

Secrets to Successful Development with Windows® Embedded

Embedded computing devices are practically everywhere these days. Items such as slot machines, kiosks, point-of sales terminals, TV set-top boxes, personal data assistants (PDAs), and industrial automation equipment are some to name a few. More often, companies are now creating these devices using Microsoft® Windows® Embedded operating systems to bring the power of Windows and millions of developers to these devices. This document looks at the two embedded operating systems that Microsoft offers, Windows CE.NET and Windows XP Embedded (XPe), and discusses why companies are choosing Windows Embedded, what is involved in completing an Windows Embedded project, and areas that require specific consideration to successfully deliver Windows Embedded products to market.

Why Windows?

Frequently, Windows is considered just a great graphical user interface for desktop PCs. What's often missed, are the additional benefits that this operating system offers, making it a strategic choice for all types of devices. The Windows Operating System provides a standardized platform that not only has wide hardware and software support, but communicates with other technologies, and is familiar to millions of developers worldwide.

A key advantage of developing platforms around Windows is that it allows companies to focus on a single software architecture and use standard PC development tools such as Visual Studio .NET and embedded Visual C++ to easily create value-added device components. Because Windows separates the OS and applications from the underlying hardware, designs that incorporate Windows Embedded can choose from a wide range of hardware and software components. This separation allows developers to concentrate on their specific value-add rather than having to become an expert in each piece of hardware or rewriting application code. The skills already present by millions of Windows programmers can be directly applied to developing Windows-based embedded devices.

Another compelling feature of Windows CE .NET and XPe are the built-in provisions for networking and Internet support. Items such as LAN, WAN, Dialup, Wired, and Wireless as well as the latest Microsoft browser are standard. Microsoft also designed both operating systems to be modular, meaning you can pick and choose which system components you want in your embedded device. For example, if you are developing a

headless device that does not require a display, you can leave out all the graphics modules to decrease the footprint and cost of the end device. This selection of components is done with easy-to-use visual development tools such as Platform Builder for CE .NET and Target Designer for XPe which will be discussed in detail later.

Next, we'll look at the Windows CE .NET and XPe operating systems to determine which is best for a given project.

Windows CE .NET

Windows CE .NET is the successor to Windows CE 3.0. As of the writing of this document, the current version of Windows CE .NET is version 4.2. For clarity, Windows CE was written from the ground up; it's not a direct subset of the desktop version of the OS as some are led to believe. This was to address the demands of the deep embedded market. Primarily it was designed to:

- Be extremely flexible in size and functionality;
- Continue to leverage the skills of millions of existing Windows developers;
- Include communications functionality with the operating system;
- Address the requirements of the embedded marketplace (e.g. cost, real-time, power management, etc);
- Be portable to many types of CPU architectures and device platforms.

As mentioned, Windows CE .NET is comprised of modules, each of which is broken into smaller subsets called components. These modules and components consist of different functionality to support target device designs such as Web Tablets, Smartphones, and others mentioned in this article. With Platform Builder, the primary tool for developing Windows CE .NET systems, you create your custom version of the operating system by picking and choosing which modules and components you want to include in your specific device.

The minimum footprint for Windows CE .NET is less than 350 KB (essentially the kernel); you then add the specific components your device needs. You can create task-specific devices with very small footprints, or devices with full Pclike functionality. Another cost and memory saving feature of Windows CE is that it can support XIP (execute in place), which allows the OS to run directly from Flash or ROM without first being shadowed in to RAM, thus saving additionally on memory costs.

Windows CE uses the Win32 programming model as with the desktop version of the OS and includes some of the most commonly used Win32 APIs. Windows CE also supports such technologies as the .NET Compact Framework, MFC, ATL, ADO, COM, MSMQ, ActiveX, CryptoAPI, DirectX, USB, Bluetooth, and 802.11 to name a few. This compatibility allows Windows application developers to leverage their existing skills and apply them to Windows CE-based devices.

Windows CE also supports a broad range of communication capabilities, including support for TAPI, Winsock, TCP/IP, WinInet, RAS, NDIS, serial, IrDA and many others. Windows CE has built in power management for instant on/ off capabilities, making it possible for CE-based devices to run substantially longer with increased functionality than a typical laptop. Users can instantly turn the device on and start right where they left off without having to wait for the OS to boot and resume.

