What’s New in Windows Embedded Compact 2013
Windows Embedded Compact 2013 is the latest iteration of the smallest and most flexible Microsoft operating system, which has been under continuous development for more than 15 years. The previous version, Windows Embedded Compact 7, significantly upgraded the kernel to support multiple core processors and quadrupled the supported amount of RAM from a max of 512 MB to the new limit of 2 GB. In Windows Embedded Compact 2013, Microsoft developers have turned their sights on tools, with a major effort to bring Windows Embedded Compact 2013 to parity with desktop Windows. This focus has been both on platform enhancements, where the operating system is ported to new hardware, as well as application development, where applications are designed and written to run on embedded systems.

Tools
The big news with the release of Windows Embedded Compact 2013 is the move to the Microsoft Visual Studio 2012 development system for both its platform and application development environment. By aligning to Visual Studio 2012, the embedded developer can now take advantage of features in the latest Microsoft development platform. Included in this move are significantly improved compilers and substantially better runtime libraries.

The integrated development environment (IDE) for Visual Studio 2012 provides significant improvements for developers. A simplified UI and much better syntax colorization provide a cleaner environment that enables the developer to focus on the code they are writing instead of the tool they are using. Visual Studio 2012 also supports auto-generation of code snippets for standard language constructs that speed code entry. Individual tabs can be dragged to their own window outside the IDE. Sidebar windows such as the Solution Explorer now include search boxes to make it easy to locate files and functions in large projects.

The Intellisense computer program in Visual Studio 2012 is also significantly improved. In addition to being faster, the new Intellisense brings some of the features previously limited to managed development to the Microsoft Visual C++ development system environment, with better parameter information as function calls are written. The figure below illustrates the convenience of Intellisense prompting the programmer to complete the function.

```c
case WM_COMMAND:
  wmlId = LOWORD(wParam);
  wmlEvent = HIWORD(wParam);
  // Parse the menu selections:
  switch (wmlId)
  {
    case IDM_ABOUT:
      DialogBox hWnd,
      hInstance,
      lpTemplate,
      hWndParent,
      lpDialogFunc
  }
```

Notice that as the function name DialogBox is entered, Intellisense prompts the developer for the four parameters required for the function. This is old news for managed developers but now this feature is available to the native development world where most developers live. Visual Studio 2012 also brings code analysis features that examine your code and make suggestions on improvement.

Application Development
Any installation of Visual Studio 2012 can be used for embedded application development. First the Visual Studio install should be updated to at least “Update 1.” The developer can then install Application Builder for Windows Embedded Compact 2013, also known as App Builder. This installs the remote tools and low-level infrastructure necessary for connecting with the target embedded device.

After App Builder is installed, any Windows Embedded Compact software development kit (SDK) can be installed. As with earlier versions of Windows Embedded Compact, an SDK is generated using Platform Builder. The built SDK will match the specific componentization of the operating system for that device. More than one SDK can be installed in Visual Studio if the developer needs to create applications for more than one type of device.

Separating platform development such as OAL and driver writing with Application Development allows a developer to acclimate with their environment without having to be burdened with the complex tools used by the other side of the embedded development process.
After App Builder and an SDK are installed on a system with Visual Studio 2012, the New Projects dialog will have entries for Windows Embedded Compact applications under Visual C++ for native and the Microsoft Visual C# development tool for managed applications. Native support includes creating classic Win32 application programming interface applications as well as Active Template Library (ATL)–based and Microsoft Foundation Classes (MFC)–based applications. The improvements to the ATL and MFC libraries have been updated to their latest versions and are discussed later in the document.

Similar to Windows Embedded CE 6, the application development environment provides its own “corecon-based” connectivity to the device. Configuring the connection is a simple matter of entering in the IP address of the target device and starting a client program on that device. Applications are downloaded to the device over this connection. A full application debugger is available that mimics the standard Visual Studio application debugger used when developing Windows applications.
Compiler Improvements

One of the key advantages of the alignment to Visual Studio 2012 are the significantly improved compilers. The updated Visual C++ compiler supports new features of the Visual C++ 11 compiler standard. Lambda functions allow a developer to write unnamed functions that simplify code in situations such as callback functions.

Lambda expressions allow simplified declaration of functions by allowing them to be defined in line. For example, the qsort function that needs a sort function can now be written as:

```c
int arr[10] = {2,6,22,32,2,0,9,1,52,323};

// Sort the array
qsort (arr, 10, sizeof(int),
      [](const void *pa, const void *pb)->
        int {return *(int*)pa - *(int*)pb;});
```

In the code above, the double brackets [ ] signify the declaration of the anonymous lambda function that is defined in line as a parameter to the qsort function. The parameter list is defined after the brackets and the body of the function is defined after the -> symbol.

