Programming WinCE 3.0 for the Pocket PC

Dino Esposito, Artis.Net

The Pocket PC is a palm-sized, tablet footprint, mobile device — and a useful personal companion. Microsoft's Pocket PC platform combines the power of a personal information manager (PIM) with already-familiar software (with a high degree of compatibility with Win32 desktop computers) and a sophisticated hardware design to provide a device that's cheaper than a laptop and more usable than earlier palmtops.

Pocket PC is a hardware platform based on Windows CE 3.0, a 32-bit operating system designed to support a broad range of devices, including industrial controllers and consumer products (for example, cameras and phones). Pocket PC joins other CE-based platforms such as H/PC Pro, AutoPC and Palm-size. The differences between these are mostly determined by the nature of the hardware and typical usage of the device.

Microsoft provides Windows-like development environments and tools to allow you to create applications for CE-powered systems. The latest of these tools is **Microsoft eMbedded Visual Tools** — a CE-localized and shrink-wrapped version of more familiar programming environments such as Visual Basic and Visual C++.

The goal of this paper is to guide you through the major characteristics of the Pocket PC SDK, while providing design and coding principles specific to the platform. After a quick tour of the platform, I'll illustrate the Pocket PC runtime API and present a few Visual Basic and Visual C++ applications that touch on XML and ADOCE.

Pocket PC

The Pocket PC has been designed to offer two levels of services. It is primarily a PIM device that can be extended to become a tool for personal productivity. These two categories of functionality correspond to the two commercial versions of the product: "Standard" and "Pro". The difference between editions goes to the hardware elements, and the number and quality of the software components and services included in the ROM. Explorer and Inbox are part of the Standard Edition, but Pocket Word and Pocket Excel are not. XML support is also specific to the Pro Edition.

In terms of hardware, it is worth noting that the Pocket PC has no keyboard, but makes up for it with support for handwriting, and a software component that simulates a keyboard through tapping on the screen. The typical amount of RAM/ROM is 32Mb, with a CompactFlash slot available for further expansion.

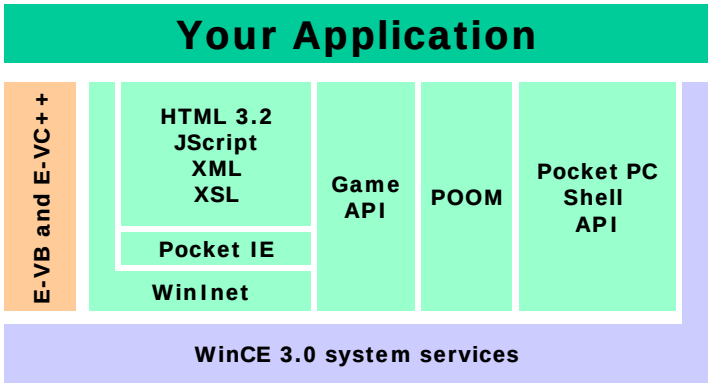
The main directive as far as Pocket PC applications are concerned is, "Be simple, not simplistic." *Simple* means having Windows affinity, and consistency of application behavior. *Simplistic* could mean a lack of UI guidelines and design principles, and looseness in the organization of the screen. Pocket PC allows for customization and user-specific adaptations, but within reason and subject to rules. For more details, look in the documentation that comes with eMbedded Visual Tools.

Platform Diagram

The Pocket PC is built on Windows CE 3.0 system services, including the real-time features in the new kernel. The base set of CE 3.0 functionality has been extended with a number of APIs that span from Internet support and XML, to a customizable shell and games programming.

The standard arsenal for a Pocket PC programmer is completed by a collection of "pocket" applications that reproduce the functionality of famous programs such as Outlook, Word and Excel in pocket format. Due to the limitations of the Pocket PC, some features (such as dynamic HTML) are not supported, and we have to make do with a made-to-measure mix of DirectX and HTML 3.2. XML/XSL and JavaScript have been luckier in the conversion process, and are fully supported (along with a number of other features [see below]).

How do you use all of these supported features together to produce and run an effective Pocket PC program? The diagram below clearly shows where your application sits in relation to the services provided by Windows CE, and what you can rely on during its execution:



The orange block on the left side of the diagram represents the tools at your disposal to write Pocket PC applications. The Microsoft eMbedded Visual Tools (MSEVT) toolkit comprises Visual Basic and Visual C++ for embedded systems, which are the newest versions of those products for Windows CE. The main difference between these and earlier versions is the support for the Pocket PC platform (in addition to H/PC and Palm-size), and their standalone nature. While Visual Basic and Visual C++ for Windows CE were extensions to the Visual Studio environments, MSEVT is a product in its own right, although the IDEs remain similar to those of its desktop cousins.

The CEF Format

Windows CE supports a wide range of processors and hardware platforms. When compiling and creating an INF file for application setup, you must specify the type of processor you're expecting. ARM, MIPS and SH3 are the main CPUs supported.