Another feature of CE is its high degree of portability. Windows CE is not limited to just x86-based architecture. Windows CE .NET primarily supports four families of microprocessors:

- ARM. Examples of supported processors include ARM720T, 920T, 1020T, StrongARM, XScale, and OMAP
- MIPS. Supported processors include AU1500, VR4310, and 20Kc core.
- SHx. Supported processors include SH-3, SH-3 DSP, and SH-4.
- x86. Supported processors include 486, 586, Elan, Cyrix, Geode, Celeron, and Pentium I, II, III, and IV.

For a complete list of supported processors, check the list on the Microsoft CE .NET website at:

<http://www.microsoft.com/windows/embedded/ce.net/evaluation/hardware/processors.asp>.

If you are further interested in learning more about Windows CE, BSQUARE offers Windows CE training and consulting services to help companies get started on CE projects.

Windows XP Embedded

For XP Embedded (XPe), Microsoft adapted its desktop Windows XP operating system for the embedded environment. Like CE, XPe is also modular so that you can pick and choose the components that you want in your embedded device. With XPe you can choose from more than 10,000 individual OS features, services and drivers. Instead of using Platform Builder as the development tool, there is a suite of tools that you use to build your custom XPe OS. Target Designer is the

primary environment used to build a customized version of XPe. A second tool called Component Designer is used to create or modify components such as custom applications and drivers, and a third tool called Target Analyzer which analyzes target hardware and creates a descriptive file to import into Target Designer or Component Designer.

A great advantage to XPe is that it's binary compatible with the desktop version of Windows XP Professional (XP Pro), meaning you can run just about any application or driver on XPe that you can with XP Pro. XPe also allows any of the Microsoft Server applications, such as SQL Server, Exchange, Internet Information Server (IIS), to run unmodified on XPe. Unlike CE .NET, XPe runs only on x86 PC platforms and requires a PC BIOS.

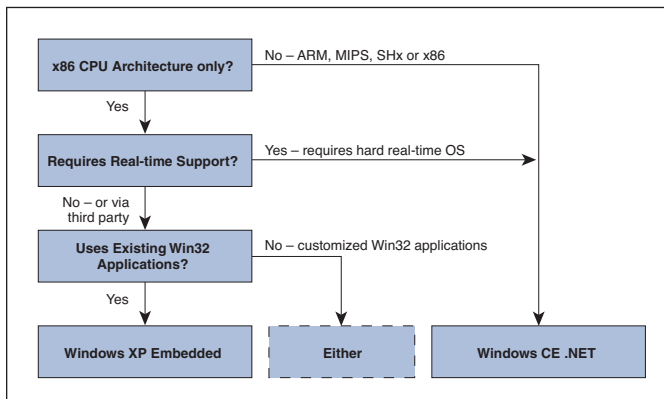
At its smallest state, XPe requires a minimum of 8MB of Flash/ROM and 12MB of RAM. Of course if you want to include network support, display capabilities or additional applications, this size can greatly increase. XPe supports the Microsoft Management Console, Microsoft SNA Server, Microsoft System Management Server, a large selection of off-the-shelf 3rd party applications, as well as the NTFS secure file system. You can use any off-the-shelf development tools for creating XPe applications as long as you have all the required OS modules included on the device to support the application in question. In addition to supporting the features of XP Pro, XPe also includes features specific to the needs of embedded development, such as flexible boot and storage options and the ability to operate without a hard drive with supported network boot capabilities.

Choosing Between Windows CE and XP Embedded

Since both operating systems have common features and strengths, it's important to know which one is best suited for the device you are creating. As mentioned, Windows CE can run on many CPUs — including very high performance RISC processors that have a lot of computing power but very low power consumption or heat dissipation — and is a natural choice for handheld battery operated devices. CE is also lower in cost than XPe in per unit licenses. XPe on the other hand is binary compatible with XP Pro, allowing you to run off-the-shelf XP 8 applications and drivers for the desktop on an XPe device. It also has all the security features (e.g. C2 rating, NTFS, domains, etc.) that XP Pro provides. Developing under the XPe OS is usually much faster than CE.

Because Windows CE and XPe are both based on the same Windows standards and can communicate well with each other, it is also possible that your company may develop a whole product line that includes both CE and XPe devices.

To help choose the best OS for your product, the remaining sections look at the important considerations to keep in mind when designing and implementing an embedded project, as well as the tradeoffs to consider when using either CE or XPe.



Developing with Windows Embedded

To properly develop Windows Embedded devices, it's important to fully understand the best way to get your project started as well as understand all of the options that this development environment offers.

Getting Started

Getting a solid start on a Windows Embedded-based project is critical. Unfortunately many companies start these projects only to get stalled for weeks on otherwise straightforward issues that could have been easily avoided with basic training and education. Here are some suggestions to help you get going quickly and productively.

