Technical Article

Overview of Windows Embedded Versions

Summary: Original Equipment Manufacturers (OEMs) have several embedded options, and they may be unsure of which version of Windows Embedded best meets their needs. To help guide OEMs in their selection process, we have created this introductory guide to Windows Embedded. In this document, we discuss the various versions of Windows Embedded. We also describe their features, provide comparative scenarios, and discuss approaches to prototyping.

Published: October 2015

Applies to: All currently supported Windows Embedded operating systems

Table of Contents

Overview

Microsoft's Windows Embedded family of operating systems, tools, and services help enterprises leverage customized intelligent system solutions to gather, store, and process data. All the Windows Embedded versions we will be discussing in this paper are either fully equivalent or a subset of the functionality that shipped in the desktop version of the OS. Therefore, your current knowledge of Windows XP, Windows 7, or Windows 8/8.1 will continue to be of use.

An OEM is an individual or company that selects one of the products mentioned in this paper. The OEM customizes and configures a unique software image based on that product. Then the OEM combines that image with hardware to produce a task-specific or industry-specific solution that is sold to customers. The OEM must have a license with Microsoft for each product and pay a run-time license fee for each device sold to its customers.

In Scope

In this document, we will discuss the following:

- All the currently supported versions of Windows Embedded.
- Embedded Enabling Features (EEFs) or Lockdown functionalities.
- Comparative scenarios.
- Approaches to consider when beginning to prototype and develop your own embedded system design.
- Common usage scenarios for a few categories of devices.
- Features that can be of benefit in each usage scenario.

Out of Scope

This paper is not a replacement for any existing product documentation or training that is currently available on MSDN for Windows Embedded. The top-level documentation that discusses all of Windows Embedded can be found at http://go.microsoft.com/fwlink/?LinkID=665697.

We will not discuss the following forms of Windows Embedded in depth:

- Windows CE/Windows Embedded Compact.
- .NET Micro Framework.
- Windows 10.
- Windows NT Embedded.
- Windows for Embedded Systems.

In the following sections, we provide brief descriptions of these technologies so that you are aware they exist and know when they are used.

Windows CE/Windows Embedded Compact

For those not familiar with it, Windows CE or as it's now known Windows Embedded Compact (now shipping as Windows Embedded Compact 7 and Windows Embedded Compact 2013) is a componentized, real-time operating system. It is designed for small footprint devices at the edge of enterprise networks, and it is supported on x86 and ARM architectures. To learn more, go to http://go.microsoft.com/fwlink/?LinkId=669699.

Windows Embedded Handheld

This is a product that is based on a version of Windows Mobile or Windows Phone. It generally includes Bar-Code Scanner support and other hardware, such as Mag-Stripe readers. The devices are usually more ruggedized than a standard Windows phone and are geared towards use in retail, industrial, and warehouse environments. To learn more, go to http://go.microsoft.com/fwlink/?LinkID=671702&clcid=0x409.

.NET Micro Framework

The .NET Micro Framework is designed to take up even less space than the Windows CE/WEC operating systems. The .NET Micro Framework can even run where no operating system is present. For new information on its current development, please go to the blog located at http://go.microsoft.com/fwlink/?LinkID=671704&clcid=0x409.

The previous version of the framework is currently released as-is, and you can find discussions about it by going to the page at http://go.microsoft.com/fwlink/?LinkId=671705.

Windows NT Embedded

There was an early, released version of Windows Embedded that was based on Windows NT. However, because support for that product has reached end of life, we will not discuss it here.

Windows 'For Embedded Systems'

Later, we will briefly discuss server and desktop versions that are marked as 'For Embedded Systems' (FES) with regard to Windows Embedded.

Windows Embedded Versions

In this section, we describe each of the Windows Embedded versions that have been released. We also identify the original desktop operating system on which they are based. For most of these, you will see that there are generally two categories of product that have been released.

Windows Embedded Standard (WES)/XP Embedded – These versions are designed to provide a very high level of componentization. This allows you to pick exactly the components that your application or device might require and leave out everything else. This means that you can build a very small image or build an image that is almost the same size as a desktop installation. With the exception of Windows Embedded 8 Standard, images built on these products generally do not require any form of activation. These products also included a variety of EEFs and Lockdown features that we will discuss later.

POSReady/Windows Embedded 8.1 Industry/Windows Embedded for Point of Service (WePOS 1.1) – These versions are generally based on the nearly complete version of the desktop equivalent. They are not as configurable as the WES versions. Often, they include a subset of EEFs or Lockdown features. They are generally intended to be used on Point of Service (POS) devices, such as a cash register in a retail store.

Important These embedded versions of Windows are not intended to serve as a full replacement for the desktop versions of the OS. Often, the runtime license or End-User License Agreement (EULA) prohibits users from running certain applications locally on a Windows Embedded device. These might include things like Microsoft Office or Visual Studio. ***Please review the EULA and licensing terms specific to the version you plan to use before deployment***.

Table 1 identifies the base OS on which the embedded version is based.

BASE OS VERSION	WES VERSION	POSREADY VERSION
WINDOWS XP PRO	Windows XP Embedded	Windows Embedded POSReady 1.1
WINDOWS XP PRO	Windows Embedded Standard 2009	Windows Embedded POSReady 2009
WINDOWS 7 PRO/ENT	Windows Embedded Standard 7	Windows Embedded POSReady 7
WINDOWS 8 PRO/ENT	Windows Embedded 8 Standard*	Windows Embedded 8 Industry*
WINDOWS 8.1 PRO/ENT	N/A	Windows Embedded 8.1 Industry*
WINDOWS 10 ENT	N/A	Windows 10 Enterprise / Windows 10 IoT Enterprise

Table 1: Base OS for Windows Embedded OSs

*These versions require runtime activation.

Windows XP-Based OS

There are currently four released products that are supported and all are based on Windows XP Professional. Two of the original versions were based on Windows XP Service Pack 1 (SP1) or Service Pack 2 (SP2). They were known as Windows XP Embedded and Windows Embedded for Point of Service.

These two products were very popular. However, upgrading them fully to Service Pack 3 would be complex. Therefore, the product team decided to release two new products in 2009. These new versions were known as Windows Embedded Standard 2009 and Windows Embedded POSReady 2009. Both were based on Windows XP Service Pack 3 (SP3). Because they were new products, they had the advantage of starting a new support lifecycle.

“Microsoft has ended support for Windows XP, correct?” Yes. Microsoft did end support for Windows XP as a desktop OS on April 8, 2014. All of these XP-based embedded versions no longer receive mainstream support. However, they continue to receive *extended support*. Extended support means that they will continue to get security updates provided to them.

Windows XP Embedded will no longer receive extended support as of January 12, 2016. WePOS 1.1 will lose extended support on April 12, 2016. Images based on WES 2009 will continue to receive extended support until January 9, 2019. POSReady 2009 will receive extended support until April 9, 2019.

Important If you have created a product, you can continue to ship and use licenses for a product for five (5) years after extended support ends. However, your products will no longer receive any security updates. Therefore, they will continue to become less secure and more vulnerable to attacks over time.

Support for SATA drives was not included (inbox) with Windows XP-based embedded OSs. Therefore, you must have a third-party SATA driver available for use at installation (added to the image for WES). Alternatively, you can install the OS on an older, IDE/ATA-based HDD. If you need to use the SATA driver, have it available on a floppy disk, especially for WePOS 1.1 or POSReady 2009-based products.

While an OEM-only version of Windows XP for 64-bit was released, these Windows XP Embedded products are all 32-bit for x86 only. The minimum requirements were 128 MB of storage and 256 MB of RAM.

Windows XP Embedded (XPe)/Windows Embedded Standard 2009

Both of these versions allow you to create a very small OS image. The image can be as small as 50 to 100 MB in size for a headless system with only kernel and networking support. If you included almost every component, the image would still not be much more than 700 MB in size. By today's standards, that is still quite small.

These products shipped with a tool called *Target Designer*. Target Designer allowed you to build a custom image based on almost 1,000 operating system components and over 9,000 unique device drivers. The components were stored and referenced in what was known as a repository. A local installation of Microsoft SQL Server Express managed this repository. The system requirements only mention XP Pro SP2 or Vista. However, we have verified that you can also install and use these tools on Windows 7 and Windows 8.1. Depending on the desktop version you installed the tools on, you might have needed one of the following versions of SQL Server Express:

- For Windows XP - SQL Server Express 2005.
- For Windows 7 - SQL Server Express 2008.
- For Windows 8/8.1 - SQL Server Express 2012.

If you are considering developing on either of these products, we recommend using Windows Embedded Standard 2009 because of its longer support lifecycle. You may have an existing deployment of XP Embedded images that have not been upgraded from SP2. If that is the case, then we strongly urge you to upgrade them to SP3. The upgrade would at least provide you with one more year of support because images based on SP2 are no longer supported.

Consider starting a new design based on WES 2009 only if you have determined that your hardware cannot perform adequately when running a newer OS version (such as Windows Embedded 7). Also consider WES 2009 if you are physically limited in the amount of storage or RAM on the system. For example, you have a storage device that only supports 1 or 2 GB of storage or it only has 256 or 512 MB of RAM.

Because of the overall architectural improvements, we generally advise that you use embedded OSs based on Windows 7 or Windows 8/8.1 instead. They provide better security and functionality and will continue to be supported for a longer timeframe. These newer versions offer support for the Enhanced Write Filter (EWF), the File Based Write Filter (FBWF), and the Registry Filter. We will discuss these in greater detail in the Windows Embedded Enabling Features section.

Windows Embedded Point of Service/POSReady 2009

These versions are ideally suited to customers who want to run a sales application for a cash register. They are also well suited for use as thin-clients that primarily use the remote desktop application to connect to servers to run remote applications.

Their installation requires a setup wizard that is similar to the one used when deploying the desktop version of Windows XP. However, you can also use a script to make the embedded installation setup wizard deploy using an *unattended.xml* answer file.

The File Based Write Filter and the Registry Filter are available on these systems for disk protection. These services will normally be on (running) by default. However, unless you specifically enable and select a partition to protect, they should not affect anything in the system. If you will not be using them with your design, we recommend that you disable those services to avoid any potential problems. You learn how to disable them later in the Troubleshooting section.