Range-based loops allow a standardized syntax for a Visual C++ application to iterate through any collection with an implicit or explicit iterator function. This replaces the < for each > construct that was a Microsoft-specific extension to the language. The new syntax is as follows:

```c
int arr[10] = {2,6,22,32,2,0,9,1,52,323};

// enumerates any collection
for (int x : arr)
{
    printf (“%d\n”, x);
}
```

The new compiler supports auto-vectorization, a technique where the compiler analyzes loops to see where vector-style opcodes can be used to perform identical operations across multiple data locations. The compiler uses SSE2 instructions on x86 CPUs and the NEON instructions on ARM processors to apply SIMD (single instruction, multiple data) opcodes to improve performance. Auto-vectorization is enabled by default in the new compiler. For specific code fragments where you many not want the compiler to do this, you can use a pragma statement to disable auto-vectorization as shown below.

```c
#pragma loop(no_vector)
for (int i = 0; i < 1000; ++i)
    A[i] = B[i] + C[i];
```

The new compiler can also generate code to use multicore CPUs. Auto-parallelization is a technique where the compiler recognizes code that can be generated across multiple threads to take advantage of multiple CPU cores. By default, the analysis is quite conservative so that the generated code does not spin up an excessive number of threads. However, it is possible to recommend to the compiler that a specific loop should be parallelized. In the code below, the pragma recommends that the compiler use eight threads to parallelize the loop because the compiler doesn’t know the typical number of iterations.

```c
void loop_test(int u)
{
    #pragma loop(hint_parallel(8))
    for (int i=0; i<u; ++i)
        A[i] = B[i] * C[i];
}
```
Improvements to the Runtime Libraries
The C Runtime (CRT) used by Windows Embedded Compact 2013 now matches the CRT used by the desktop. This commonality provides compatibility with the desktop; however, the new CRT is significantly larger than the old Compact-specific CRT provided by earlier versions of Visual Studio.

Improvements to MFC and ATL match the CRT efforts. Windows Embedded Compact 2013 uses the desktop versions of these class libraries. Like the CRT change, this improves desktop compatibility at the cost of a significantly larger runtime library. To minimize the size increase, Microsoft recommends that MFC and ATL modules statically link to their respective runtime libraries.

The Standard Template Library (STL) has also been significantly improved. In addition, the memory footprint of container classes has been noticeably reduced. For example, the footprint for a single item in the Vector container class has been reduced from 24 bytes to 12 bytes. The list, map, and multimap containers have footprint reductions as well. Each now has a per-item footprint of 8 bytes from values of 28, 32, and 32, respectively, in Visual Studio 2008. Because the STL libraries are also shared with the desktop, STL has significant improvements in functionality across the board.

All these improvements do come at an additional cost. All applications and drivers written for earlier versions of Windows Embedded Compact and Windows Embedded CE will need to be recompiled for Windows Embedded Compact 2013. This is due to a new application binary interface (ABI), which is the stack frame and calling format generated by the new compiler and the requisite changes to the operating system interfaces.

In addition, Windows Embedded Compact 2013 requires that ARM CPUs support the ARM v7 as well as Thumb2 opcode sets. Current x86 chips will be supported, although x86 applications and drivers will also need to be recompiled for Windows Embedded Compact 2013. x86 CPUs will be required to support floating point as well as SSE2 (Streaming Single Instruction Multiple Data Extensions 2) extensions to the classic x86 opcode set.

Managed Application Development
There is great news for managed application developers as well. The .NET Compact Framework runtime has been significantly upgraded to .NET Compact Framework 3.9 (previously known as version 3.5). This is the first upgrade available to embedded developers since Windows CE 6 was released in 2006. The user interface remains for the .NET Compact Framework Windows Forms to ensure backward compatibility with existing .NET Compact Framework applications.

The new runtime includes a generational garbage collector that reduces garbage collection (GC) overhead. Earlier versions of the .NET Compact Framework used a simple “mark and sweep” algorithm for garbage collection. Apps were frozen, each object in the managed heap was examined if it was “reachable” from another object, and if not, that object was added to a list of objects to be recovered. This works great for smaller applications but as the complexity and size of .NET Compact Framework applications grew, the time needed for garbage collection increased to unacceptable levels.