Training

Training on Windows Embedded can teach you the proper use and capabilities of the OS, development tools, techniques to create drivers and other parts required for the operating system. Though training may have its initial costs, in the long run it is an important cost saver on most Windows Embedded projects. Decision makers often have inaccurate or missing information about the embedded operating systems to make crucial product decisions. By training both the technical and non-technical staff on CE, XPe or both, decision makers will have a strong knowledge base to make key decisions. The earlier problems and issues are discovered, the cheaper it is to resolve them.

For Windows Embedded training, Microsoft offers lab-based MSDN courses specifically designed for developing devices using Windows CE or XP Embedded. BSQUARE is one of

several companies which offers courses as is or if required, custom training is available to meet the specific needs of your organization. Go to www.bsquare.com/training for more information on BSQUARE training.

Researching

Keeping on top of the latest CE .NET and XPe news is a good way to learn about the opportunities of using these operating systems. There is a wide range of information on Windows Embedded available from Microsoft as well as 3rd party resources. A good starting point is www.microsoft.com/windows/embedded, or do a search on "Windows CE .NET" or "XP Embedded."

Consulting

Initial consulting from an experienced Windows Embedded system integrator can quickly assess what can and can't be done with Windows CE and XPe, and how to implement your value-add features on an embedded device. The consulting will answer your feasibility issues and establish a direction for your particular project. Since BSQUARE is one of the most experienced Windows Embedded system integrators and the only Porting Partner for Microsoft, we often help companies get started on CE or XPe. BSQUARE offers special consulting services for this initial consulting. Go to www.bsquare.com/services for more information on BSQUARE's consulting capabilities.

Don't Jump the Gun

Everyone wants to be first to market with new technologies, but be careful not to get too far ahead of the wave.

One company was eager to come out early with a VoIP phone device. They started developing it on a version of Windows CE that did not yet support VoIP. They also chose to use a new CPU that did not yet have any design wins in the category. They built their hardware based on a reference design that was not yet qualified. And they staffed for development using inexperienced contract engineers.

Both the hardware and software development experienced repeated delays. Ultimately they were unsuccessful creating their VoIP phone in the initial timeframe allocated for the product development. They ended up with a 4+-month delay waiting for CE version 4.2 to be released which included VoIP capabilities. It was an unfortunate exercise resulting in alienating suppliers and fingerprinting all around.

Investigating

After deciding on the product development details, prototyping the device can help answer key questions and substantially reduce the risk of the project, as well as wasted resources. Cost versus features, performance versus battery life, CPU selection, and memory requirements are just some factors you may want to consider in your prototype. Occasionally several (if not all) of the features you include in your device will depend on the investigation and tradeoff analysis in this stage.

Custom or Targeted Device?

With Windows Embedded devices, one of the first decisions to make is whether you will build a Microsoft targeted device or a custom device. A targeted device is based on a standard Microsoft configuration of the operating system that each

OEM must strictly follow. Examples include the Handheld PC, Pocket PC, and SmartPhone designs based on the Windows Mobile version of CE. Devices in each of these groups have similar features and share the same applications and software development kits (SDKs). In general, targeted devices are consumer oriented. OEMs will coordinate development of targeted devices directly with Microsoft so that users will have similar hardware features and applications will work across all the supported devices.

For custom devices, OEMs customize the Windows Embedded operating system to fit the specific needs of a particular application. It's the most common way OEMs develop Windows Embedded devices. By customizing the operating system, an OEM can take the features of CE or XPe and create a custom device with device specific features — such as a barcode reader, slot machine, or gas pump — while the end user may not be aware of the underlying operating system. To the user, it's just a slot machine, cash register, or gas pump. Since XPe and almost all CE devices are custom platforms, we will focus on these types of devices.

Prototype to Reduce Risks

You will want to prototype your end product in the early stage of your product development. This is where you can reduce the risk of disastrous surprises that can show up late in the project development cycle.

In this stage you will want to answer questions like:

- Can my device features be implemented successfully with the chosen OS?
- What type of performance will I need to have for the device?

- Which processor should I use?
- How can the features of the OS be used best for the device?

If you don't know the answers or how to determine them, you may want to get help from an experienced gold-level systems integrator such as BSQUARE.

Reference Boards

If you are developing a custom device, starting off with a reference board from the semiconductor company of the CPU specified will greatly assist in your development process. First, you can prototype your system on real hardware for proof-of-concept and start looking at high-risk areas instead of having to wait for your first alpha level boards from manufacturing. It will help to move development tasks earlier in the schedule instead of having to run the system software for the first time after the first alpha level boards are even produced.