If you are considering a new deployment, consider using POSReady 2009 instead of WePOS 1.1. Again, consider starting a new design based on POSReady 2009 only if you have determined that your hardware cannot perform adequately when running a newer OS version (such as Windows Embedded 7). Also consider POSReady 2009 if you are physically limited in the amount of HDD storage or RAM on the system.

Windows 7-Based OS

These products are based on Windows 7 and were released one to two years after the desktop version was launched. The two general versions are known as Windows Embedded Standard 7 (WES 7) and Windows Embedded POSReady 7. Each version allows the selection of features based on Windows 7 Professional or Windows 7 Enterprise. The run-time licensing cost per device depends on the set of features you select.

Windows Embedded Standard 7

This version is the componentized version that allows you to include only the components that are needed for your device to function. You can create images that take up as little as 500 MBs, but not much functionality is available at that image size. If you need UI support, then your image will probably be 1 to 2 GB in size. If you want Aero Glass support (the UI look introduced with Windows Vista), then the image size increases to 3.5 GB.

While you can still choose the features you want to include, this version is slightly less granular than its XP-based predecessors. Instead of approximately 1,000 individual OS components, you have approximately 150 different feature packages to select from to build your image. Instead of 9,000 individual driver components, there are now around 500 driver packages. The tool that you use to configure your image for deployment is no longer called Target Designer. It is now called *Image Configuration Editor* (ICE). The component packages are now stored in a folder called a *distribution share*. No SQL Express or repositories are needed for management of these packages.

WES 7 continues to offer the Enhanced Write Filter and the File Based Write Filter for disk protection. Additionally, there have been some improvements to USB Boot, Custom shells support, and Notification and Popup suppression features.

Windows Embedded POSReady 7

This version is much closer to the desktop version of Windows 7 Enterprise or Professional in the functionality that it provides. Because of this, the minimum suggested hardware is a bit more than it is for a WES 7 image. For x86, we suggest a minimum of 1 GB of RAM and 16 GB of free disk space. For x64, you will need 2 GB of RAM and 20 GB of available disk space.

Beyond the base installation, there are approximately 50 other packages that you can add to an image using DISM as needed. To read the list of what is available, go to http://go.microsoft.com/fwlink/?LinkId=671706.

EWF and FBWF are available for disk protection. However, for POSReady devices, it is more common to see them using FBWF when running on a single partition. Additional out-of-band (OOB) updates are available for POSReady 7 Volume License customers from the following site: http://go.microsoft.com/fwlink/?LinkId=671707.

Windows Thin-PC

This is mentioned here because it is a specific, static image that Microsoft created for use on Thin Client devices. It was based on Windows Embedded Standard 7. This OS includes the File Based Write Filter and Registry Filter components if customers choose to enable them. Windows Thin-PC is an OS that an enterprise would license directly from Microsoft for use on their existing hardware systems. Learn more by going to the page at http://go.microsoft.com/fwlink/?LinkId=671710.

Windows 8/8.1-Based OS

These products are based on Windows 8 or Windows 8.1. The two main embedded versions we will discuss are Windows Embedded 8 Standard and Windows Embedded 8.1 Industry.

Windows Embedded 8 Standard

This product works in a similar fashion as WES 7. You can configure your own custom image based on around 150 feature packages. The Image Configuration Editor is still there to help you create and configure that image. There were some improvements to ICE that help make building an image for deployment and placing it onto a USB drive simpler. For x86, we suggest a minimum of 1GB of RAM and a minimum of 3 GB of available disk space. For x64, you will need 2 GB of RAM and a minimum of 6 GB of free disk space.

This version supports running Windows Modern Apps using enterprise side-loading. However, the Microsoft Store application is not available, so users cannot purchase third-party applications using that mechanism. Windows Embedded 8 Standard includes a number of new UI paradigms to better support touch screen devices and new form factors such as tablets.

In addition to still offering EWF and FBWF, this version introduced a new write filter called Unified Write Filter (UWF). A new tool called the Embedded Lockdown Manager (ELM) was also introduced. ELM makes enabling and configuring the embedded features on a deployed image easier.

No update or new release to include 8.1 support is planned for this version. Windows Embedded 8 Standard continues to receive functional and security updates as defined on the product lifecycle site.

Windows Embedded 8/8.1 Industry

These embedded versions are actually a superset of the desktop versions. This means that they include everything that the Professional or Enterprise products do, but they also include all Lockdown features. Even though the name is different, these should be thought of as POSReady versions. The system requirements for 8.1 Industry are the same as those for the desktop Enterprise or Professional editions.

This version no longer includes EWF or FBWF as write filters, but UWF is present. Additionally, some new Lockdown features have been added, including Gesture Filter and USB Filter.

A version called Windows Embedded 8 Industry did ship, but support for that version will end on January 12, 2016. All customers currently have 24 months from when that product was released (June 24, 2013) to move to the 8.1 version in order to remain fully supported by Microsoft. To move to 8.1, you must perform a new installation of the image. Unfortunately, it is not possible to perform an in-place upgrade of existing 8 Industry images.

Windows 10-Based OS

These products are based on Windows 10. There are two versions that we will discuss: Windows 10 Enterprise and Windows 10 IoT Core.

Windows 10 Enterprise/Windows 10 IoT Enterprise

At this point, there is no longer a difference between the desktop and embedded binaries that ship with the product. The lockdown features that are available for Windows 10 are present as an optional Windows Component that can be enabled. To turn them on or check the status, go to **Programs and Features** and select **Turn Windows Features On or Off**. The following lockdown features will be available:

- Embedded Boot Experience.
- Embedded Logon.
- Embedded Shell Launcher.
- Unified Write Filter.

There is no difference between the binaries that ship in Windows 10 IoT Enterprise and those that ship in Windows 10 Enterprise. The difference in the titles indicates a difference in the license terms about how they can be used. For more information on the system requirements, go to http://go.microsoft.com/fwlink/?LinkId=671720.

Windows 10 IoT Core

This is a new product that Microsoft has introduced as part of the release of Windows 10. It is based on something called “OneCore” and runs the same kernel that is present on the desktop OS. However, this OS is fairly limited with regard to the services and features that are present or can be added to its image. The footprint it requires is around 2 GB of storage. It currently runs on the Intel-based Minnow Board Max and the ARM-based Raspberry Pi 2.

Developers can run C/C++ console apps or background tasks. They also have the ability to develop and run Universal Windows Apps on this OS. Currently, only one UWP App can run at a time on the device. To learn more about this and see some sample projects, please go to http://go.microsoft.com/fwlink/?LinkId=671721.

For Embedded Systems

This category is mentioned here because it includes the word “embedded” in the title. These products are exactly the same as the desktop or server versions. They are only different in how they can be used based on license restrictions defined in the EULA that is specific to industry scenarios. They are generally known as *FES editions*. FES versions do not contain any of the EEFs or Lockdown features found in the other embedded versions previously mentioned.

Some of the products names in this category include:

- Windows XP Professional for Embedded Systems.
- Windows 7 Professional for Embedded Systems.
- Windows 7 Ultimate for Embedded Systems.
- Windows Embedded 8.1 Professional.
- Windows Embedded 8.1 Enterprise.
- Windows Server 2008 for Embedded Systems.
- Windows Server 2012 R2 for Embedded Systems.
- Microsoft SQL Server 2014 for Embedded Systems.

Purchasing and Licensing

OEMs who plan to develop systems will need to work with a distributor in their area. For someone in the United States, read the information on the page at http://go.microsoft.com/fwlink/?LinkID=671723. A distributor is a company that sells the embedded development tools and the packs of run-time licenses. This model for OEMs has operated in the same way since Windows XP Embedded. The same model exists for Windows Embedded 8 Standard and the POSReady versions as well.

With the exception of Windows 8/8.1-based products, images that were licensed only required a product ID (PID). They did not require activation as part of the deployment process. However, WE8S/8.1 Industry now require activation like the desktop versions. The volume license editions of POSReady 7 also require activation.

Enterprises may have an interest in using some of the embedded versions in their environment. Now, they can also purchase Volume Licensed-versions as upgrades to existing systems directly from Microsoft. For additional details on what versions are available, review the "Enterprise Volume Licensing FAQ" at http://go.microsoft.com/fwlink/?LinkId=690206.

Support Lifecycle

In most cases, when Microsoft releases a product, it establishes a support lifecycle of 10 years. For the first five years, the product is in what is called *mainstream support*. This means that, if a customer identifies a problem or bug with some functional aspect of the product, they can request a fix for it. They must, however, provide a sufficient business impact case. A quick fix engineering (QFE) solution or Hotfix would then be released to address the reported issue.

After the product exits mainstream support, it enters what is known as extended support. Issues related to functionality will no longer be considered, but Microsoft will address any new, security-related issue and release security updates. A customer may want to continue receiving functionality fixes for a product after mainstream support ends. To do so, the customer must purchase the appropriate tier of *Custom Support Agreement* (CSA) from Microsoft before mainstream support ends. The customer must then continue to renew the CSA each year to maintain that agreement.

For easy reference, Table 2 contains the end dates for extended support for the products in this paper. Note that the date format is Month/Day/Year.

Version	Extended Support Ends
Windows XP Embedded (all versions)	1/12/2016
Windows Embedded for Point of Service 1.x	4/12/2016
Windows Embedded Standard 2009	1/8/2019
Windows Embedded POSReady 2009	4/9/2019
Windows Embedded Standard 7	10/13/2020
Windows Embedded POSReady 7	10/12/2021
Windows Embedded 8 Standard	7/11/2023
Windows Embedded 8.1 Industry	7/11/2023
Windows 10 Enterprise	10/14/2025

Table 2: Extended Support End Dates

For a complete list that shows when the embedded products were released and when they lose extended support, go to http://go.microsoft.com/fwlink/?LinkId=671725.

Windows Embedded Enabling Features

In some instances, WES, users can select subsets of OS functionality for use. There are also features that are embedded specific that have been created and included with the products over time. Most of these features are intended to help improve the reliability or stability of the embedded system. In some cases, they help to restrict how a user can interact with the system. Often, the goal is to help transition the device from being a general purpose computer to more of an 'appliance'. As an appliance, it can be reset to a factory original condition simply by rebooting it.

