

# Windows CE Displaces Proprietary RTOSs in Embedded Systems Applications

*Though originally developed for hand-held PCs, Windows CE provides an attractive alternative to proprietary RTOSs for many embedded applications. This article will discuss the advantages of using Windows CE as an embedded operating system, with particular emphasis given to the development environment, Win32 API, and run-time environment. The article will also discuss the porting and device driver development issues faced by embedded Windows CE developers.*

A sea change is about to occur in the way that embedded systems software is developed and deployed. No longer will embedded systems designers have to master proprietary real-time operating system APIs and development tools. Instead, Windows CE will assert itself as the dominant embedded operating system, displacing proprietary RTOSs in all but the most resource-constrained hard-real-time applications. And embedded systems designers will gain access to a familiar API and development environment that greatly reduces time-to-market and development costs.

Windows CE doesn't yet offer the hard real-time response of some real-time operating systems. But it does offer significantly lower latency to interrupts than off-the-shelf Windows NT and 95. As such, designers will find Windows CE an excellent alternative to proprietary RTOSs for a broad range of embedded applications with soft real-time requirements, including medical instrumentation, industrial automation, test and measurement, and point of sale.

Moreover, Microsoft has announced plans to equip Windows CE 3.0 with hard real-time capabilities that include faster, more deterministic interrupt processing, additional priority levels, and more versatile inter-process communications facilities. These enhancements, together with the broad appeal of the standard Windows CE development environment, will relegate proprietary RTOSs to niche applications that demand ultra-low cost, miniature size, or ultra-fast hard-real-time response.

Because Windows CE was originally developed for hand held PCs, the infrastructure needed to support general-purpose embedded systems development was initially slow to materialize. However, early hurdles such as the lack of a port to the popular x86 and Pentium architectures, a limited selection of low-cost embedded development and target platforms, and minimal support for custom target platform development, have been overcome.

Today, Windows CE developers can choose from a wide range of off-the-shelf embedded PC target platforms with ready-to-run images that are certified to run

Windows CE. And designers who want to create custom configurations can utilize off-the-shelf software such as Annasoft Systems' Jump Start Kit, which greatly simplifies Windows CE platform adaptation.

## THE WINDOWS CE DEVELOPMENT ENVIRONMENT

The principal advantage of using Windows CE for embedded applications is that it supports the same Interactive Development Environment (IDE) and programming model used to develop software for Windows 95 and Windows NT. It also uses the standard Win32 virtual memory, process and thread model. This gives the more than 500,000 Win32 programmers easy entry into the world of embedded systems design without having to learn proprietary APIs and development environments.

Windows CE applications are developed on a Windows NT host using the Windows CE Embedded Toolkit (ETK) for Visual C++ 5.0, an add-on to the Visual C++ toolkit used to develop software for Windows 95 and Windows NT. The ETK provides everything needed to customize not only Windows CE applications, but the Windows CE operating system itself. Included are cross compilers and assemblers, a remote source-level debugger, a ROM image builder, an OS loader, and sample drivers. A binary version of Windows CE is also provided for designers who want to prototype and execute their Windows CE application on the host PC before deploying it on the target system.

Third party Windows CE tools like Annasoft's Intrinsic Integration Expert (IX) for CE (Figure 1) build upon the baseline capabilities provided in the ETK to further simplify and speed CE development. For example, IX replaces the ETK's command-line user interface with an intuitive graphical user interface. In addition, IX provides a variety of analysis, optimization and integration tools not available in the ETK. Among these are:

- **Modelling.** IX provides a visual decomposition of the target, including applications and operating system software. The display shows the relationships and dependencies between various system objects, giving developers a comprehensive view of