Second, a reference board also allows you to port your value-added hardware components to your design and start developing your drivers in the early stages of the project. It will give your team more time to flush out design issues and bugs prior to the product shipping.

In most cases, a company's device is initially based in part on a reference board. For XPe you can use most off-the-shelf PC motherboards or SBCs (Single Board Computers) as your reference platform. It's best to evaluate performance characteristics early in your project so that you can more easily modify the design, if necessary. For CE, you first will need

Beware Feature Creep

Just because it is possible to add another capability to your device doesn't mean it's wise.

That's the lesson to be learned from one company that attempted to develop an enterprise PDA device. They got a little carried away.

The design consisted of a telephone, a personal information manager, a Wi-Fi connected web browser, media player, camera, and support for a GPS. All of this in a very small form factor with limited memory and a slow video controller.

Unfortunately, it wasn't a success in the marketplace. Users were confused by setup and configuration, as well as the marketing strategy trying to hit too many targets. In the end the device was a jack of all trades, master of none.

to port the OS to the reference board of choice. Most semiconductor companies offer “free” sample level code which supports CE and Platform Builder though BSPs (Board Support Packages). But be aware with this approach if you are going to measure performance. If you base key decisions of performance on sample or “free” code, you stand a good chance of basing decisions on bad data. The free code usually has been developed to get the selected OS up and running on the CPU manufacturer’s hardware in its simplest form and hasn’t been optimized for performance criteria such as speed and battery life. Using high performance code that has shipped on multiple products from a company like BSQUARE can help ensure your work and decisions are made on a production quality port of Windows CE.

Too often companies put together a CE platform only to later determine that it will not meet the project’s needs, or discover that the build of CE was not done properly or a driver was not optimized correctly. The problems could allow the company’s competitors to beat them to market. Due to the flexibility of Windows CE, there are some issues that need to be addressed (as there are with any operating system) before you begin developing. BSQUARE, a company that has completed hundreds of CE projects, can help you navigate these waters.

Using software and services from a company like BSQUARE allows your company to focus on value-added components that can set your device apart from those offered by competitors. BSQUARE has board support packages that can bring up a reference board running Windows CE with production drivers within minutes. It also can save months of work for companies who want to do the work in-house. BSQUARE can also provide engineering services to help port Windows CE or XPe to your device.

Use the Best Processor for the Product

When designing an embedded device often you will encounter a series of conflicting design requirements. This in turn requires a number of compromises. For example, you may want your device to be rugged and lightweight, feature rich yet inexpensive, or have a high-resolution display but have a small screen size. With the Windows Embedded environment, you can limit the amount of compromise needed. Platform Builder supports most processor Board Support Packages (BSPs) out-of-the-box, so quick evaluation of the processor that best supports your design requirements is quickly and easily achieved.

Supporting Different Processors

It is not uncommon for a company to support different processors on a particular product over the span of the product’s life cycle. A company might also have several products supporting the same application platform running on the best processor for each product. Using different processors could allow you to have a more integrated power-conscious processor for a handheld device or a screaming power hog CPU for a device plugged into the wall. Dealing with different processors used to be a challenge, having to compile and test for each platform. With .NET and the Compact Framework (CF) for CE .NET, this is no longer an issue. By building the CF into the ROM CE image and running a JIT (Just in Time) compiler, you can now run the same application on different CE devices as well as XPe, or even a conventional desktop Windows platform.

Developing with CE and XPE: Things You Need to Know

The following sections point out some of the key areas that should be addressed when developing a Windows Embedded product. Items such as screen, storage, localization, and quality assurance testing are a few that need to be considered. An XPe-based project differs in several ways from a CE-based project; these differences will be called out in the sections below.

Screen and Display Issues

When developing a Windows Embedded device, the following display and screen design issues are often overlooked and not only effect the way the user interfaces with the device but can also have consequences on how the platform is updated and further maintained.

If your device has a display or some type of shell or user interface, you will need an interface for users to access all of the functions of the device. There are several possibilities and though they may be fundamental in nature, each need to be considered as to what is appropriate for your device.

Desktop

A desktop resembling the standard Windows interface with a Start button and task bar, is the most familiar interface. Desktops are appropriate for devices used as general-purpose computers. If your device performs a dedicated purpose (e.g. barcode reader, point of sale terminal, vending machine) you probably do not want to include a desktop interface that accesses the whole device.