With Pocket PC (and right now, *only* with Pocket PC), however, there is a new type of binary code that is *not* specific to a particular CPU. This new format is called **Common Executable Format (CEF)**, an intermediate form of executable that gets translated on the target device. CEF is to Pocket PC as bytecode is to the Java Virtual Machine, and the file produced by the translation is native to the device's processor. An application that uses CEF-compiled files can run on all Pocket PC processors.

You can choose the CEF format while compiling using one of the embedded visual tools. To install a Pocket PC application on a device, you need a CAB file, and it's during this step that you can translate from the CEF to the native format. Application installation supports CEF by invoking the CEF translator on the device during setup. To support CEF in a CAB file, you need to:

- Compile all the files in CEF format
- Set the `ProcessorType` parameter in the `[CEDevice]` section of your INF file to zero (the conventional processor type for a processor-neutral binary format like CEF)

The following code snippet shows how to write a `[CEDevice]` section in an INF file to support CEF files:

```
[CEDevice.CEF]
ProcessorType = 0
```

> *The full name of the `[CEDevice]` section is `CEDevice.XXX`, where `XXX` is the target processor. If your binaries are CEF, use `CEDevice.CEF`.*

While building the installation routine, you might need to run a program called `Cabwiz.exe` — this is done automatically by EVB whenever you start the setup wizard. However, the EVB wizard *doesn't* list CEF as a possible target platform. The solution to this problem is revealed by a README file that explains how to rebuild the CAB file manually. Do this, and add the following command line switch to generate a CEF-specific cabinet file.

```
/cpu:CEF
```

But we're not done yet. Because CEF is supported only on the Pocket PC, you need to target the setup only to that platform. In order to do so, you must resort to a little trick. Create a list of *un*supported platforms that does not include "Palm PC", and indicate appropriate values for the minimum and maximum version numbers of the Windows CE operating system:

```
[CEDevice.CEF]
ProcessorType = 0
UnsupportedPlatforms = "HPC", "HPC Pro"
VersionMin = 3.0
VersionMax = 100.0
```

At heart, the Pocket PC is a Palm PC platform where the underlying version of Windows CE is at least 3.0. The content of `VersionMax` must therefore be at least 3.0, and the higher it is, the larger the number of future releases your CAB file will still be supported under.

Embedded Visual Basic

At first sight, the EVB IDE looks exactly like its desktop/laptop equivalent. On closer examination, though, you'll notice some minor differences in the menus and toolbars. What really differentiates EVB from the standard desktop edition, though, is the limited number of ActiveX controls. When you consider the significant differences between the UIs of the Win32 desktop and the Pocket PC, this is more than reasonable.

There are three categories of controls:

- **Standard controls:** The family of standard controls includes the textbox, label, button, shape, checkbox, scrollbar, list and combobox. One control that is noticeably lacking here is the picture control.

- **Specialized controls:** The menu bar control (a specific element of the Pocket PC UI), the Winsock control, the ADOCE control (see below), the FileSystem control, and a few others.
- **Third-party controls:** If the system-provided controls don't do what you need, you can always write your own Pocket PC ActiveX controls by using EVC. In fact, EVC is the *only* way to write custom components. Embedded Visual Basic supports only two types of projects: form and formless applications. You can't use it to create ActiveX controls or COM components.

ADOCE 3.0

ADOCE is a stripped-down version of ADO that primarily works with JET 3.5 tables. The JET 3.5 engine is the default OLE DB provider, although you also have the option of working with JET 4.0. JET tables under WinCE aren't identical to the MDB files you can read/write on desktop machines; the format used is somewhat simplified, and tables are stored locally on the device in file with a CDB extension.

Version 3.0 of ADOCE supports connection and recordset objects, but not commands and properties, and the objects that *are* supported have a limited set of functionality. For the most part, you should use SQL commands to accomplish operations: create and delete tables; add new records; delete existing ones. The following is a code snippet that creates a table in a CDB database:

```
Private Sub Command1_Click()
   Dim rs
   Set rs = CreateObject("adoce.recordset.3.0")
   rs.Open "CREATE table MyContacts (firstname text, lastname text, email text)"
   Set rs = Nothing
   MsgBox "Done"
End Sub
```

You can also use the ADOCE control instead of creating a dynamic instance of the ADO component through the `adoce.recordset.3.0` ProgID. Notice that the version number is important in terms of touching base with the right component.

The ActiveSync 3.1 component allows you to convert from CDB to MDB, and vice versa. In this way, you can import Access tables, and export Pocket PC tables to PCs. The Pocket PC SDK also makes available a CE-specific version of the OLE DB Simple Provider Toolkit to write CE versions of OLE DB providers.

Embedded Visual C++

As with EVB, Embedded Visual C++'s IDE looks much the same as its parent environment, and it's slightly richer than EVB. The number of available projects is significantly larger, and there's almost the same level of support for COM objects as in the Win32 desktop edition of Visual C++.