# WINDOWS CE

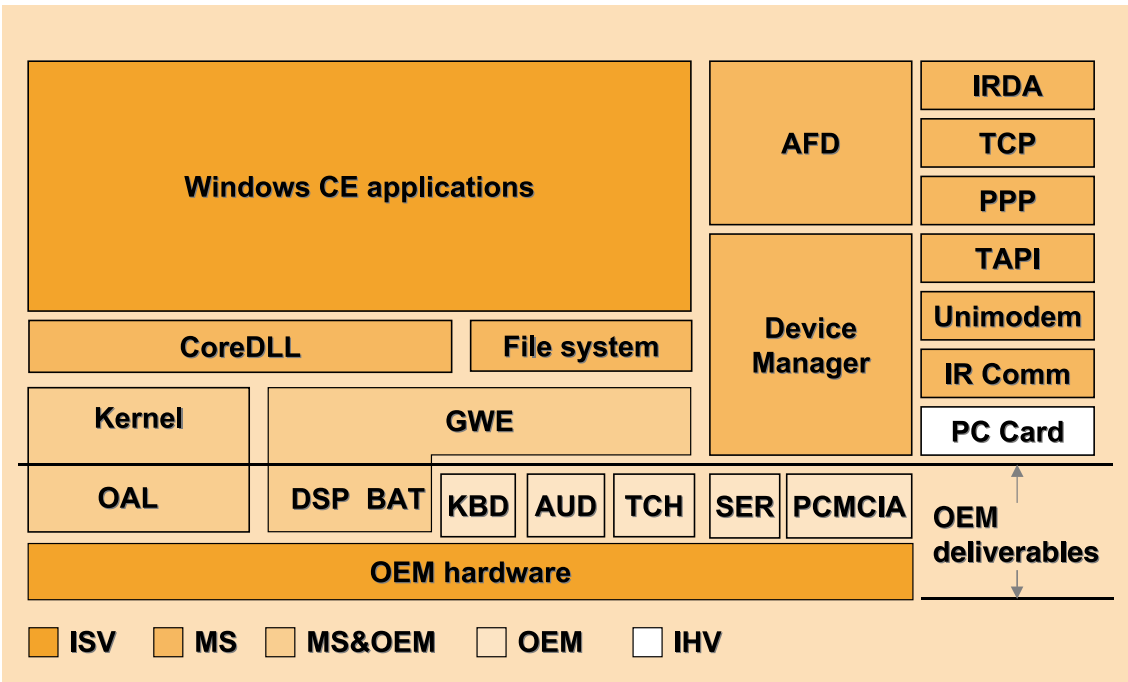


Figure 1. Intrinsyc's Integration Expert augments the baseline capabilities provided by Microsoft's Windows CE Embedded Tool Kit (ETK) Modeled after the Microsoft Developer's Studio, the ETK combines an intuitive GUI with automated build and analysis tools that greatly simplify Windows CE development, configuration, and automation.

the software to be included in their embedded system.

- **Parallel Development Model.** Unlike the ETK, which requires that the CE configuration be defined before commencing application development, IX enables application development to proceed in advance of OS configuration. This enables designers to delay OS configuration until they have a better feel for OS requirements.
- **Advanced Windows CE Configuration Tools.** IX provides visual tools that make it easy to set up ETK projects, configure Windows CE, manage dependencies between objects, configure the contents of ROM, and configure the registry, database and file system initialization files.
- **Windows CE Component Optimization.** IX provides analysis tools that makes it easy for designers to minimize their ROM and RAM requirements by eliminating duplicate and unnecessary components. The tools include facilities for DLL import, run-time DLL import, file access, boot-time configuration, memory usage analysis, registry checkpoints, and registry change.
- **Component Integration.** The IX Component Gallery allows users to visually select and integrate Microsoft and third-party operating systems and application components from a local network-based gallery of components. Pre-built Windows CE images for specific embedded PC platforms, complete with drivers, are also available.
- **Input, Analysis, and Optimization Tools.** IX lets designers import and edit a wide variety of data types. It also provides application, operating system, and dependency analysis tools to populate and manipulate the data model.

- **Integrated Build Environment.** Based on the configuration selected by the developer, IX runs the batch files provided with the ETK to build the system.