A generational garbage collection algorithm takes advantage of the general rule that most objects “die young”—that is, they don’t have a long life on the heap. Assuming this, the generational GC algorithm doesn’t repeatedly examine the reachability of older objects. This significantly reduces the time needed for the “Generation 0” GC scans. When necessary, a GC can scan both the Gen 0 and older Gen 1 objects to completely examine the heap, but this process is executed significantly less than a simple Gen 0 scan and is about as costly as the entire heap scan of a non-generational GC algorithm. In short, the generational garbage collection routines in the .NET Compact Framework 3.9 runtime are much less a burden on the system and managed applications.

Another valuable improvement in .NET Compact Framework 3.9 is that core framework assemblies are now shared across AppDomains. This results in smaller footprints because JITed code is shared, and also results in quicker application startup times because the second application does not need to JIT an assembly previously JITed by currently running applications. The new runtime is also multicore safe, which allows managed threads to run on different CPU cores. The new runtime also uses floating point hardware on ARM CPUs to speed floating point calculations. As a result, the new runtime is noticeably faster than the existing runtime on Windows Embedded Compact 7.
Managed embedded development in Visual Studio 2012 is implemented in a similar manner to native development. After the EmbeddedAppBuilder extensions to Visual Studio are added, the developer then installs the SDK built by Platform Builder that matches the target device. This adds a Windows Embedded Compact item in the Visual C# build templates. Under that item, Visual Studio lists all the SDKs installed on the system. Managed templates are available to build a class library, console application, device (Windows Forms) application, and an empty project for those who wish to either configure their own project or have preexisting files to add to a project.

Windows Forms projects display the Form Builder window where components can be selected from the toolbox and dropped onto the form. This, of course, is the classic method of building a managed application and will be familiar to desktop and embedded developers experienced in developing managed applications.

Platform Development
Platform developers can take advantage of Visual Studio 2012 to port the operating system to new hardware, develop drivers, and debug software remotely from their development PCs. As before, the Visual Studio IDE will host the editors, download host, and debuggers for the remote platform.

The structure of the Platform Builder menu items and dialog boxes has not changed from earlier versions of Platform Builder. The build menus, catalog window, debug windows, and target options all remain the same. The primary difference is a visual alignment to the new, modern UI of Visual Studio 2012.

The build tree in Windows Embedded Compact 2013 is also laid out in the way it has been for all earlier versions of Windows CE and Windows Embedded Compact. The root directory is now \WINCE800 with the classic platform, private, public, other, and SDK directories below the WINCE800 root. There have been some modifications to the lower levels, however.
Under the platform directory, the board support packages (BSPs) are fewer because Windows Embedded Compact 2013 only supports ARM v7 and x86 CPU architectures. BSPs that supported now obsolete CPUs are no longer included. In addition, the functionality that was in the VirtualPC BSP has been merged into the CEPC BSP. To use Windows Embedded Compact in a Virtual PC environment, simply use the CEPC BSP and use the new virtual disk that is in the \WINCE800\platform\cepc\vm directory to base your virtual machine upon.

Within the “public” side of the build tree, the number of subdirectories has been reduced. Folders such as Data Sync and remote desktop protocol (RDP) have been removed because the features that were supported by those folders have been removed from the operating system.

With the release of Windows Embedded Compact 2013, Microsoft has focused on optimizing the operating system footprint to deliver a lean, vertically targeted embedded operating system. Some features such as RDP and Data Sync have been removed from Windows Embedded Compact 2013. Other features such as the image decoders have been removed but replaced by modern, security-enhanced equivalents. In the case of the image converters they have been replaced by a port of the Windows Imaging Component from the desktop.

The componentization strategy of Windows Embedded Compact has also been rethought. Instead of hundreds of very small components, the new version of the operating system now has dozens of somewhat larger components. This reduction was accomplished by both removing components as mentioned above as well as merging other components that were typically included in an image. This simplifies testing of the operating system, which will result in a more secure and robust operating system going forward.

One change to Windows Embedded Compact 2013 is apparent when the system boots. The classic “Explorer Shell” is no longer included in Platform Builder. The new shell is actually an updated version of a very old shell that has been around since Windows CE 2.0. The MinShell, for minimum shell, provides all the support a standard shell application does but doesn’t have the “eye candy” of a Windows 98–style shell. MinShell provides a desktop window, including displaying the appropriate background image, as well as supporting the Alt+Esc key combination to bring up a Task Manager–style dialog box. This dialog box lists currently running applications and can launch new applications. This interface does not provide any navigation around the file system. The old command line interface shell CMD.EXE is also still in Windows Embedded Compact 2013. This useful application can navigate the file system to copy files and launch applications.

The source for MinShell is provided in Windows Embedded Compact 2013 and can be used as a basis for developing additional functional shells, or used as a model for the code to add to the primary application of an original equipment manufacturer (OEM) to have it support shell-like activities such as maintaining the desktop window and fielding system-level keys.