Single User Application as Shell

A single application can be used as a simple shell. In this situation, you would create an application that would always be the interface to the device. Your application would also need to provide the functions allowing the device to be configured and maintained (e.g. security, network, volume). Sometimes access to these functions is hidden behind graphics, or is called with specific three-key calls.

Command Line

With a command line interface, you provide a non-graphical desktop though which you can access Windows applications. It's similar to the old DOS command prompt interface and the cmd.exe from the Run dialog in XPe or CE. With this option, the end user needs to know the specific command calls as well as a requirement for a keyboard or other text-based input method.

Custom Interface

Custom interfaces access functions specific to the type of device you are building and are the most common for embedded products with a display. For example, a barcode may include a custom interface with a Scan button for operating the laser scanner and soft alphanumeric keys for entering data. The custom interface is also usually a separate application that provides access to other programs and device features. It provides maximum flexibility by allowing the shell to be modified while not requiring changes to an application.

Custom interfaces are often the best solution, not only for end user interaction with the device, but also added flexibility for remotely managing the interface as well as the end image size. Unfortunately these are also the most complex to implement. With over many years of developing custom interfaces for customers, BSQUARE has developed a proprietary browser-based shell design technology based on ActiveX controls and HTML scripting. This makes it substantially easier to deliver high quality GUI designs for its customers in a short time-frame.

Browser Support

Both Windows CE and XPe support Internet browsers. In a majority of the new smart devices (e.g. Internet appliances, set-top boxes, kiosks) the browser is used as a key component (if not the only component) in the device interface.

Windows CE .NET 4.2 as well as XPe include an IE 6.0 browser control. Also supported is Pocket Internet Explorer (PIE) which includes similar features but is smaller in size and optimized for mobile handheld devices. Be aware that the browser can

also be one of the larger components for the end image. Care needs to be taken in selecting this as a user interface engine for memory and performance constrained devices.

Display Resolution

If your device includes a display, you will need to choose the correct resolution and display size to ensure users can easily view the information the device is displaying. Most embedded designs have the flexibility and requirements to support full VGA (640x480) or higher, half VGA (640x240), quarter VGA (320x240), or any multitude of custom resolutions.

If your Windows CE .NET-based device will have less than a full VGA display, you will need to ensure that the graphical elements displayed on your device will fit onto the smaller display. For instance, if you are using a quarter panel VGA display, the dialog boxes (which are fixed-sized bitmaps) might have been designed to work on a full VGA display. The result is that part of the dialog box will extend off the edge of the display where it cannot be accessed. There are different ways to solve this problem. One way is to redesign your dialog boxes and graphical elements to fit into your smaller display. Another way is to design your interface using scalable dialog boxes. Regardless of how you solve this issue, you will need to make sure your graphics driver supports the size and resolution of your display. If your driver does not know the correct display size, Windows will not function properly.

Do Your Homework

Neglecting proper research and investigation in the early stages can be hazardous to your project. A well known consumer electronics company had developed a Web Pad device in 2001-2002, in parallel to the 400MHz XScale CPU becoming available, but chose instead to stick with the less powerful 206 MHz, SA1110 processor. To save on costs, the design also allowed for only minimal amounts of RAM/FLASH memory and a slow video controller.

The challenge discovered late in the project, was an under-powered hardware design coupled with a complex graphical interface intended to give desktop PC level performance.

As a result, the GUI was painfully slow and the project suffered multiple delays in attempts to correct the poor design. In the end, with over \$1 million invested, the project was cancelled.

Screen Orientation

If your device uses different screen orientations — such as portrait or landscape — you will want to ensure your applications display the correct orientation as well. The two most common ways to solve this problem are with a “screen rotation,” which automatically rotates the screen depending on what the device is displaying. You can make the rotation with software (called “software rotation”) or hardware (“hardware rotation”). The tradeoff between the two choices is performance versus cost. Hardware rotation is more expensive but faster, while software rotation is less expensive but slower.

Software rotation usually involves modifying the display driver to optionally rotate the screen 90, 180, 270, or 360 degrees. Software rotation allows the unit to be physically held in any orientation for the applications to be seen right side up. This is a low cost solution since it doesn't require additional hardware, but the rotation translation takes CPU cycles and affects performance. The performance degradation depends on many factors — such as resolution and what's being displayed — but can be as high as a 30 percent hit.

With hardware rotation, the rotation can be performed very fast with little or no performance degradation. The disadvantage, however, is that hardware rotation can require additional hardware and can be more costly.