## EVALUATING WINDOWS CE FOR EMBEDDED APPLICATIONS

The standard development environment, API, and extensive third-party support will most likely attract designers who are inclined to use Windows CE in place of a proprietary RTOS. But before they commit to Windows CE, they'll need to make sure that the underlying Windows CE kernel is right for their application. In particular, they'll need to determine whether the Win32 API provides the real-time multitasking services required by their application; whether the run-time environment meets their size and performance constraints; and how difficult it will be to adapt Windows CE for their target platform.

Windows CE services are accessed through the standard Win32 API, which is generally a subset of the same Win32 API used by Windows NT and Windows 95. The Win32 API provides all the key pre-emptive multitasking and interrupt processing facilities needed to build embedded real-time applications. Included are services for creating and destroying threads, setting thread priorities, and opening and closing streaming I/O channels. The Win32 API also provides versatile interprocess communications and synchronization mechanisms such as semaphores, mutexes, and events: a notification API that makes it possible to handle user or application notification events (such as timer events) at the operating system level (rather than in a running application); and a built-in touch-screen API.

One of the first things that designers must consider when evaluating the Windows CE run-time environment for an embedded application is whether their target system provides adequate memory resources to accommodate Windows CE plus their application. Like most proprietary RTOSs, Windows CE is fully ROMable, requiring a minimum of 512 Kbytes of ROM and 256 Kbytes of RAM. Windows CE programs can execute in place out of ROM or RAM. Windows CE also implements demand paging for running applications stored in compressed form or on media that does not support execution in place.

Though not small by RTOS standards, the Windows CE footprint is quite compact considering that it includes a baseline set of drivers. The 10- to 20-kbyte minimum footprint touted for many proprietary RTOSs includes only the kernel. When equipped with a comparable set of drivers, proprietary RTOSs lose their size advantage. In a set-top box application, for example, the drivers required for a GUI and networking interface quickly negate the incremental size advantage that a proprietary RTOS might enjoy at the kernel level.

## MODULAR AND SCALEABLE

To best accommodate embedded applications with tight memory constraints, Windows CE is shipped as a set of modules and components that implement various operating system services. By incorporating only those Windows CE modules that are required by their application and target environment, designers are free to create custom OS configurations that take full advantage of the available memory resources.

The Windows CE kernel module provides the base operating system functionality, including virtual memory management, scheduling, multitasking, process management, exception handling, and certain required file management functions. Most of these kernel components are required for all Windows CE configurations. But some are optional and need only be included to support other operating system features and corresponding modules.

In addition to the kernel, the Windows CE operating system provides a number of other componentizable modules that can be customized in full or part. These modules implement functions such as the Windows CE file system, Graphics Device Interface (GDI), window and event managers, dialogs, controls, communications facilities, and support for OLE, fonts, power management, and add-on shells.

The ETK, in conjunction with third-party tools like Annasoft's Intrinsic IX for CE, make it easy to create a custom Windows CE configuration. IX's familiar Microsoft Developer Studio-like GUI, for example, lets designers set up the target display, add and remove system fonts, configure network support, and add device drivers. Designers can also use analysis functions that let them identify dependencies, eliminate unnecessary components, and calculate the CE footprint. And once designers have developed their application, IX automatically determines the minimum components needed as well as the exact combination of files and registry entries required for the application

and copies them into the correct directories on the target media. It also generates bootable target images for deployment on disk or flash media.

## PRE-EMPTIVE MULTITASKING

Once designers are satisfied that Windows CE meets their minimum size requirements, the next step is to evaluate kernel services and performance. Windows CE is a pre-emptive multitasking operating system that supports a maximum of 32 simultaneous processes, each of which can contain an unlimited number of threads. Windows CE uses a priority-based time-slice algorithm to schedule thread execution.

Windows CE currently supports eight discrete priority levels, from 0 to 7, where 0 represents the highest priority. Levels 0 and 1 are typically used for real-time processing and device drivers; levels 2-4 for kernel threads and normal applications; and levels 5-7 for applications that can always be pre-empted by other applications. Microsoft has announced plans to support 32 priority levels in Version 3.0.