In this section, we will review the most common Embedded Enabling Features and Lockdown features that have been created. We indicate the versions in which they are present and any particular limitations to their use. We will not discuss every EEF or Lockdown feature because some are just not used that often anymore. The reason they are not used is because better alternatives now exist or hardware advancements have made them unnecessary.

Write Filters

Write filters are designed to limit or, in some cases, prevent any writes from occurring to a disk or selected partition(s). Their advantage is that they can significantly help to minimize disk corruption. They can help prevent things, like a virus or malware, from permanently infecting or altering a system. With some write filters, you can boot from media that is physically locked as Read-Only, which makes the OS even more stable and reliable.

Write filters do have some disadvantages. Unless properly configured, user data can be lost, and the write filter overlay can fill up over time. This can limit the potential maximum system uptime before a system reboot is needed. If the overlay space becomes exhausted, the system will become unresponsive and eventually crash. Additionally, avoid using a page file with your system on a partition that is protected using a write filter. In general, for embedded systems, it is better to disable the page file to avoid conflicts or other issues.

With write filters, there is the potential for repetitive effects when booting from an image that is not updated with persistent data at each boot. The Daylight Savings Time (DST) change is a common example of this issue. The clock will need a way to automatically adjust if necessary. We urge you to test this scenario. To learn about approaches that can be used for the DST issue with regard to WES 7, go to http://go.microsoft.com/fwlink/?LinkId=671726.

The table below shows which write filters are available for various embedded versions:

OS Version	**EWF**	**FBWF**	**UWF**
Windows XP Embedded	Yes	Yes (SP2 or later)	No
Windows Embedded Standard 2009	Yes	Yes	No
WePos 1.1	No	Yes	No

POSReady 2009	No	Yes	No
Windows Embedded Standard 7	Yes	Yes	No
POSReady 7	Yes	Yes	No
Thin-PC	No	Yes	No
Windows Embedded 8 Standard	Yes	Yes	Yes
Windows Embedded 8 Industry	No	No	Yes
Windows Embedded 8.1 Industry	No	No	Yes
Windows 10 Enterprise	No	No	Yes

Enhanced Write Filter

The first write filter introduced for Windows XP Embedded was the Enhanced Write Filter. When enabled, it protects one or more selected partitions from writes at the sector level. This means that everything on a protected partition can be read from. However, any attempt to write to the disk caches in an overlay. EWF offers three modes of operation that define where the overlay is stored:

- **EWF-DISK:** This mode creates a separate EWF partition to store the write filter state and status. It then uses the remaining unallocated disk space on the partition as one or more overlay checkpoints. This can be useful for systems where they might want to roll back to a previous update level if a problem is detected. EWF-Disk is not generally used that much and has been removed in newer versions.
- **EWF-RAM:** This mode also creates a separate EWF partition in the unallocated portion of the disk to store filter state and status. This partition is usually around 8 MB in size per protected partition. Any data writes that would normally go to the disk are cached in a RAM overlay instead.
- **EWF-RAM-REG:** Unlike EWF-RAM, which requires a separate EWF partition to be present on the disk, this mode stores the filter state in the registry instead. This can be useful especially on certain devices that only support having a single partition present, such as Compact Flash devices. Again, all writes that would normally be written to the disk are cached in a RAM overlay instead. The disadvantage of this method is that, instead of just disabling the filter, you must also perform a *commit* before the filter will disable. If you do this directly after a boot, there should be no issues. However, if you have been running the system for a while, you may inadvertently commit other unwanted changes.

For EWF-RAM and EWF-RAM-REG, the system allocates a certain portion of the system RAM to use for the overlay space as needed. If your system has 2 GB of RAM, it is not uncommon for 512 MB or 1 GB to be used for the overlay. Alternatively, if you have 4 GB of RAM, 1 or 2 GB may be used for the overlay. To learn more about how the overlay space is used, go to http://go.microsoft.com/fwlink/?LinkId=671905.

There is one vital thing to keep in mind with regard to the overlay. As the overlay becomes full, you must make the system perform a reboot to free up the overlay space. If you do not do this, the system could become unresponsive or crash. For this reason, it is important to determine the average number of writes to any protected partitions over a 24-hour period. For example, you determine that 100 MB of writes occur each day, and there is 1 GB of free RAM space available for overlay use. You could infer that the system will operate for almost 10 days before needing to reboot. However, as a precaution, you might want to schedule a weekly reboot. Alternatively, you can examine the written data and decide to turn off those services or redirect them to another disk or partition that is not protected.

EWF also provide an Application Programming Interface (API) so that you can create your own application that interacts with it. To learn more about the API, go to http://go.microsoft.com/fwlink/?LinkId=671906.

To read about common system items that you should review when trying to minimize writes on a system partition, go to http://go.microsoft.com/fwlink/?LinkId=671908.

To read the Embedded Device Robustness Checklists, go to http://go.microsoft.com/fwlink/?LinkId=671909.

To find other writes that might be occurring on a given partition, we suggest you monitor activity with a utility like *Process Monitor*. Configure Process Monitor so it writes its data to a separate, unprotected partition if EWF is enabled or performs the monitoring when EWF is disabled. This utility will capture a large amount of data in just a few minutes. To simplify analysis, do not capture more than 2 to 5 minutes of data at a given time. To learn more, go to http://go.microsoft.com/fwlink/?LinkId=671910.

EWF-RAM for Read-Only Systems

Because of industry regulations or an attempt to increase the overall reliability of the system, some systems types will boot and run from Read-Only media. This could be media that has been locked using a physical switch, such as an SD card or certain USB Flash drives. It may be an item that is always read-only like a DVD-ROM. For these deployments, you will normally need to let the system run on read/write media while you perform other setup tasks. Then turn on the read-only status once everything is fully configured.

A read-only setup significantly reduces the chance of accidental changes or corruption to the disk that power surges or sudden power loss might cause. Mechanical failures, gamma rays, or extreme weather conditions might still be able to affect the information on these drives. However, this setup is about as safe as you can make an image without including multiple read-only disks for fallback and redundancy purposes.

There is a disadvantage to an image that is physically write-protected. It generally becomes impossible to update that image without the physical intervention of a technician in the system. Even if functional updates are not required, new security threats continue to be found. Unless these systems do not operate on any form of network, plan an update on some periodic basis, even if only once a year. A security threat would not be able to permanently infect the image on the read-only media. However, a threat could possibly infect the system as soon as it boots or modifies any drives that are not read-only.

To properly enable this for Windows 7 and beyond, there are two files you must delete before flipping the read-only switch. Both are called *bootstat.dat*, and one version is located in \boot and the second version is in the \window folder.

One question you might ask is, “How much space can my EWF Overlay consume in my system?” A blog was written when Windows Embedded Standard 7 was still in beta. WES 7 was called Windows

Embedded Standard 2011 at that time. The blog discusses this question and the potential range of overlay use.

EWF-HORM for Quicker Boot

If you have used a Windows laptop, you might be familiar with shutdown, log off, reboot, sleep, and hibernate. Sleep sets the system to a very low power state, but continues to draw power from the battery. This keeps the contents stored in RAM active and available. Sleep can preserve that system state for a few hours, or possibly even a few days, but the battery will run out of battery power. If that happens, the outcome is similar to pulling the power cord while the system is running. Hibernate takes all of the information currently stored in RAM and writes it to the hard disk in a single file. This file is called *hiberfil.sys,* and it is generally the size of all the running applications loaded in your system's RAM.

HORM stands for Hibernate Once, Resume Many. HORM was created to help embedded OEMs who wanted to have a quicker apparent "boot time" for their systems when customers turned them on. It also allowed them to have an application already "running" as the shell. This could save time loading and initializing that application. For example, on XP Embedded, a system could take two minutes to boot and load an application before it was fully ready for use. However, if you used HORM, the entire system restore might only take 30 to 45 seconds. A system resuming from hibernation still has to re-enumerate hardware and re-establish any network connections.

There are other design concerns and complications that must be considered before using HORM. For example, the system may want to use one or more unprotected partitions as a data storage drive. There is a certain sequence you must follow programmatically for dismounting those partitions from the system before creating the hibernation state. Any application that was running cannot have any open file handles to those dismounted partitions. First, because the partition is being dismounted, the file handle is no longer valid at that point. Second, because this is a HORM image, if data had been written during a previous run, the file pointer location would no longer be correct. Therefore, you would start overwriting any previously stored data at each boot. To avoid all these issues, only open files after the system resumes from hibernation. For information on how to dismount partitions before creating the hibernation file, go to http://go.microsoft.com/fwlink/?LinkId=671912.

This article, "Dismounting Volumes in a Hibernate Once/Resume Many Configuration", has a complete code sample. Generally, you would want to use a batch file that calls the sample code, such as *dismountsample.exe.* You would build this utility based on the sample code in the link above. For example:

```
@echo off
Dismountsample.exe
YourApp1.exe
YourApp2.exe
```

When called, the sample batch file above first dismounts any drives you specified and then causes the system to go into hibernation. Once the system resumes from hibernation, it continues to execute the next instruction in the batch file, which is to launch YourApp1.exe and then YourApp2.exe, etc. Using this mechanism, you can launch any applications you need after resuming from hibernation.

Servicing a device that is protected with a write filter is already complex, but servicing a system running HORM adds another layer of complexity. In addition to disabling the write filter, you also have to disable HORM. Once the updates have been installed, you need to create a new hibernation state again. To read about the steps you should follow when servicing a HORM-based system, go to http://go.microsoft.com/fwlink/?LinkId=671914.

EWF API

The EWF API is available to OEMs, and it provides mechanisms to check the EWF state and control EWF, if desired. You can use the EWF API in place of ewfmgr.exe. Some OEMs have developed utilities that run in the system tray based on the EWF API's functions.

To check the overlay details, you can write an application using the **EwfMgrGetOverlayStoreConfig** function to view the EWF overlay configuration information. The following pages contain some sample source code that demonstrate how to get the details:

- http://go.microsoft.com/fwlink/?LinkId=671915.
- http://go.microsoft.com/fwlink/?LinkId=671916.