Graphics Acceleration

Windows-based systems are basically graphics-based systems. Even if you are displaying text in an editor within Windows, the text is a graphic. Anything you can do to improve graphics performance on the device should help increase the speed of graphic intensive applications and the perceived speed of all windowing operations. It is important to include graphics benchmarks in any performance testing you do for your device. To enable graphics acceleration on a Windows Embedded device, an accelerated graphics driver as well as a separate controller can be used to control the graphics acceleration. This is especially true for SOC (System on Chip) designs when the selected video resolution is higher than QVGA. Most internal controllers are optimized for smaller screens, thus larger screens running directly from the SOC can exacerbate this potential problem.

Storage Issues

On a Windows Embedded device, as with any software-based system, storage must be provided for the OS image, drivers, applications, databases, and a bootloader or BIOS. A unique feature of Windows Embedded is the flexibility in storage options.

Image Storage

For both XPe and CE supported devices, you need to provide storage for the actual OS image and components. While it is possible to use RAM to store an image as well as run software, for embedded systems we will assume that the storage is in persistent memory such as ROM or Flash memory. Since XPe runs on standard PC hardware, an NTFS file system is usually used to store the OS and applications on standard hard drives as well as Flash memory. The ability to remotely boot the OS over a network is also supported with XPe, but be aware of the required time it may take to load a 100+ MB image and supported applications. Windows CE has much more flexibility and can run on a wide variety of hardware and bootable selections. The CE image can reside in either a linear Flash array such as a DiskOnChip or NAND type Flash, or run on NOR and support the ability to XIP (Execute in Place).

For XPe, a barebones non-networked, command shell interface (no display, keyboard, or mouse) version will require approximately 12MB RAM and 8MB of disk space. For Windows CE, a barebones installation is approximately 256K ROM and 40KB RAM.

As mentioned, Windows CE allows an image to be executed in place (XIP), which means the OS image is run directly out of ROM, and not shadowed copied into RAM first. XIP requires ROM or linear Flash such as a Linear Flash Card or a Resident Flash Array. DiskOnChip, CompactFlash or other Flash media such as NAND or any type that uses a Flash file system cannot be used for XIP, and must be copied to RAM to execute.

During development you will need to run a debug version of the OS and supported components. In this instance you will want your development hardware to have enough RAM and disk space to store debug versions of the image (typically twice the size of the retail version). Of course you only need the extra storage during the development phase of the project. It can be removed on your production hardware to save costs. In addition, you will often use a separate EEPROM to store the BIOS or bootloader for your system.

You may also want additional applications and run-times on your system. For example, support for a browser, 3rd party applications such as Flash or a Java Virtual Machine (JVM) may be necessary. For these types of applications you will want to do a careful analysis of your memory requirements before freezing your hardware design as these can take substantial amounts of memory to execute and store. Also code density varies depending on the CPU type, so you may want to plan on having extra room to allow upgrades and enhancements to the device.

Registry Storage

The Windows CE registry database, which is similar to the registry in Windows 98 or Windows 2000, can be stored on the device in two different ways. Neither the user nor the applications will be aware of which method has been implemented, but as the developer you need to decide which option best suits your device.

The first option is to use a RAM-based registry in the part of the memory called the object store. Due to high Flash memory costs of early CE devices as well as challenges in writing to Flash, this method was common on some of the first versions of Windows CE. This is a good model for devices with backup batteries that are often warm-booted but rarely or never cold-booted. It provides a fast and efficient boot because the registry information always remains current in RAM and ready for users to pick up right where they left off. For devices that cold boot more often, this option is less efficient. Either the registry will not be persistent, meaning the device always boots to its initial state, or the developer needs to flush the memory to a persistent storage device whenever the registry is changed. You can also implement kernel-level calls so that the saved registry is loaded at boot time, but this method can become very slow for large registries.

There is now a second option, known as a hive-based registry, intended for devices that cold boot regularly but seldom warm boot. The hive-based registry stores registry data inside special files called hives. Because the registry files are kept on a persistent file system rather than in RAM they do not need to be backed-up and restored, making the cold boot process cleaner and faster. (The trade-off is that warm booting is less efficient because the registry needs to be read again from the file system rather than remaining in RAM.) Using a hive-based registry is also a good choice for devices that are intended for multiple users, because in addition to system hives that always load, there are also individual user hives that load only when a particular person logs on. Care must be taken with applications that require frequent registry updates, as this may impose potential performance challenges as the Flash memory is being updated.

