970s/1980s CPU.

1.2 Operating Systems Supported (OS)

The currently major PDA operating systems are:

- **Palm OS** - owned by PalmSource.
- **Windows Mobile (Pocket PC)** - based on the **Windows CE kernel** owned by Microsoft.
- **BlackBerry OS** - owned by Research In Motion.
- **Many operating systems based on the Linux kernel** - free (not owned by any company) These include
    - Familiar (comes in three flavours: GPE, Opie and barebone)
    - OpenZaurus (for Zaurus PDAs)
    - Intimate (for PDAs with lots and lots of memory)
- **Symbian OS** (formerly EPOC) owned by Ericsson, Motorola, Panasonic, Nokia, Samsung, Siemens and Sony Ericsson.
- **Windows Vista** - owned by Microsoft; has seldom use on PDAs.

2. Increasing Popularity

According to some modern studies, the overall market for PDAs grew by 20.7% in the third quarter of 2005, compared to the third quarter of 2004, with marketshare resolving as follows (by operating system):

- **Windows Mobile for PDAs** that comply with the Microsoft's Pocket PC specifications - 49.2% (increasing)
- **RIM BlackBerry for BlackBerry PDA** - 25.0% (increasing)
- **Symbian OS** - 5.8% (increasing)
- **Various operating systems based on the Linux kernel for various special designed PDAs** - 0.7% (stable)
- **Palm OS for Palm**- 14.9% (declining)

3. .NET Compact Framework (.NET CF)

The Microsoft **.NET Compact Framework** (.NET CF) is a version of the .NET Framework that is designed to run on mobile devices such as PDAs, mobile phones and set-top boxes. The .NET Compact Framework uses some of the same class libraries as the full .NET Framework and also a few libraries designed specifically for mobile devices such as Windows CE Input Panel.

It is possible to develop applications which use the .NET Compact Framework in Visual Studio.NET 2003 or in Visual Studio 2005, in C# or Visual Basic.NET. The resulting applications are designed to run on a special, mobile-device, high performance JIT compiler.

To be able to run applications powered by the .NET Compact Framework, the platform must support the Microsoft .NET Compact Framework runtime. Some operating systems which do include .NET CF are Windows CE 4.1 or later, Microsoft Pocket PC, Microsoft Pocket PC 2002 and Smartphone 2003. .NET Compact Framework applications don't run on the desktop computers with full .NET Framework, because, although the executable files are binary compatible, they use a different digital signature. This is used to prevent user from trying to run a full .NET application in a .NET Compact Framework device. The .NET Compact Framework is for devices that have the following characteristics:

- A capable CPU
- RAM for program and data storage
- Persistent storage, such as a disk drive or RAM disk

And this what is in our case for devices run on Windows CE 5 (WinCE 5)

3.1 .NET Compact Framework Capabilities

The .NET Compact Framework was designed to be consistent with the desktop .NET Framework. Like it or not, the desktop version always seems to be lurking in the background, like a worried parent or a homeless puppy. A crude way to compare the frameworks is by number of files and disk space occupied. The .NET Compact Framework has 18 libraries (including optional ones) and occupies about 2.5MB. The .NET Framework version 1.1, by contrast, has 86 libraries and occupies about 40MB.

3.2 Supported Runtime Elements

So what can the .NET Compact Framework library do for us? First, the .NET Compact Framework implements most of the Common Language Infrastructure (CLI), so the standard data types found on the desktop are also present for smart-device programming. At present, Microsoft supports two programming languages for the .NET Compact Framework: C# and Visual Basic .NET. This is quite a reduction from the 20 or more languages that support desktop .NET Framework. And yet, support for C# and Visual Basic .NET in the .NET Compact Framework shows that interoperability between languages is more than a promise; in the .NET Compact Framework, as on the desktop, we have direct evidence in the form of working, commercial-quality compilers that allow assemblies created in one language to freely interoperate with assemblies created in other languages.

The .NET Compact Framework supports an execution engine that provides Just in Time (JIT) compilation of IL and that verifies the type safety of managed modules before loading them. The runtime for the .NET Compact Framework provides the same memory management as on the desktop, enabling memory allocation, heap management, and garbage collection of unreachable objects.

The .NET Compact Framework supports application domains, which is a kind of lightweight process. Multiple application domains can run in a single operating system process. On some operating systems, processes have high overhead, and the cost of context switching from one process to another is high. That certainly describes processes on desktop versions of Windows, and the application domains provide a lightweight alternative.

A side effect of having lightweight processes is that there is a fairly low limit to the number of processes that can be created on a Windows CE system. That limit is 32, and it is so tightly ingrained in the memory and process architecture of Windows CE that we usually call this a hard limit, by which we mean that this limit is unlikely to change in future versions of the operating system. The operating system itself uses a few of these processes, which leaves 24 or 25 available for application programs. A .NET Compact Framework programmer might use application domains to avoid using up all the available operating system processes for large, complex applications.

The .NET Compact Framework runtime does support P/Invoke (Platform Invoke), the ability for managed code to call through to unmanaged code. There are some limitations in comparison to the desktop. One involves the complexity of the parameters that can be passed. Simple value types can be passed as parameters, as you might expect. Most reference types can also be passed as parameters, including strings, structures, and objects. But complex structures and complex objects—meaning those with embedded structures or embedded objects—cannot be marshaled automatically.