Pre-emption is based solely on the thread's priority. Threads with a higher priority are scheduled to run first. Threads of the same priority run in round-robin fashion with each thread receiving a slice of execution time. Threads at a lower priority do not run until all threads with a higher priority have finished (i.e., until they yield or are blocked). The exception is threads that run at the highest priority level (0). These threads do not time slice with other priority 0 threads. They run until they are finished.

Unlike other Microsoft operating systems, the Windows CE priorities are fixed. They can only be temporarily changed by the Windows CE kernel to avoid priority inversion, a situation in which the use of a resource by a low-priority thread delays the execution of a high-priority thread. To avoid priority inversion, Windows CE allows the lower priority thread to inherit the priority of the higher priority thread until it releases its resources.

Windows CE supports a number of thread synchronization mechanisms, including wait objects such as mutexes, named and unnamed events, and critical sections. These objects are queued by Windows CE in FIFO order by priority, with a different FIFO queue defined for each of the eight discrete priority levels. New requests from threads for these objects at a given priority level are placed at the end of that priority's list. The scheduler adjusts these queues when priority inversions occur.

## INTERRUPT PROCESSING

For real-time programmers, fast, predictable interrupt response is typically a baseline requirement for the embedded operating system. Like most RTOSs, the Windows CE kernel enhances interrupt processing flexibility and minimizes interrupt disable time by splitting interrupt handling into two distinct parts: the interrupt service routine (ISR) and the Interrupt Service Thread (IST). ISRs, triggered by hardware interrupt requests, do little more than direct the kernel to the location of the IST, which handles the bulk of the interrupt processing. Once the IST has been initiated, the

system can respond to the next interrupt.

Windows CE improves tremendously on the interrupt response performance of Windows 95 and Windows NT, and closes the gap considerably on proprietary RTOSs. For example, the interrupt latency (the time required to schedule an interrupt service routine in response to an interrupt) for Windows CE running on a 59-MHz Hitachi SH-3 is just 1.3 to 7.5 microseconds. The typical IST latency (the time required to activate an IST in response to an interrupt) is only 93-275 microseconds.

In addition to its impressive raw interrupt processing capability, Windows CE offers a much higher degree of determinism than its desktop counterparts. Determinism refers to the ability of a system to respond to an event within a bounded time window that can be calculated in advance by the programmer. This capability, which is essential for ensuring timely response for critical events, is often more important than raw interrupt response to designers of real-time systems.

In Windows 95 and Windows NT, kernel-mode drivers perform the bulk of the critical processing for interrupts. Because all drivers run at the same priority (kernel mode is the highest priority) and are scheduled on a FIFO basis, programmers have no way to move the most critical drivers to the head of the pack. They also have no way of pre-determining worst-case response time, which depends on the number of drivers that have been scheduled to run at the time an interrupt occurs.

In Windows NT, interrupts trigger ISRs, which run for a short period of time at the kernel level. ISRs, in turn, trigger Deferred Procedure Calls (DPCs), which implement

the bulk of the time-critical driver code and also run at the kernel level. DPCs, finally, launch application threads (scheduled with other Windows NT processes) that perform less critical driver functions.

The principal drawback to this approach, with respect to real-time applications, is that all DPCs run at the kernel level, where they have the same priority and run until completion on a FIFO basis.

Thus, while DPCs take precedence over application threads, they cannot preempt each other, leaving programmers no means of establishing a hierarchy of DPCs based on priority. As a result, application and driver latency can vary considerably depending on the number and type of DPCs that are scheduled at any given moment—from a millisecond when no DPCs are scheduled, to tens or even hundreds of milliseconds when DPCs for slow devices such as disk and network controllers have been scheduled.

The situation is similar in Windows 95, where events provide the functionality associated with DPCs, and interrupts and events are encapsulated as VxDs, which run at the kernel level (Ring 0).

Here again, the problem is that all events run at the same priority. Programmers can control the scheduling order somewhat.

But once events are scheduled, the order of execution cannot be changed.