International Support

Both Windows CE and XPe support internationalization. Unicode is a 16-bit codeset that allows characters from multiple countries to be represented within a supported operating system. Both operating systems include support for the NLS API (National Language Support API), Input Method Editor (IME), and a Soft Input Panel.

Unicode is the native code set of XPe. The Win32 subsystem provides both ANSI and Unicode support. Character strings in the system, which also includes file, path, and directory names, are represented with 16-bit Unicode characters. The Win32 subsystem converts any ANSI characters it receives into Unicode strings before manipulating them. It then converts them back to ANSI, if necessary, upon exit from the system. Windows CE is also Unicode-based, but unlike XPe, CE only supports Unicode strings. CE includes international support for many locales and the user can localize for other locales as well.

Check out the Microsoft Website at www.microsoft.com/windows/embedded for a list of the locales supported in Windows CE and XPe.

Quality Assurance

A very strong Quality Assurance program and Test Plan are essential for any embedded device project. Because both XPe and CE allow you to pick which components to include in your device, the devices can be harder to test than a standard Windows 2000/XP- or Windows 98-based system. With a Windows CE-based product, the actual CPU and hardware components can vary from creating a completely unique device. For XPe, even if a subset of components is selected, the hardware it will run on is still a standard x86 PC-based platform and needs to be tested as a complete system to make sure any removed components don't interfere with the selected HW and supported applications.

Since the CPU type and OS components can vary on a CE-based device, you need an automated testing tool designed specifically for CE. This will allow developers to unit test added or modified code to ensure they haven't created new bugs or broken any existing code. Developers can then hand the tested code off to be integrated. This pre-testing ensures a much shorter integration phase. Automated testing can then be used on the whole system during the integration stage. When new bug fixes are provided, the whole system can be re-tested to ensure the bug fix works and ensure that the fix hasn't broken anything else. At the end of integration, the code will already be well tested, providing less chance of any major bugs being discovered late in the development cycle which can potentially cause huge delays. The automated tests also allow the QA group to focus more on testing value-added components and write their own tests to add to the test tool.

Manufacturing can also use the automated test tool to check the units coming off the production line to ensure that the hardware is working and that all the software components

have been loaded correctly. This will help reduce the Dead on Arrival (DOA) units that must be dealt with in the field.

Here are some key facts to consider when thinking about an automated test plan for your company:

1. Bugs found prior to the product shipping are much less costly to fix. Bugs found after the product has shipped can require sending a service person or returning the unit to the factory, and the bug will probably be in all the devices that were shipped.
2. If bugs are discovered, typically your best people are needed to resolve them as quickly as possible. It pulls your best people off of projects and affects future product schedules.
3. Bugs mar the reputation of your company and product lines. The bad name will cost you more than initial lost sales.

Some companies choose to outsource the specialized work of QA testing to an experienced team like BSQUARE. At BSQUARE the testing is done using two different testing tools. We use the CE Test Kit (CETK) that is included as part of the Platform Builder package. We also use our own proprietary QA automation tool called CEValidator. CEValidator includes over 9,500 test cases to verify functionality in 26 functional areas. The tool was developed over 8 years, using all the bugs discovered in more than 200 CE projects. The tool includes five system stress-testing methods and new tests can be easily added.

Tools Requirements

To port XPe or Windows CE to your hardware and create the specific applications for the device, a set of development tools are required. In this section, we will describe the tools and break up the software development task into two components:

1. Porting the OS to the device
2. Developing applications for the device

Porting the OS to the Device

The process of porting the OS to a device differs depending on which operating system and device you use. For Windows CE, you need to determine which components of the operating system are required for your device and then port the CE kernel to your device hardware. You can do so by developing routines that are specific to your hardware and linking them with the Microsoft provided kernel components (which will call your hardware specific routines) to create the final CE kernel.

Next, develop all of the drivers needed to access the hardware on your device (e.g. video, Ethernet, serial, parallel, custom peripherals, etc). You can then create an "nk.bin" (a CE binary image) that contains the CE OS components, device drivers, and configuration files as well as any applications you want shipped with your device. This image is then downloaded to your device. For the above tasks you will use the development tool, Platform Builder. After you have tested your port (often called an "adaptation"), and are comfortable with the build, you can use Platform Builder to create a custom SDK (Software Development Kit) for your device. This SDK is used to then write application specific for your device. It will include information about the unique features your device has and the operating system components you selected.

Now you have a working CE device. The next step is to create applications for it. For this part, your software developers will have several choices, which are discussed in the next section. The two main approaches are to write managed applications using Visual Studio.NET, or you can use eMbedded Visual C++ to write applications that run natively on Windows CE.