3.3 The .NET Compact Framework Provides CPU Independence

By meeting the goals for a framework with small size and good performance, the .NET Compact Framework team created a class library that is attractive to a wide range of application developers. For some developers, though, the most important feature is that .NET Compact Framework programs are CPU-independent. Five different processor families have been validated to run Windows CE: ARM, MIPS, PowerPC, SH, and x86. To build and ship native programs or libraries requires a large development and testing effort, especially when a target market needs support for several different processor families.

Things are much simpler with a .NET Compact Framework program or library. Instead of building and shipping five (or more) different versions, a single version can support all supported processors. (A single .NET Compact Framework program or library may require separate installation files, however, to accommodate CPU-specific setup programs.)

4. Windows CE Overview

As a result, the .NET Compact Framework supports all versions of the Pocket PC, including the first two generations (the Pocket PC 2000 and the Pocket PC 2002), which are built on Windows CE 3.0. This is noteworthy because, except for the Pocket PC, all Windows CE–powered platforms must run Windows CE version 4.1 or later to support the .NET Compact Framework (The third generation of Pocket PC, the Pocket PC 2003, runs this later version of the operating system.) It is worth noting that the .NET Compact Framework comes preinstalled on the PocketPC 2003 in device ROM but must be manually installed for the Pocket PC and Pocket PC 2002 devices.

To many people, Pocket PC and Windows CE are one and the same. This is due, in part, to the use of the term Pocket PC OS to refer to the software that runs a Pocket PC. Some people seem to assume that the Pocket PC is running not Windows CE but some other operating system. However, that is not the case: the Pocket PC is built on the Windows -CE operating system.

When starting to look at any technology, it helps to know what the creators of the technology had in mind during their development process. We start, then, with a discussion of the **design goals for Windows CE**. We have found that each design goal has a real impact on what an application programmer can do and that learning about these design goals helps programmers understand a wide range of choices made by the Windows CE development teams.

Design goals for Windows CE are the following aspects: Small, Modular, Portable, Compatible, Connected and Real time. Each of these aspects is needed to build the desired operating system.

5. Why .NET Compact Framework in Windows CE?

The reason is simple: Microsoft sees Windows CE–powered devices—including, of course, the Pocket PC—as part of its larger .NET strategy. Using the .NET Compact Framework, programmers can build traditional GUI applications that consume the services of Web Service servers.

With two different browsers—Pocket Internet Explorer (PIE) and Generic Internet Explorer (GENIE)—Windows CE–powered devices can also run Web Forms applications. An add-on to the ASP.NET Web server, ASP.NET Mobile Controls, allows for server-side customizations to Web-based applications for small-screen devices.

The .NET Compact Framework team adopted the following design goals for the .NET Compact Framework:

- Build on the benefits of the .NET Framework
- Maintain consistency with the desktop version.
- Ensure that the framework runs well on mobile and embedded devices.
- Expose the richness of target platforms.
- Preserve the look and feel of platforms.
- Provide portability to other mobile operating systems.

Some PDAs figures are shown at Fig .1

Fig .1 Three of devices running Windows CE derived operating systems: O2 XDA , iPaq H3870 and Sendo Z100.

6. PDA Sockets Programming

Network programming in WinCE is possible with sockets. A socket is like a handle to a file. Socket programming resembles the file IO as does the Serial Communication. You can use sockets programming to have two applications communicate with each other. The application is typically on the different computers but they can be on same computer. Sockets are a communication mechanism. A socket is normally identified by a small integer which may be called the socket descriptor. The socket mechanism was first introduced in the 4.2 BSD Unix systems in 1983. This stated that:

A **socket** is defined by a group of four numbers, these are

- The remote host identification number or address
- The remote host port number
- The local host identification number or address
- The local host port number

For any communication mechanism or socket mechanism the concept of protocol appears .A protocol is a set of rules that governs how information is delivered. For example, to use the telephone network, the protocol is to pick up the phone, listen for a dial tone, dial a number having a specific number of digits, wait for the phone to ring, and say hello. In radio, the station

uses amplitude or frequency modulation with a specific carrier frequency and transmission bandwidth, and you know to turn on the radio and tune in the station. In technical terms, no one protocol or set of protocols can be used for any communication situation. Be that as it may, communication engineers have found that a common thread runs through the *organization* of the various protocols. This grand design of information transmission organization runs through all modern networks today.

7. Protocol Architecture

What has been defined as a networking standard is a layered, hierarchical protocol organization. As shown in Fig.8, protocols are organized by function and level of detail.

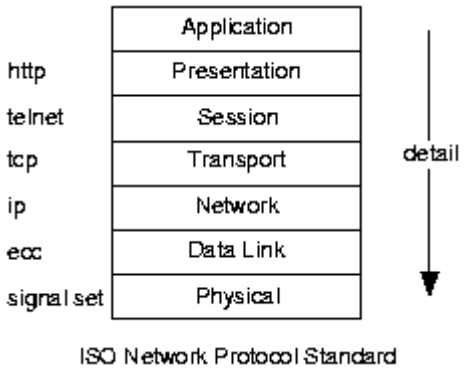

Figure .2: Protocols are organized according to the level of detail required for information transmission. Protocols at the lower levels (shown toward the bottom) concern reliable bit transmission. Higher level protocols concern how bits are organized to represent information, what kind of information is defined by bit sequences, what software needs the information, and how the information is to be interpreted. Bodies such as the IEEE (Institute for Electronics and Electrical Engineers) and the ISO (International Standards Organization) define standards such as this. Despite being a standard, it does not constrain protocol implementation so much that innovation and competitive individuality are ruled out.