In Windows CE, the ISTs that implement the bulk of device driver functionality run as pre-emptable user threads with a priority level between 0-7. When an interrupt occurs, an ISR in the device driver disables interrupts, discerns the source of the interrupt, retrieves an identifier for the interrupting device, passes the identifier

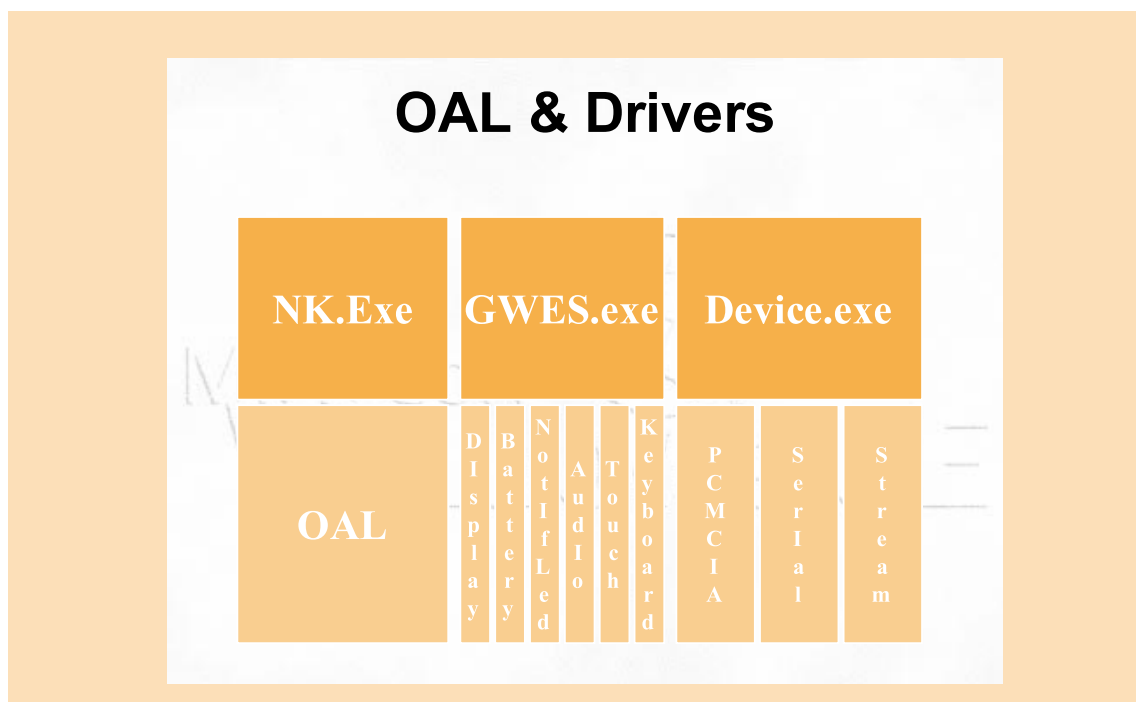


Figure 2. Annasoft's JumpStart software enables Windows CE to be booted directly from a floppy drive, hard drive, flash disk, or without the assistance DOS. This significantly reduces CE system costs by reducing space requirements by almost 200 kbytes and eliminating the need to purchase a separate DOS license.

# Ad Imagination Systems

er to the task scheduler in the kernel, and re-enables interrupts for all devices except the one that caused the current interrupt. The OS scheduler resolves the logical ID for the interrupt event and schedules the ISTs (Interrupt Service Threads) associated with that event. Because ISTs are scheduled along with application threads, and are preemptable, programmers can ensure timely access by critical drivers by assigning the appropriate priorities to their ISTs.

For designers who require even more timely, predictable real-time response, Microsoft is readying hard real-time enhancements for Windows CE Version 3.0. As mentioned earlier, scheduling latencies for ISTs will be reduced below 50 microseconds, and the number of thread priority levels will be increased from 8 to 32. In addition, Version 3.0 will add support for nested interrupts, which enable interrupts at higher priority levels to be serviced immediately from within the context of an ISR, and semaphores, which build on the Mutexes already present in Windows CE.