For applications that are time critical or have tight performance criteria, eMbedded Visual C++ (eVC) will be the tool used. eVC was created specifically for Windows CE development and ties in to Platform Builder. Once you have loaded eMbedded Visual C++, import the SDK that was created using Platform Builder as mentioned above. This will allow software developers, who might not be aware of all components in your device (or which components were left out), to create applications that will work on your device. Without the SDK, the application developers might try to use a function or feature in their application that is not included on your device.

For XPe, the situation is a little different. The two main tools for creating Xpebased devices are Target Designer and Component Designer. Target Designer is used to select among available XPe components. Component Designer is used to create components of your own. If you have written an XP Pro desktop application, for example, and you want to make it run on XPe, use Component Designer to turn your application into an embedded component of the device. Since XPe is binary compatible with the desktop (meaning desktop XP applications and drivers can run unmodified on a XPe device) the only porting you need to do is to create new drivers for value-added components. It is also important to remember that if you do not include all of the components, an off-the-shelf XP application may not run properly since it may require one of the components you left out of your system.

Developing Applications for the Device

In addition to what's mentioned above, there are many ways to develop an application for your device. Four common strategies are:

1. Create the application from scratch for either CE or XPe:
Use embedded C++ to create a native Windows CE application, or use one of the components of Visual Studio or 3rd party development environments to create an application for XPe. Either application would need to be ported to the other platform if you wanted to switch operating systems.
2. Port the application from a desktop Windows or non-Windows version: Basically the same model as above except starting with some existing code and application logic.
3. Develop to the lowest common denominator for Windows: Develop your application using only those APIs and components available on both CE and XP. You could even code to APIs that are included in CE, XP, Windows 98 and Windows 2000. There will be other differences in the code that you can handle using environment variables and pre-processor compiler commands. This strategy allows a single source code base to be used for all the Windows operating systems, and porting consists of simply recompiling. This method, however, can get complicated because the APIs and features are different between the OSs. Unless you avoid features that are not supported equally on all the OSs, you will need to degrade functionality on the operating systems that don't have that feature.
4. Create a .NET managed application: Using Visual Studio .NET you can create .NET applications for Windows CE and XPe. Working in the same application environment that you use for developing applications for the desktop and Windows XP, you use the Smart Device Programmability plug-in that ships in Visual Studio to write your applications to the Microsoft .NET Compact Framework for Windows CE.

Information about BSQUARE

Since 1994, BSQUARE has helped world-class manufacturers and integrators with the building blocks necessary to design, develop, and test innovative smart device systems quickly and cost effectively. We offer our customers deep expertise in the latest hardware and software, providing critical engineering services that have resulted in the successful launch of hundreds of new products and applications. As a Gold-level member of the Windows Embedded Partner Program, BSQUARE has been awarded the Windows Embedded Systems Integrator and Distributor of the Year awards by Microsoft.

Take the Next Step: Contact BSQUARE Today

To find out more about Windows Embedded, as well as BSQUARE professional engineering services, Windows Embedded licensing and development tools that can accelerate your time to market, contact BSQUARE at 1-888-820-4500 or email sales@bsquare.com.



For more information, please visit www.bsquare.com

About BSQUARE:

BSQUARE is a solution provider to the global embedded device community. Our teams collaborate with OEMs at any stage in their device development to bring quality products to market faster. Since 1994, BSQUARE has been a trusted partner to smart device makers worldwide.

©2006 BSQUARE Corporation. BSQUARE is a registered trademark of BSQUARE Corporation. All other names, product names and tradenames are registered trademarks of their respective holders. CS-WINDOWS_EMBEDDED-2003, Rev2006-10

By email at sales@bsquare.com

BSQUARE Headquarters
110 110th Ave NE Suite 200
Bellevue, WA 98004, USA
Phone: 888-820-4500 or
direct at 425-519-5900

BSQUARE San Diego
6450 Lusk Blvd. Suite E210
San Diego, CA 92121, USA
Phone: 858-535-9265

BSQUARE Akron
3480 West Market Street
Suite 105
Fairlawn, OH 44333, USA
Phone: 330-864-2300

International:

BSQUARE Vancouver
3751 Shell Road Suite 100
Richmond, BC V6X 2W2, Canada
Phone: 425 519 5900

BSQUARE Taiwan
18F-B, No. 89
Songren Road, Xinyi District
Taipei City 110, Taiwan
Phone: +886-2-8780-9100