The MFC and ATL libraries both have CE/PocketPC counterparts that boast some new classes for UI-specific features, but many other classes have either been dropped or had their functionality reduced. There's support for a subset of desktop Win32 and C runtime functions, but no support whatsoever for the Standard Template Library (STL). Overall, the libraries are pretty stripped-down versions.

Apart from the UI classes, ATL implements almost the same set of features as the desktop version — in particular, dual and tear-off interfaces, aggregation, connection points, and enumerators. However, it cannot create out-of-process servers or modules that demand apartment threading models. ATL for Windows CE only supports the single threading or the free threading models.

ActiveX Controls with EVC

If you want to create a custom ActiveX control for your Pocket PC-based application, you can do so only with EVC. Using the same Win32 C++ code as for a desktop control is possible in principle, but depends on what the control actually does. However, you need to follow a relatively complex procedure to test and debug the control within the EVB or EVC environment.

Once you've compiled the control (say, `MyCtl`), rename its folder (say, `MyCtl_CE`) and create a brand new Win32 ActiveX control project with the original name (that is, `MyCtl`). Drop all the files from the WinCE project into this new folder, save the Win32 project, and rename its `.dsp` and `.dsw` files to `MyCtl_Win32.dsp` and `MyCtl_Win32.dsw`. Next, move these to the `MyCtl_CE` folder, get rid of the `MyCtl` folder (the Win32 project) and rename `MyCtl_CE` back to `MyCtl`. Now, in the `MyCtl` folder, you have identical copies of the component for both the desktop (`MyCtl_Win32.dsp`) and the Pocket PC (`MyCtl.vcp`). Compile the desktop project, and all the registry settings will be set, allowing testing of the WinCE control.

The point is that whenever you attempt to instantiate the ActiveX WinCE control during the development phase, you're working on a desktop PC. You need to have the WinCE control registered in the desktop registry, and the only safe way to go is to have an identical component for Win32. Remember not to choose lightweight ATL objects when creating COM components to be used to simulate Pocket PC components.

User Interface Guidelines

The Pocket PC looks a lot like desktop Windows, but it is not the same thing at all. By far the most important constraint for developers is the memory available. You must keep the footprint of your applications as small as possible, and be ready to answer to any hibernation messages that arrive. (`WM_HIBERNATE` is sent to an application when system resources are running low, and in response an application should release as many resources as possible. This means unloading dialog boxes and destroying windows, without killing the internal state.)

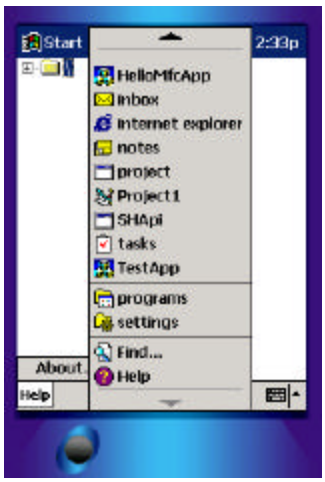
Here are a few more pieces of advice to consider:

- The single tap is the preferred way of accessing functionality, and this is what Microsoft recommends you to do. Any form of input is seriously compromised by the limited functionality of the hardware. For this reason, try to limit the need for using a keyboard as much as possible. When the keyboard is absolutely necessary, try to make it easier for the user by using SIP (Smart Input Panel) functions. For example, make the program display the keyboard set to handwriting or keyboard mode automatically, according to preferences.
- The shell can be customized by using your own DLLs. Exploit that feature to personalize all the devices running your application.
- Avoid floating objects such as windows and message boxes. Consider working with web application logic: page after page, where each page occupies the whole of the available screen.

Since Pocket PC applications are supposed to be very simple programs, the need for such things as online help should be minimal. However, if you do need it, avoid placing buttons all over the UI. Instead, integrate help with the application's system menu, and tailor it dynamically to the current context of the application.

The picture below shows two possibilities. You can use a Help submenu, with as few items as possible — ideally, just a couple: About, and the current topic.

Alternatively, you can associate a link to the current topic with the system's Help menu item.



Code Samples

During my presentation, and in the accompanying material, I've explained how to create an ADOCE application and an XML-based "to do" list editor in Visual Basic. With EVC, I've created an ActiveX control and demonstrated how to integrate it with EVB applications.

Links and Resources

The web site for the Pocket PC is http://www.microsoft.com/pocketpc. From there, you can find and follow links to other, more specific web sites where Pocket PC devices are listed, described, and available for purchase. Examples of such devices currently available are the Casio E-105, Compaq Aero, and HP Jornada, and new pocket devices are coming out at a rapid pace. Among these is a new, richer version of the HP Jornada (v548), and the iPaq.

Another web site that's worth a visit if you plan to use pocket PCs (in all respects) is http://www.avantgo.com. AvantGo is a custom service that lets you synchronize news sources to your PocketPC, Palm, or cellphone.

Lastly, more information about developing for Windows CE can be found at http://www.microsoft.com/windowsce/developer