For systems that use applications based on Extensible Application Markup Language (XAML) for the user interface, the XAML-shell example still exists in Windows Embedded Compact 2013. Along with the XRShell, the XAML-based control panel applets are also included. This provides a great resource for developers to learn how to write XAML-based applications as the source for both the XAML shell and the XAML control panel applets that are provided in Platform Builder.

Another change is the removal of the Windows Internet Explorer Internet browser. However, the underlying HTML rendering control remains from earlier versions of Windows Embedded Compact. The IESample application has been removed. OEMs that need a browser can either license a browser or write their own container application that hosts the rendering engine. Because IESample is simply a host for the rendering engine, OEMs can refer to the source of IESample in previous versions of Windows Embedded Compact.

The same concept has also been applied to the multimedia area. While the media player application has been removed from this edition of the operating system, the underlying audio, video, and imaging technology remains. Microsoft DirectX also remains, along with the accompanying codecs.

One Internet-related set of features that has not been reduced is in the area of servers. The HTML server remains as a component in the operating system. The file transfer protocol (FTP) and telnet servers also remain. However, the FTP and telnet servers are, as in previous editions of the operating system, only engineering samples. They are not written to be secure, production-quality code that can ship with an embedded system. Instead, the FTP and telnet servers should only be used in a lab environment. If FTP or telnet functionality is required in a shipping product, these servers should be heavily modified by the OEM to meet modern standards.

What’s New in Windows Embedded Compact 2013
Operating System Improvements
While most of the work of this new release has gone into the new tools suite and porting to Visual Studio 2012, the core operating system has also been improved. Performance, memory management, and networking improvements have been added to Windows Embedded Compact 2013.

Improved Performance
With this release, Microsoft made a detailed examination of performance bottlenecks in the operating system. Deep, end-to-end analysis has been performed on the file system stack to increase file read performance dramatically. These file system improvements result in much faster launch times for applications, especially managed applications that can tax a file system. Of course, these file system improvements also help standard data reads and writes.

Snapshot Boot
Devices running Windows Embedded Compact have always started quickly, but with Windows Embedded Compact 2013, systems can start even faster. The new version supports "snapshot boot," which pulls a complete image of system RAM off a storage device and resumes from that image. Snapshot boot has benefits beyond starting the operating system. The restored RAM image can also contain applications, allowing for faster launch times. With snapshot boot, all the needed applications can be captured in a running state and resumed when the operating system restarts.

Snapshot boot works by initially starting up the system to a known state. An application then creates a “snapshot” of the RAM image and stores this image in a persistent store. Then, when the system starts, it can use the saved image instead of starting from scratch. Applications do not need to be specially designed for snapshot boot. Drivers will need some minor tweaking to support the driver entry calls xxxPowerDown and xxxPowerUp. These entry points have been around since the original version of Windows CE but are typically only used on systems that suspend and resume. Snapshot is similar to a suspend/resume cycle except the RAM is restored from the snapshot image instead of being maintained from a previous suspend.

Improved XAML Support
Windows Embedded Compact was the first Microsoft operating system to support an XAML user interface with native code. This unique combination joins the flexibility and power of an XAML-based user interface with the speed and low-level access of native code, a perfect combination for embedded development.

XAML support continues and is improved in Windows Embedded Compact 2013. The improved support includes much better support for data binding. UI-to-UI data binding is now supported. DataContexts can now be defined in XAML. New Blend triggers are also supported, including ChangePropertyAction, ControlStoryboardAction, GoToStateAction, and RemoveElementAction.

Networking Improvements
Windows Embedded Compact 2013 has upgraded support for IPv6, a requirement in today’s IP address–challenged world. Windows Embedded Compact 2013 fully supports IPv6 Dynamic Host Configuration Protocol (DHCP) as well as other improvements to earn the operating system the “Ready for IPv6” logo.

With this new release of Windows Embedded Compact 2013, Microsoft also completes the transition to Network Driver Interface Specification (NDIS) 6.0. The NDIS 6 standard provides improved performance while also providing consistency with the desktop NDIS drivers. Network drivers from previous versions of Windows CE and Windows Embedded Compact will need to be ported to NDIS 6 to run. Microsoft will be providing a porting document to assist developers with this transition.

This new version of Windows Embedded Compact brings a welcome upgrade of the development tools into parity with the desktop development environment. This change, along with the performance improvements and feature enhancements, make Windows Embedded Compact 2013 a significant release in the continual development of the most flexible Microsoft operating system.

© 2013 Microsoft. All rights reserved. This document is provided “as is.” Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.