## BUILD OR BUY

Once designers have determined that Windows CE meets their minimum size and performance requirements, they'll need to make a build-or-buy decision. Designers who prefer an off-the-shelf Windows CE solution can choose from a variety of embedded PC platforms utilizing a variety of popular bus interfaces (i.e., ISA, PCI, and STD32) and form factors (such as PC-104 and CompactPCI). The vast majority of these are based on 486- and Pentium-compatible processors, which are the dominant 32-bit processors used in embedded systems today.

In order to run Windows CE on these platforms, designers will have to do two things: First, they'll have to license Windows CE from a Microsoft distributor such as Annasoft Systems. Second, they'll need to obtain a kit that either adapts Windows CE for that platform, or contains a build tree for a ready-to-run image.

Annasoft's Jump Start, for example, provides drivers for core PC functions such as floppy, IDE hard drive, CRT/LCD video, serial mouse, and parallel port that enable a variety of off-the-shelf embedded PCs to run Windows CE. The kit also provides software (CE Launcher) that enables 486- and Pentium-based embedded PCs to load Windows CE from a rotating or flash disk and execute it out of RAM without the assistance of MS-DOS, thereby eliminating the need to purchase a separate DOS license (Figure 2). Jump Start is also available as part of a Platform Library Kit (PLK), which enables developers to build images for a wide range of commercially available platforms without having to learn the ETK.

Designers who want to build their own Windows CE platform or modify an existing reference platform will need to obtain an ETK from a Windows CE distributor such as Annasoft Systems. The ETK includes object code for Windows CE; source code for the OAL (OEM Abstraction Layer), which is configured and recompiled for each hardware platform; reference materials and source code examples for creating platform-specific files (the equivalent of a BIOS), which interact with the

OAL in Windows CE; and tools for building a ROMable image that includes Windows CE, the OAL, platform-specific files, and the user's application.

Figure 3 depicts the overall Windows CE system architecture. Figure 4 focuses on the parts of the OS that OEMs must adapt to support new platforms—namely, the OAL and device drivers. The OAL is the clay that molds the operating system to the hardware. It contains the start-up routine needed to initialize the hardware registers, map hardware interrupts to ISRs, set up the timers, and boot the CE kernel. The OAL also contains the power management routines that suspend the processor and other hardware when they are inactive.

Once designers have configured the OAL for their particular hardware platform, they can turn their attention to device driver development, which is typically the most time-consuming aspect of the porting process. As shipped by Microsoft, Windows CE provides a limited set of drivers for devices centric to handheld computers, including touch screen, keyboard, PS-2 pointing device, flash storage device, and a low-resolution (320 x 320 pixels) VGA display controller.

To simplify the process of adding new drivers not included in the standard Windows CE package, Microsoft provides an ISR with the OAL that can be triggered on the standard IRQs used in Wintel PCs. To add support for a specific device, designers simply modify the data structures that map each interrupt to specific devices.

Once designers have developed their drivers and modified the OAL, they are ready to build an executable version of Windows CE that has been adapted for their specific hardware platform.

## IT'S GOING TO BE A CE WORLD

Adapting a custom hardware platform to run Windows CE is a straightforward process, and one that should become even easier as third party tools for embedded CE development takes root. Proprietary RTOSs will always occupy a niche in embedded applications that require miniature footprints or ultra-hard real-time response. But for embedded designers who can tolerate 100-usec interrupt response and spare a half Mbyte of memory, the widespread popularity of Microsoft's development tools and APIs make Windows CE the odds-on favorite as the dominant mainstream embedded operating system. Near-term improvements in hard real-time response should make Windows CE even more formidable. ■

---

*Dick Eppel is the President of Annasoft Systems. Prior to joining Annasoft Systems, Dick served seven years as President at I-Bus, a manufacturer of embedded PCs. Dick Eppel was also Vice President of Operations at National Advanced Systems, a distributor of IBM compatible mainframes, and President of Parallel Computers, a manufacturer of fault-tolerant minicomputers. Dick holds a BS in Mechanical Engineering from the Stevens Institute of Technology.*