File-Based Write Filter

Microsoft introduced a new type of write filter when the Service Pack 2 update for Windows XP Embedded was released. This new write filter is the File-Based Write Filter (FBWF). EWF works at the sector level and can only protect an entire partition at a time. FBWF works at a higher level in the file system to protect files and directories. You can configure FBWF to write certain files or directories directly to the disk instead of having the write filter cache them. HORM is not an available option for a disk when FBWF is in use.

FBWF is mostly used by systems configured as thin-clients or point of sale terminals. These systems use FBWF because they often run with only a single disk and a single partition. These systems might want an application to write out any logs they need. However, they want to keep the C:\Windows folder locked down to prevent corruption of system files. FWBF allows them to do that.

FBWF and EWF are offered as disk protection mechanisms in several Windows Embedded products. They can even be included and used in the same OS image. They cannot be used at the same time on the same partition. However, you could configure EWF to protect everything on a C: partition, while FBWF protects some files and directories on the D: drive. Generally, this is an uncommon configuration, but it is possible.

With regard to the overlay, there is a key difference between EWF and FBWF. EWF will dynamically expand the overlay as needed so long as there is free RAM. However, you must specify how much RAM FBWF should use for an overlay. By default, FBWF will only reserve 64 MB, so you will likely want to increase this value. To change the size of the overlay that will be used on the next boot, use **fbwfmgr /setthreshold 1024** to reserve 1 GB of space. For more information on FBWF configuration commands, go to http://go.microsoft.com/fwlink/?LinkId=671917.

Common FBWF Scenarios

When using a single partition with FBWF, there are some common exclusions you might want to add to the system to make it easier to maintain. Some of these will also list registry key locations, which we will discuss further in the Unified Write Filter WMI Provider Reference

Instead of exposing an API like EWF or FBWF, you can now control UWF using WMI Providers. To learn more about those providers, go to http://go.microsoft.com/fwlink/?LinkId=671930.

Registry Filter section. For more information, go to

- http://go.microsoft.com/fwlink/?LinkId=671918.
- http://go.microsoft.com/fwlink/?LinkId=671919.

FBWF API

Like EWF, FBWF also offers an API so that OEMs can programmatically control the filter if they wish. For more information on the API, go to http://go.microsoft.com/fwlink/?LinkId=671929.

Unified Write Filter

Microsoft introduced the Unified Write Filter with Windows Embedded 8 Standard. UWF is now the only write filter included with Windows Embedded 8.1 Industry. While it is intended to provide an experience that is similar to EWF and FBWF, it is implemented at the sector level like EWF. This means that it does not recognize files or directories in the same manner that FBWF does. Therefore, UWF cannot write directly to excluded locations. It does allow you to specify folders or files for an exclusion list so that writes to those locations will persist. Unlike FBWF, writes to those excluded locations will impact the overlay space being used. Only a system shut down or a clean reboot causes the modified data in those locations to persist and frees up overlay space.

UWF allows for two modes of operation: UWF-RAM and UWF-DISK.

- **UWF-RAM:** Uses system RAM to cache any writes to the protected partitions. You must use this mode if you are using Read-Only media or HORM.
- **UWF-DISK:** This mode uses a pre-defined file on the system volume. The volume size is fixed and cannot be increased during run time. Unlike EWF-DISK mode, this mode does not allow for multiple checkpoints of overlays to be established for rollback purposes.

UWF offers HORM support, but it currently requires that all disks and partitions in the system be protected by UWF, and you cannot specify exclusions. If you need to persist data somewhere when the system is running, use a network share or some other location not locally mounted by the system.

UWF has now incorporated the Registry Filter service. You can still add registry exclusions, but there is not a separate RegFilter service running on the system.

Common Usage and Design Considerations

Writes to excluded folders can still increase the overlay space a protected UWF partition uses. We recommend using UWF as if it was EWF by having it protect an entire partition. For example, you might have data you want to persist or a database you want to run on your system. We suggest that you create one or more additional partitions for data and do not protect them with UWF. For example, you might use C: as your system partition, which is protected by UWF. You can then create an unprotected D: partition (and possibly an E: partition) as a location to permanently store any data.

Unified Write Filter WMI Provider Reference

Instead of exposing an API like EWF or FBWF, you can now control UWF using WMI Providers. To learn more about those providers, go to http://go.microsoft.com/fwlink/?LinkId=671930.

Registry Filter

The Registry Filter allows customers to use a write filter and still join their devices to a domain, such as thin-client systems in an enterprise. Before the Registry Filter, domain devices would appear to work correctly after initial deployment. After a few months, however, a system reboot or sudden power loss might occur. Consequently, the domain controller would stop trusting hundreds, or even thousands, of these domain embedded clients, and these clients would “fall off” the domain.

The reason for the fall off is the domain controller updates its *Machine Secret Key* periodically. Usually it is every thirty, sixty, or ninety days, depending on how the domain administrator has configured the domain controller. This value normally persists to the registry and serves as a handshake between the domain controller and a device. However, when a write filter was enabled, the updated domain Machine Secret Key would be lost in the drive overlay. A device would try to use the original Machine Secret Key it had been given when it first joined to the domain. Because this key was now expired, the domain controller would reject the device.

To solve this issue, Microsoft created the Registry Filter. It is designed to work with EWF and FBWF and allows the persistence of two monitored registry keys. The first is the *TSCal* key for use with RDP connections, and the second is the *Domain Secret Key*:

- HKEY_LOCAL_MACHINE\Software\Microsoft\MSLicensing
- HKEY_LOCAL_MACHINE\Security\Policy\Secrets\$MACHINE.ACC

The location of the Registry Filter-monitored keys is:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RegFilter\Parameters\MonitoredKeys\0]

"ClassKey"="HKLM"

"FileNameForSaving"="MSLic.rgf"

"RelativeKeyName"="Software\\Microsoft\\MSLicensing"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RegFilter\Parameters\MonitoredKeys\1]

"ClassKey"="HKLM"

"FileNameForSaving"="MacAcc.rgf"

"RelativeKeyName"="Security\\Policy\\Secrets\\$MACHINE.ACC"
```

In addition to the two monitored keys, OEMs are also allowed to add additional registry keys for monitoring. Anything else you add is used “As-Is”, which means that only the two official keys are supported. You can only use Registry Filter to persist custom keys in the HKLM registry root. The system can update registry keys early in the boot process before the Registry Filter loads. Therefore, we cannot guarantee that the Registry Filter will persist all registry keys in the SYSTEM hive. The Registry Filter can only persist registry keys that change after it loads and starts tracking registry changes.

The Registry Filter does allow monitored keys to persist, but that persistence is not always instantaneous. If a monitored key is updated, the Registry Filter attempts to persist that change immediately. However, because of disk write caching policies, that persistence is not always instantaneous. Because of this, it can sometimes take 1 to 2 minutes before an updated key is actually persisted to the physical HDD. Keep this in mind if you make registry changes quickly and then immediately shut down or reboot the system within a few seconds. Those changes might not have been fully persisted to the HDD. If you want to make rapid updates to files and registry keys, it is better to disable your write filter when you are applying those updates.

Shell Launcher

A common scenario for embedded systems is to present a single application to a user. This can be true with an information kiosk, medical device, or gaming system. For the XP-based systems, it was possible and common to present one shell for an end-user account. In addition, the system could provide *explorer.exe* as the shell for the administrator account. This was accomplished using the registry keys documented at http://go.microsoft.com/fwlink/?LinkId=671931.

This registry key approach worked well. However, it would sometimes require some additional script development, such as creating a script to log the user off if the shell application exited. The registry keys for Windows XP continue to be valid. They are an option for OEMs on all the embedded versions including 8.1 versions.

In Windows Embedded Standard 7, Microsoft introduced the *Shell Launcher* component. Shell Launcher helps to simplify the custom shell process and provides additional support, such as re-launching the chosen application when it exits. In addition to supporting custom shells for specific users, it also allows a shell to be specified for a particular user group. This can be beneficial in an enterprise environment where you might have thousands of users, but only need three different shells for the user groups. Based on the return code the application provided when it exits, the Shell Launcher can perform one of four actions:

- Restart the Shell (the exiting application).
- Restart the device.
- Shut down the device.
- Do nothing.

You can also use the Embedded Lockdown Manager to help configure the Shell Launcher. For more information, go to http://go.microsoft.com/fwlink/?LinkId=671932.

Keyboard Filter

The *Keyboard Filter* was introduced as part of Windows Embedded Standard 7. Before its introduction, you could prevent some key combinations by intercepting them if you registered them with custom applications. However, not all combinations could be blocked. In an effort to help lock down devices on which OEMs only wanted users to interact in a certain manner, Microsoft created the Keyboard Filter. It allows you to block the use of individual keys or key combinations. Some common, pre-defined key combinations that you can block include: Alt+F4, CTRL+ALT+DEL, Windows+L, etc.

In addition to the pre-defined keys, you can also filter specific, custom key combinations. In Windows 7-based products, filtering only worked with physical keyboards. In the 8 and 8.1 versions, you can also filter keys generated from the *On-Screen Keyboard* (OSK) or a combination of physical keys and OSK keys simultaneously.

You can now configure the Keyboard Filter as part of the Embedded Lockdown Manager. Note that the Keyboard Filter cannot block the *Sleep* key. For additional information, go to http://go.microsoft.com/fwlink/?LinkId=671933.

Gesture Filter

The concept of gestures was introduced in Windows 8 as part of the new UI design to make touch an integrated aspect of the system. Touch is critical for using Windows Store Apps and is also involved with configuring the system using charms bars.

The Gesture Filter was introduced in Windows Embedded 8.1 Industry. With it, OEMs can control what gestures they want their users to access. Certain gestures, such as the top edge and bottom edge gestures, bring up an app bar that is specific to each Windows Store App. These gestures cannot be blocked. You can block the following gestures: Left, Right, Top extended swipe, and each corner individually.

When you block a gesture, you can still access its associated UI using a keyboard shortcut or an application. For example, when you block the right swipe gesture, the Win+C keyboard combination can still access that charms bar. To block a key combination, you must also use the Keyboard Filter.

Important The Gesture Filter does not filter gestures on accounts with Administrator rights. For more information, go to http://go.microsoft.com/fwlink/?LinkId=671953.

Dialog Filter/Pop-up and Message Box Filter

A portion of this functionality was first introduced as part of XP Embedded. The functionality was known as the *Message Box and Balloon Pop-Up Interception* component. With that component, you could set a default reply for a message box and also suppress balloon pop-ups. For more information about the XP Embedded component, go to http://go.microsoft.com/fwlink/?LinkId=671954.

In WES 7, the Message Box and Balloon Pop-Up Interception component was known as the *Message Box Default Reply*. You configure Message Box Default Reply to capture windows when they are created. Once a window is drawn, Message Box Default Reply determines if the window should perform a default action or if Message Box Default Reply should block the window. All of the action occurs off screen. To help with this configuration process, Microsoft created a tool called the *Dialog Filter Editor*. For more information related to these features for WES 7, go to the following:

Dialog Filter Editor (WES 7):

http://go.microsoft.com/fwlink/?LinkId=671955.

Add Message Blockers (WES 7):

http://go.microsoft.com/fwlink/?LinkId=673956.

For 8 and 8.1, the functionality offered by the two components has been merged into a single Lockdown feature known as the *Dialog Filter*. The Dialog Filter Editor tool has been moved to the Embedded Lockdown Manager to simplify where and how this feature is configured. For more information, go to http://go.microsoft.com/fwlink/?LinkID=673957&clcid=0x409.

USB Filter

This is a new Lockdown feature that was recently introduced as part of Windows Embedded 8.1 Industry. It allows OEMs to control the devices (if any) and USB ports that are available for use on a system. With USB Filter, you can prevent users from attempting to manipulate or hack your system by plugging in a physical keyboard. You can also stop them from inserting a USB Flash drive that might contain viruses or malware. You can also authorize the use of specific devices, such as a particular brand and size of Flash drive by your technicians. The system would ignore all other Flash drives when inserted.

USB Filter functionality only works when the OS is running. Therefore, if you set the system BIOS to boot from a Flash drive, USB Filter cannot control USB access after a power failure or reboot. Check your system BIOS/UEFI settings when using this feature. For more information, go to http://go.microsoft.com/fwlink/?LinkID=673958&clcid=0x409.

Embedded Lockdown Manager

The Embedded Lockdown Manager (ELM) was introduced as part of Windows Embedded 8 Standard. It is a snap-in that runs in the Microsoft Management Console (MMC) and is designed to simplify the configuration of the installed Lockdown features. You can also configure ELM remotely on an 8.1 Industry or Standard 8 device. To do so, run ELM on your development computer, and then connect to those systems. ELM will only display the Lockdown features installed on the system. To make a feature visible, add that package or turn on the feature.

In addition to configuring the Lockdown features, ELM allows you to export a PowerShell script based on your settings for each feature. After configuring a single system, you could use the exported PowerShell scripts to configure other systems you wanted to lock down on new installation deployments. You can also use WMI and PowerShell to configure these features directly. To minimize problems, we recommend that you get one system working properly using the exported Power Shell script first before using it on other systems. For more information on ELM, go to http://go.microsoft.com/fwlink/?LinkID=673959&clcid=0x409.

Assigned Access

This feature allows you to restrict a specific standard account to only run a specific Windows Store App. Unlike the other embedded features, Assigned Access is only available on the desktop versions of Windows 8.1. It incorporates a combination of other Lockdown features; for example, some keyboard combinations and some gestures are restricted. Users cannot customize the individual keys or gestures that are blocked. For more information on Assigned Access, go to http://go.microsoft.com/fwlink/?LinkId=673960.

Windows 8 Application Launcher

While similar to the Shell Launcher, the Windows 8 Application Launcher launches a single Windows Store App when a given user account logs in. This feature and the Shell Launcher cannot be used together simultaneously. When this feature is turned on with kiosk mode enabled, even the start menu will not offer any programs.

Important The app you intend to launch must be installed on the device for the chosen user or be available to all users. If the application does not launch correctly when the user first logs in, the system will retry the launch every 100 milliseconds for up to 10 attempts. For more information on customizing your Windows Store App to provide a custom exit code to the Application Launcher, go to http://go.microsoft.com/fwlink/?LinkId=673961.

AppLocker

This is not an embedded-only Lockdown feature; it is present on the enterprise-based SKUs. AppLocker can restrict the system so that it only runs a specific subset of application executables. This functionality is available on 7 and 8-based systems. You can write scripts for AppLocker and customize it using group policy objects (GPOs).

This feature was known as *Software Restriction Policies* on Windows XP. However, its mechanism had management overhead. The overhead became very cumbersome in some environments when new versions of an application were deploying. Microsoft created AppLocker in Windows 7 to simplify this mechanism. To learn more about AppLocker on Windows 7, go to http://go.microsoft.com/fwlink/?LinkId=673962. For updates to AppLocker on Windows 8.1, go to http://go.microsoft.com/fwlink/?LinkId=673963.

Common Usage Scenarios

In this section, we will try to cover some common approaches OEMs use when they create an embedded image for a particular device market segment.

Kiosk Device

A kiosk device might be intended to only serve as an electronic billboard showing advertising. It might also be designed to provide more specific information, such as directions or locations of stores. Depending on how you expect users to interact with this device, you might turn on features such as Keyboard Filter, Gesture Filter, and USB Filter. The goal is to limit how users can interact with this system so that it does not change to an unknown state.

You will also likely use the Shell Launcher (or the shell registry keys) and configure the kiosk to have two user accounts. For the end-user account, the system logs on automatically and runs the custom kiosk application as its shell. For the administrator account, the system runs explorer.exe as the shell. Using the administrator account, you can install service updates and perform other necessary servicing, such as updating the ads that display on the device. Other options are to run the kiosk from an SSD type drive and use some form of write filter. Both of these approaches can help minimize disk corruption in the field once you deploy these devices.

Gaming

Creating an embedded system for use in a gaming machine, such as a slot machine, must meet and pass several, very strict regulations. In the United States, the Nevada Gaming Commission primarily mandates these regulations. However, each state or region can have its own unique regulations as well. If this is a market segment you are considering, carefully research these requirements before creating a gaming device.

For example, in Nevada, one of the requirements is that the system boots and runs from read-only media. This helps to guarantee that the image and software cannot be changed after it has been certified by the gaming commission. In this type of environment, most customers choose to leverage EWF-RAM mode. Then the system boots from an SD card, which has physically been locked, or a CD/DVD-ROM that contains the OS. Windows XP or Windows 7-based embedded versions are currently the best options for this type of scenario because they do not require individual OS activation. Windows 8/8.1 systems do require an activation code that must be written to each individual image. This activation code requirement makes them an impractical choice given the current gaming regulations.

Gaming devices can sometimes experience sudden power loss caused by power outages. However, it is not very common that they experience disk corruption because their boot media is generally in a physical, read-only state. Anytime you update a gaming device image, it must be re-certified. Therefore, these systems are not generally updated as frequently as enterprise devices. More often, they might get a new image with feature and security updates quarterly or yearly.

Medical

Medical systems need to pass their own set of certifications and regulations depending on where and how they will be used. In the US, you might need to have the device certified by the American Medical Association or the Food and Drug Administration. You should also carefully check the licensing terms for the specific embedded version you are considering using. Certain life-critical applications may not be allowed or supported by the license terms.

Windows Embedded is often used in medical diagnostic equipment, such as ultrasound systems. These systems will often leverage a write filter and possibly HORM to provide a faster boot time. They often need to have one or more partitions available to persist patient data locally as it is generated.

It is very common in hospitals for the power on these devices to be removed suddenly when they are moving from one location to another. Even if a write filter is used, this sudden power loss can lead to some level of disk corruption over time. For this reason, we strongly urge you to make the boot partition or OS use physical read-only media. Alternatively, include a small Uninterruptable Power Supply (UPS) as part of the system's design. The UPS allows the system to gracefully shut down over a few seconds once power has been removed. One way to include a built-in UPS is to leverage a hardware design that incorporates laptop hardware so it contains its own battery.

Military

This is another category where you should carefully review the license terms for the specific version of Windows Embedded that you are considering using. Depending on where and how you intend your system to be used, it is likely you must pass some level of Department of Defense certification in the US. Additionally, it might require other levels of certification depending on which branch intends to use the system.

If these systems are going to be on a network, then the regular application of security updates will be an important design consideration. Using a write filter and making them as ruggedized as possible is another option to consider given the environment in which they might be used.

Because timing may be critical in military situations, consider disabling background services and tasks to limit the number of processes that are running on the system. For example, the system should not start a disk defragmentation process in the middle of relaying mission critical information.

Space or Extremely Remote Locations

It very difficult to service a device that you intend to send to space or place in a remote location (Antarctica, ocean floor, etc.). Therefore, this category of device must be as robust as possible. Because of this, you might decide to make certain trade-offs. For example, you might never install updates or never provide other regular servicing to prevent the device from getting stuck in an unrecoverable state. Hopefully, these devices would only be on your own custom, private network. Using a private network should help minimize potential security threats compared to using a public network.

For this type of device, consider using drives that have been physically write locked. Also consider using EWF or UWF to protect the entire drive for your OS and application. If you need to store or log any data, use a second writeable drive.

To provide additional levels of redundancy, we suggest you use a second, or possibly a third, identical copy of the OS drive. You can then use a third-party utility, like GNU GRUB, to chain the boot loaders. That utility can also set up the fall back boot options or use a UEFI BIOS if that is available. If the first drive becomes corrupted for some reason (gamma rays, etc.), then the boot would attempt to fall back to the second drive. The system would use that second drive as its system disk and boot from it. You should plan to test and verify that this fallback operation works as intended before deployment.

Important Remember to verify that any hardware or system components used in this design can withstand the environmental conditions and temperatures of your chosen target environment. Additionally, conduct long-term tests, and carefully consider the scheduling frequency of a regular system reboot to help clear up the overlay space.

Thin Client Terminals

These devices are most generally used in enterprises or universities. Their main purpose is to allow users to remotely connect to a server using a remote desktop connection (RDP). Regular client systems on a domain must download and persist user-specific client settings. With Thin Clients, this can all be handled on the server once the connection is established.

With Thin Clients, users are generally only interacting with RDP because it is usually the only application running on these systems. Therefore, they are less vulnerable to corruption or inadvertent changes by an end user. We still encourage use of a write filter to protect them from potential virus infections acquired over the network or through some other mechanism.

RDP is the main focus on these systems; it will sometimes be set to launch automatically as part of the automatic logon process. RDB might be set as the shell instead of Explorer using the Shell Launcher. This helps to minimize potential user interactions with the local system. In addition to the built-in remote desktop application, some customers may also want to leverage third-party solutions, such as those offered by Citrix Systems.

Point of Sale Terminals

You will find these systems at retail store locations. Their primary purpose is to process customer purchases or perform price checks on an item. Often they will be running a form of POSReady. They might have a touch screen for input, possibly a magnetic stripe reader for credit cards, and a barcode scanner for reading UPC codes attached. Usually these devices will be connected by a serial connection, or, more often, USB.

POS terminals will most likely use their own custom third-party application as a full-screen application that their employees interact with for processing the customer sales. They might need a database connection or regular local updates for pricing and other information. Therefore, you should have multiple partitions with one unprotected if using EWF. FBWF would be another option on a single partition setup.

Embedded System Prototyping and Development

This section is intended to help OEMs understand the general process they should follow when starting embedded system development. The section will focus on best practices and some common troubleshooting approaches. The term *hardware platform* represents the components that you intend to use in your embedded system. This would include, but is not limited to, CPU, motherboard, RAM, Hard Disk Drives (HDD), Solid State Drives (SSD), video card, power supply, mouse, keyboard, and monitor.

Hardware Validation

When considering any new hardware platform for your embedded system, first deploy the desktop equivalent OS version. Then verify that your embedded system runs on the desktop OS without any problems. For example, if you want to use WES 7 as your embedded OS, first install your embedded system on Windows 7, and verify that it performs correctly. Your embedded system may have limited drive space; for example, the target drive you want to ship with is 8 GB in size. The solution is to attach a larger drive for this portion of the testing. Once you have installed the desktop OS, ensure that all the drivers you need are installed, and apply all applicable updates from Windows Update.

Additionally, test the performance of any in-box, custom, or third-party applications you intend to run to ensure they meet your requirements. To deliver the required performance for your users with a given application or usage scenario, you must choose the appropriate CPU. A single-core Intel® Atom™ chip might run everything. However, you might discover you need to use a quad-core Intel Atom chip or some other type of CPU.

After verifying that the desktop OS runs correctly and all in-box and third-party drivers are installed, collect a PMQ file from this system. A PMQ file contains information about your system hardware, including the installed in-box and third-party drivers. To generate the PMQ file, copy a tool called *Target Analyzer Probe (TAP.exe);* TAP.exe installs as part of your embedded toolkit. For WE8S, you generally find this utility (with two versions) at the following path:

%PROGRAMFILES%\Windows Embedded 8 Standard\Toolset\Embedded Tools\

1. tap_x86.exe
2. tap_amd64.exe

Copy the appropriate TAP version from your desktop development system to your target hardware device. (Use a USB Flash drive or whatever mechanism is easiest for you). Then run TAP.exe on the target system from an administrator command prompt:

```
tap_x86/o mytargetdevice.pmq
```

Copy the resulting PMQ file from that system, and move it back to your desktop development system.

At this point, set the first hard disk that contains the full desktop OS installation aside. As you move forward with your embedded development, you may experience problems. First, check if the same problems occur on the full desktop OS installation. If the problems are present, a general Windows issue or bug may be the cause. If the problems do not reproduce on this full desktop installation, you might need a component or feature in your embedded configuration. If you have ruled that out, check for differences in the registry setting or policy for a service in the system. Having the desktop OS installation as a reference for comparison can be very helpful in identifying and resolving these types of issues.

Virtual Machines

You can deploy an embedded OS to a virtual machine, such as Hyper-V or VMware®. However, Microsoft does not support this type of deployment. All versions of Windows Embedded are designed for deployment on physical hardware systems. You can use a VM for some initial prototyping or proof of concept work. However, it is much better to use the physical hardware you plan to ship on for a majority of your development and testing.

HDD vs. SSD Performance Considerations

When Solid State Drives were first introduced, they were quite small compared to traditional Hard Disk Drives, and their lifespan and reliability were not the greatest. They have now been in the market for several years, and those initial concerns and limitations have largely been overcome. While they are still more expensive per GB, using an SSD in your embedded system can provide some of the best overall performance improvements per dollar. It is not uncommon for SSD read and write speeds to be ten times those of a traditional spinning HDD. Some SSD drives include a small capacitor. The capacitor allows any data in the on-board RAM cache to be persisted to the drive during a power failure. This does not mean the drive will not get corrupted. However, you should not lose any data writes; this helps to minimize the potential for corruption. SSD drives with capacitors are becoming more prevalent.

The disadvantage of SSD drives is that their overall cost is approximately $0.50 (USD) per GB in the best case. The cost can be much higher on smaller drives. By comparison, current HDD prices have reached $0.04 (USD) per GB. If you need to store large volumes of data, HDDs are still the most economical option for use as a large data drive.

When SSD drives were first introduced, their expected lifespan was not nearly as long as that of traditional HDDs. However, wear leveling and other improvements in Flash storage technologies, such as TRIM support, have significantly reduced that limitation. SSD lifespans should no longer be a concern for most embedded systems. To ensure that Windows 7 properly activates the appropriate SSD support (such as TRIM), run *winsat.exe* at least once on your target hardware. This ensures that SSD support has been correctly profiled before capturing the image for deployment.

If you are using an SSD, you may still want to improve the overall system performance. Increasing the amount of available RAM is likely the next best option for improved performance. This is most applicable if you are running an x64-based version that lets you leverage the extra RAM beyond the 4-GB limit properly.

Third-Party Drivers

While a PMQ file will make note of in-box and third-party drivers, it cannot automatically add those third-party drivers to the configuration. You must create custom components for them so they are built into the image. As an alternative, you can install them during deployment, such as in the *sysprep audit mode*. This process has gotten easier on Windows 7 and later versions. However, it was also possible to install third-party drivers on XP Embedded-based versions after deployment.

When trying to install a third-party driver on an XP Embedded system, you may encounter a problem. Delete the following registry key from the system and reboot:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup]
"MinimizeFootprint"
```

If the driver then installs correctly, you can continue to run without this key or restore it. When this key is present, the system ignores all driver signing verification and does not call *Crypto APIs*. This key was created to reduce the overall size of XP Embedded images. However, given the size of modern disks, it is no longer a concern.

Third-Party Applications

If you want to include your own application with a system as you build it, you might want to create a custom component. However, a third-party application usually has its own installer. Therefore, you will most likely want to wait until sysprep audit mode or after deployment to install that application.

If a third-party application does not run on your system, there are a few approaches you can use to identify the problem. First, use *depends.exe* or *dependency walker* to scan the process that is failing to run. This should provide a list of the dependent DLLs the application expects to find on the system. If a DLL is missing, you may need to include an additional feature package when building the system. A second approach is to run Process Monitor on the system, and filter by the executable you are launching. Process Monitor should show any registry keys or files that it is failing to find. Later, we will discuss using Process Monitor for troubleshooting an embedded system.

Alternative Shell Development

You may plan to have your users interact with your own custom shell on the device. It is important that you understand that, from the system's standpoint, almost any executable can be a shell. For example, the file, *notepad.exe,* can be set as the shell for a system. The default shell for all Windows systems is explorer.exe. Explorer.exe shows you a **Start** menu and draws the Taskbar on your screen.

It is uncommon, but sometimes an application might have some form of dependency on explorer.exe. On Windows 8/8.1, certain UI options can only be seen when explorer.exe is running, such as the *Wi-Fi Networks Selection Charm* dialog. You can have your custom shell make API calls to implement your own version of this dialog. However, your custom version can no longer be launched within Control Panel like it did on Windows 7. Because of this potential explorer.exe dependency, we suggest you develop your own application to use as the shell. Avoid licensing an existing third-party application that was designed for the desktop environment.

Any shell development should happen on a desktop system because Visual Studio is not supported for installation or use on Windows Embedded clients. While the Shell Launcher is not available on a desktop version, you can still leverage the shell registry keys mentioned on the following page: http://go.microsoft.com/fwlink/?LinkID=671931&clcid=0x409.

For a test shell, create an end-user account, and log in as the end user. Then change its default shell from explorer.exe to *yourapp.exe* at the following key location in *regedit*:

HKEY_Current_User\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell

After launching your shell application, you can log off from that account in two steps. You will use a small batch file and a small VBS script. Instead of setting the shell registry key to point to your custom shell application, you would have it point to a batch file; see below.

Batch file (launcher.bat):

```
@echo off
Wscript.exe temp\mywrapper.vbs
```

The batch file above then invokes the following VB Script file (*mywrapper.vbs*):

```
sub shell(cmd)
  ' Run a command as if you were running from the command line
  dim objShell
  Set objShell = WScript.CreateObject( "WScript.Shell" )
  objShell.Run(cmd),3,True
  Set objShell = Nothing
end sub

shell "yourtestShellApp.exe"
shell "shutdown /l"
```

This version of the "shell" command (mywrapper.vbs) waits for the process to exit before continuing to run. You can also create a version that does not wait. Instead, it continues to run the script if you need to launch multiple programs simultaneously during login. To change this wait behavior, change the third parameter of the following line from True to False:

`objShell.Run(cmd),3,True` → `objShell.Run(cmd),3,False`

You can develop a custom shell in native or managed code. Sometimes developers have observed a slightly longer delay in loading and displaying .NET-based applications to the user. If that is your approach, consider creating a small, native application that displays a graphic as a splash screen to the user. The splash screen appears while the rest of the application loads and initializes. You can then create a managed app that sets its windows to draw on top, which hides the splash screen automatically. The splash screen app can then exit after a fixed time, such as 1 or 2 minutes. The app can also exit based on a notification from your .NET application.

On the following pages, there are reference samples for developing a splash screen app:

How to Load a Bitmap from a File

http://go.microsoft.com/fwlink/?LinkId=689970.

How to Draw a Bitmap

http://go.microsoft.com/fwlink/?LinkId=689971.

Simple Direct2D Application Sample (Contains examples for the APIs)

http://go.microsoft.com/fwlink/?LinkId=689972.

Debugging and Capturing System Dumps

During your embedded development, there might be times you need to debug an application or driver on your system. In this section, we will discuss some of the more common debugging tools you can use.

WinDbg

WinDbg (also known as *Windows Debugger*) is a free debugger from Microsoft. It is included with the Windows Driver Kit (WDK) SDK and is also available as a standalone download. You can use WinDbg to debug individual processes locally on a system. You can also use it to debug the kernel and drivers on a target system.

This kernel debug option is the one primarily used with embedded systems when debugging an unresponsive OS. This is especially important on embedded systems when a write filter is used to protect the single partition on the system. If the system is unresponsive, you can transfer the remote system memory to your connection to WinDbg. The command to capture the file is .**dump**.

If you are using a serial connection, transfer can take a significant amount of time. Consider using the */**burnmemory*** switch to reduce the effective amount of available RAM the system uses. For example, you could reduce the available memory to only 1 GB instead of 4 GB to make the capture process quicker. To do this, specify the amount of memory you want to remove from use. For example, to reduce memory usage from 4 GB to 1 GB of RAM, use `/burnmemory=3072`.

To establish a debug connection with another system on XP or Windows 7, you can use a serial port or 1394 connection. USB was also possible on Windows 7 using a special cable, but it is rare and not as stable a connection as the other methods. Therefore, we do not recommend USB.

With Windows 8/8.1, you can continue to use serial and 1394, but you now have the option of setting up a connection using KDNet (Ethernet). To read more about setting up kernel debugging, go to http://go.microsoft.com/fwlink/?LinkId=690057.

For more information on where to download a copy of windbg, go to http://go.microsoft.com/fwlink/?LinkId=690059.

Process Dump

Process Dump (ProcDump) is a powerful tool that captures a process that stops responding or fails because of an exception. It has quite a few parameters. Use it to gather data to analyze what is causing problems when an application is running on a system. Once in production, you do not have the freedom to interactively debug the system as you might in a lab setting. Process Dump is useful in the post-deployment scenario. For more information about Process Dump, go to http://go.microsoft.com/fwlink/?LinkId=690060.

Crash Dumps

There are three types of crash dumps you can configure Windows to collect when a problem occurs. The type of problem and the information you need to analyze it will help determine the dump you configure for capture. You may need to review and analyze the information alone. Alternatively, you may need to share it with a third-party driver manufacturer or with Microsoft directly as part of a support case.

- **Mini-dump:** This dump is captured when an application experiences an exception or something else happens to cause it to exit in an unexpected manner. Only the information related to the faulting process is captured in the mini-dump. Mini-dumps are generally small, usually only a few megabytes to a few hundred MB in size. The size depends on the size of the application in RAM when it was running. This type of dump captures data that is similar to what ProcDump captures. A mini-dump is best for identifying problems in a specific application, service, or user-mode driver.
- **Kernel-dump:** This dump captures the OS kernel and anything running within its context, including kernel mode drivers. A failure here might happen when a program makes an OS call that causes an exception, such as a registry or file operation. A failure might also happen because of a kernel mode driver that is running under the context of the kernel. Most often, kernel-dumps are used for understanding issues with a driver or the system as a whole. These dumps are usually larger and can often be half a gigabyte to a gigabyte in size or more.
- **Complete System Dump:** This dump is not always available by default as an option. It can be offered by setting a registry key value and rebooting the system. This dump contains everything that was happening in the kernel and all the processes that were running at the time of the failure. To capture this type of dump, you must configure a page file. Set the page file size to equal the physical size of the system RAM plus 300 MB for the additional register information stored on the CPU. For example, if you have a system with 2 GB of RAM, set the page file to be a fixed size of at least 2,348 MB. The resulting dump file might be that size, but it could also be less depending upon how much was running at the time of the failure.

There might also be times when a driver is not causing an exception that causes the system to fail. However, you might want to capture the system state. Usually, in these situations, the system stops responding to all commands, but certain parts still seem to be functioning. For example, you might still see the clock updating or the mouse cursor moving, but every other command is ignored. In those cases, you might want to use *notmyfault.exe* or a keyboard method to manually stop the system once you have configured it for the appropriate dump.

Initiate the Manual Capture of a Dump

1. Set the registry to enable a complete dump. Read the knowledge base (KB) article for instructions: http://go.microsoft.com/fwlink/?LinkId=690061.
2. Create a complete memory dump file.

   a. Click **Start**; right-click **Computer**, and then click **Properties**.

   b. On the **System** page, click **Advanced system settings**, and then click the **Advanced** tab.

   c. Under the **Writing debugging information** area, click **Settings**, and then ensure that **Complete memory dump** is selected.

   Note By default, **Complete memory dump** is disabled. You can only enable the option if your computer has more than 2 GB of physical RAM.

   Note If you want to enable the **Complete memory dump** option, manually set the **CrashDumpEnabled** registry entry to *0x1* under the following registry subkey, and restart Windows: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl

For more information, go to http://go.microsoft.com/fwlink/?LinkId=690062.

Important For the steps above to work properly, you must set the system page file to be equal to the size of RAM + 300 MB. Therefore, if you have 4 GB of RAM in the system, then 4,192 MB + 300 MB should be safe.

Note In addition to telling you where to download NotMyFault.exe, this KB article (http://go.microsoft.com/fwlink/?LinkID=690061&clcid=0x409) mentions the registry keys for enabling the Ctrl + Scroll Lock + Scroll Lock mechanism. With this mechanism, you can generate a manual dump using the keyboard:

Generate a Manual Memory Dump Using NotMyFault.exe

If you can log on while the problem is occurring, you can use the Microsoft SysInternals NotMyFault tool.

1. Download the NotMyFault tool from the following Microsoft website: http://go.microsoft.com/fwlink/?LinkId=690208.
2. Click **Start**; right-click **Command Prompt**, and then click **Run as administrator**.
3. At the command line, type *NotMyFault.exe/crash*, and then press Enter.

Note This will generate a memory dump file and a *Stop D1* error.

Generate a Manual Memory Dump File Using the Keyboard (PS/2 Keyboard)

Windows features let you generate a memory dump file using the keyboard. If you are using a PS/2 keyboard, you have to create the *CrashOnCtrlScroll* registry entry. For more information about how to generate a memory dump file using the keyboard, go to http://go.microsoft.com/fwlink/?LinkId=690063.

Generate a Manual Memory Dump File Using the Keyboard (USB Keyboard)

The instructions below assume that you are not running a write filter when attempting to generate a dump of this type. You might need to generate this type of dump file while running a write filter. If you do not want to download the file using a cable connection, provide another drive partition that is not protected. Specify that the dump file and page file should be located on that unprotected partition. As an alternative, you can specify what is known as a dedicated dump file location; learn more at http://go.microsoft.com/fwlink/?LinkId=690064.

You must create the CrashOnCtrlScroll registry entry on the Windows 7-based computer for this feature to work. To enable the feature on a computer that uses a USB keyboard, follow these steps:

1. Start Registry Editor.
2. Locate and then click the following registry subkey:
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\kbdhid\Parameters
3. On the **Edit** menu, click **Add Value**, and then add the following registry entries:
   - Name: CrashOnCtrlScroll
   - Data Type: REG_DWORD
   - Value: 1
4. Exit Registry Editor.
5. Restart the computer.

   Important On a computer that uses a USB keyboard, you do not have to restart the computer. Unplugging the keyboard and plugging it back again is sufficient. After that, the memory dump file can be generated.

Note The keyboard operation will generate a memory dump file and a *Stop E2* error.

To read more about configuring the keyboard registry keys, go to http://go.microsoft.com/fwlink/?LinkId=690065.

If you do not have access to the system locally, but need to generate a failure, consider using PsExec to trigger NotMyFault on a remote system. This is useful in a lab situation where systems might not have physical keyboards or displays attached.

Before you upload or share any of the dump files, compress them into a Zip file. Compression can generally reduce the file size by almost a factor of ten and save you a significant amount of upload time.

Visual Studio

As was mentioned earlier, you should not install a full copy of Visual Studio on an embedded Windows client system. However, you can use Visual Studio and remote debugging to debug a process while it is running on a client system. To learn how to set up your environment and Visual Studio 2013 for this remote debugging process, go to http://go.microsoft.com/fwlink/?LinkId=690067.

Troubleshooting

When you encounter an issue on an embedded system, attempt to reproduce the problem on the desktop equivalent OS. We discussed installing your embedded system on the desktop OS equivalent in the Hardware Validation section. Plug the hard disk that contains the full desktop OS installation into the embedded system. Configure the desktop OS equivalent installation in the same way as the embedded OS, and see if you the problem appears. If you do see the issue, look for a solution for the Windows issue or bug that is causing the problem. If you cannot find a resolution on your own, contact Microsoft for support for that desktop version. If you cannot reproduce the issue on the desktop OS version, you might be missing a required or optional component in your current OS configuration.

Common Windows Embedded Issues

POSReady 2009/WePOS 1.1/POSReady 7

Earlier, we mentioned that FBWF and the Registry Filter are present in the POSReady versions. If you are not using them, it is probably safer to deactivate them than leave them running. They are found in the following registry location, and the value of 4 sets them to disabled so they will not start:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Regfilter]
   "Start"=dword:00000004                    // = disabled

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FBWF]
   "Start"=dword:00000004                    // = disabled
```

These registry key locations should be the same on newer versions, such as POSReady 7.

Windows 7

The issues discussed in this section may also pertain to WES 7, Thin-PC, and POSReady 7 systems. In all these versions, in **Control Panel** > **Programs and Features**, the section for **Turn Windows Features On or Off** will always be blank. This is by design for these embedded products. You can use DISM/Add-Package commands online or offline to add or remove additional features or drivers for an existing image.

There are a few differences between Windows 7 and the embedded versions regarding power settings. Additionally, only certain user accounts have access to or can persist new power plan settings. First, an administrator can change some security descriptors to allow local users to modify the power settings on the embedded system. To make this change, use an administrator account to execute the following command from a command prompt on the device after the image has been deployed:

```
powercfg -setsecuritydescriptor actiondefault
O:BAG:SYD:P(A;CI;KRKW;;;BU)(A;CI;KA;;;BA)(A;CI;KA;;;SY)(A;CI;KA;;;CO)
```

Second, there is a difference in the default *Balanced* power settings for the *AcSettingIndex* value on the embedded system. This registry value defines the point at which the CPU will accelerate to high performance. On WES 7/POSReady 7, the default value is 90%, but on Windows 7, the default is 60% of CPU utilization. Because of this difference, you might observe performance differences when running applications on embedded systems out of the box. You can change the registry key so it contains a value of 60 instead of 90. Alternatively, you can select the **High Performance** power scheme instead of the **Balanced**. You can modify the following registry key for the Balanced power scheme:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\PowerSetting
s\54533251-82be-4824-96c1-47b60b740d00\06cadf0e-64ed-448a-8927-
ce7bf90eb35d\DefaultPowerSchemeValues\381b4222-f694-41f0-9685-
ff5bb260df2e\AcSettingIndex
```

Process Monitor

When you suspect that you might be missing an OS component or dependency, one of the best options is to use Process Monitor. It is a free utility that, once installed, can fully monitor all file and registry accesses on your system. It can capture quite a lot of information in a very short time. With some practice and learning how to filter the log, you can view only the processes in which you are interested. Once filtered correctly, it should be easy to identify the file or registry key that is missing or might have different permissions.

If you are having trouble identifying where the error occurs, conduct an A vs. B comparison. To do so, capture a log of a similar run of the application on a system that does not have the problem. This might mean obtaining a capture from a desktop version while comparing it to your embedded system. To download a copy of Process Monitor, go to http://go.microsoft.com/fwlink/?LinkId=690068.

System Updates (WU, WSUS, WEDU, SCCM)

Once your system is deployed to the field, you might want it to receive periodic updates for functionality and security. The need for updates will depend on how and where the system is being used. If you create a stand-alone information kiosk that never connects to a network, updates might not be needed as regularly. For a device that operates on a network or as part of an enterprise domain, regular security updates are very important for the system and overall network security.

Your OS design, or use of a write filter, can complicate the process for applying updates to the system when compared with a normal desktop client. If an update is installed on a partition that is protected by a write filter, on the next reboot, that update is lost. For this reason, you must first disable the write filters protecting any system partitions before applying updates. You can also set the write filters to servicing mode before applying updates.

There was a small utility service known as Device Update Agent included in the Windows XP Embedded-based products. OEMs could use it for deploying custom update packages. However, there are several available alternatives, and Device Update Agent is not widely used anymore.

System updates are designed to keep a backup copy of the binaries that are updated so users can uninstall them if a problem occurs. On a regular desktop system with hundreds of GB of drive space, this is not an issue. If you deploy a 4 GB OS image to an 8 GB drive, you might easily experience drive bloat from system updates over time. At some point, you will no longer have any free space. This could happen in only a year or two of updates. Because of this fact, we recommend that you ship with a minimum of double your image size in free space. However, three or four times your image size would be ideal for a higher safety margin. For example, if your image is 4 GB, shipping on a 12 or 16 GB drive would be a good option. If your image is 8 GB, consider a minimum of 16 GB or use a 32 GB drive. You can run disk cleanup on some versions to delete intermediate versions of the updates. You will still have at least one duplicate version of every updated binary still present.

Windows Update (WU)

All versions of Windows Embedded, except XP Embedded SP3 and WES 2009, can include Windows Update (WU). With WU, if they are on a network with access to the Internet, they can download updates directly from Microsoft. If no write filter is present, then this process is almost identical to a desktop Thin Client. If you are using a write filter, some customization is required to make this work properly. This customization might include some scheduled tasks to disable the write filter and reboot the system before downloading and applying any updates. In some versions, such as XP and Windows 7, the WU component is optional. However, in all versions, you can set the WU service to be disabled once the system is deployed if needed.

WU and Windows Server Update Service (WSUS) will detect and install updates for Windows desktop components and any installed versions of the .NET Framework(s). However, that is not true for embedded clients. The only version of .NET that will be detected and automatically installed is the version that shipped in-box with the product. For WES 7, that means WU will detect and apply updates for .NET 3.5 SP1. If you have installed .NET 4.0 or 4.x, those updates will not be detected for you. To help

OEMs determine if .NET updates are needed for an embedded client, Microsoft has created a spreadsheet.

Another limitation of Windows Update is that it will not detect or update any of the EEFs/Lockdown features present in the image. For that, you must use Windows Embedded Developer Update (WEDU) to install those updates when you build the image. You can also download each individually from MyOEM or Download Center.

Windows Server Update Service (WSUS)

Some enterprises might not want to connect their systems directly to the Internet. In those cases, an enterprise might plan a more scheduled deployment of its own once updates become available. You can use WSUS to deploy the same update packages you would download directly from WU to the client. WSUS can also create some level of custom deployment scripts for third-party packages if needed. This is useful if you are deploying an OOB component (such as an updated version of Internet Explorer) or your own custom application. For Volume License customers, you will find OOBs for some of the POSReady clients by going to http://go.microsoft.com/fwlink/?LinkID=671707&clcid=0x409.

There is a free tool you can use to author a package for deployment using WSUS; go to http://wsuspackagepublisher.codeplex.com/.

System Center Configuration Manager (SCCM)

System Center Configuration Manager allows you to install a small client on your embedded systems that allows you to deploy custom, scripted updates as needed. The first version capable of supporting embedded clients was SCCM 2007 SP2 using an add-in called *Windows Embedded Device Manager 2011 SP1*. This SSCM version allowed customers to manage embedded clients that were based on XP and all the other versions up to and including Windows 7. Most customers are now running System Center Configuration Manager 2012 R2 SP1. As of the R2 version, SCCM added support for Windows 8 and 8.1-based embedded clients.

If you plan to use SCCM to manage your devices, there are exclusions a configuration manager client needs when using FBWF or UWF. To learn more, go to http://go.microsoft.com/fwlink/?LinkId=690070.

There is also a utility to help create update packages. To download the utility, go to http://go.microsoft.com/fwlink/?LinkId=690071.

Windows Embedded Developer Update (WEDU)

Windows Embedded Developer Update must be installed on the development system where you have installed ICE for building your images. WEDU can detect and download updates for you so they are added to the distribution shares present on your development system. WEDU downloads regular Windows updates and updates for the EEFs or Lockdown features as well.

Resources

This paper has tried to discuss several introductory topics and provides a good start. However, it is not comprehensive enough to replace the product documentation or training that exists. In this section, we will discuss some of the options for additional information or ways to get support when developing a new product.

Microsoft Developer Support

If you need help with a problem you encountered during development of a Windows embedded system, you have a few options available to you. It is important to determine the exact type of support you need. For example, if you are having a problem with an application, consider requesting support with Visual Studio and your development language (C++, C#, Visual Basic, etc.). If you are developing a custom device driver, request support from the Windows Driver Kit SDK support team. As a reminder, first develop and test on the desktop equivalent version OS to ensure everything works correctly before testing on an embedded client.

For general support issues, go to http://go.microsoft.com/fwlink/?LinkId=690072.

For Windows embedded OEM support, which is generally handled by Developer Support, you have several options. To learn about these options, go to http://go.microsoft.com/fwlink/?LinkId=690073. This site includes links to search the Knowledge Base and the forums, view developer documentation, or contact Microsoft for additional help.

If you need to open a support request with Microsoft, there are a few options available to you. The first is to open a paid incident request; you pay a fixed cost on your credit card. To start that process, simply type the name of the product you want support with on the follow page: http://go.microsoft.com/fwlink/?LinkId=690074.

An MSDN subscription is a second option for requesting help. The subscription might include a certain number of support incidents per year. To learn more about the benefits available to MSDN subscribers, go to http://go.microsoft.com/fwlink/?LinkId=690075.

Finally, a Premier Support contract with Microsoft is another option. If your company has an unexpired Premier Support contract, you can contact your Technical Account Manager (TAM). The TAM will assist you with creating a support case. To learn more about this option, go to http://go.microsoft.com/fwlink/?LinkId=690076.

As the OEM who develops and sells an embedded image, the expectation is that you will provide support to your customers for any issues they encounter. Microsoft does not directly support end users of embedded systems. If they contact Microsoft, they will be directed back to their OEM for assistance. Volume License customers that are using an OS like POSReady 7 can contact and work directly with Microsoft because there is no OEM involved.

Training

If you want more in-depth training on how to develop and configure a particular version of Windows embedded, there are a few options available. If you have a Premier Support agreement, work with your TAM to arrange for Microsoft to deliver a two-day training session or hands-on lab. Alternatively, you can contact your Embedded License Distributor to discover if they offer training. The distributor may be able to arrange training with a Microsoft representative at its facilities.

Documentation

Each Windows embedded toolkit includes documentation. In addition, the Microsoft Developer Network (MSDN) offers documentation for all versions of the embedded products that have been released. The following is a list of a few of the available sites:

- **Windows Embedded Standard:** http://go.microsoft.com/fwlink/?LinkId=690077.
- **WePOS/POSReady:** http://go.microsoft.com/fwlink/?LinkId=690078.

- **Windows Embedded 8.1 Industry:** http://go.microsoft.com/fwlink/?LinkId=690079.

There is also a very good Windows Embedded Standard 2009 resource kit that discusses development with that product. A majority of what is discussed there is also applicable to Windows XP Embedded development as well. Some concepts are also applicable to newer versions.

Some third-party authors have also written books discussing how to configure and use Windows Embedded. If you require more documentation than what Microsoft currently provides, consider purchasing a book that discusses your target Windows Embedded version.

Conclusion

A variety of approaches and options exist when creating an embedded device. We hope that this paper has served as a good introduction to what Microsoft has to offer in the embedded space. With other systems, you might need to learn a whole new process or development approach. With Windows Embedded, you can continue to utilize your existing knowledge of Windows and Win32 development. Hopefully, you now have a clear idea of the available versions, the Embedded Enabling/Lockdown features, and how best to use them to your benefit in your system design.

Good luck in creating your new embedded device that will help to improve the lives of your customers.