Windows 10 IoT Enterprise: Why No Components Like Previous Windows Embedded Releases?

By Sean D. Liming
Edited by John R. Malin
Annabooks – www.annabooks.com

February 2020

Windows 10 Version 17763 Build 1809

For those migrating from Windows XP Embedded and Windows Embedded Standard 7 to Windows 10 IoT Enterprise, aka Windows 10 Enterprise LTSC (Long-Term Servicing Channel), there are three things that are noticed:

1. The build process is completely different.
2. The lockdown features have changed significantly.
3. The customization through the selection of components to shrink the operating system can no longer be done.

The last point about the loss of components has developers trying to wrap their heads around Windows 10's size. Some of my clients want to support older hardware platforms, but there is a lack of drive space. The cost of a hard drive to fit the system can be prohibitive. If the system is in the field, some clients have had to consider costly service calls or hardware recalls just to swap hard drives. Others, like slot machine OEMs who have to do a CRC check on boot, don't want a big OS footprint because it will take a long time to do the CRC check. Of course, there are many others who have, enough hard drive space but want to limit what services the OS is running. In this paper, I will take a step back in time to explain how we got here, discuss some ideas to shrink Windows 10 and look at what the market is really looking for.

Image Size Comparison

As of today, the approximate minimal footprint image sizes for the Windows Embedded/IoT releases are as follows:

Windows OS	32-Bit Minimal Image Size	64-Bit Minimal Image Size	Componentizes / Package
Windows NT Embedded	~9.5MB	N/A	Yes
Windows XP Embedded	~75MB	N/A	Yes
Windows Embedded Standard 7	550MB	830MB	Yes
Windows Embedded 8 Standard	~2.7GB	~4.4GB	Yes
Windows Embedded 8.1 Industry	~3GB	~5.6GB	No
Windows 10 IoT Enterprise	~7GB	~13GB	No

These are some big jumps in a minimal footprint. Embedded/IoT systems last a long time. I have many clients that are still using XP Embedded and are looking to create their next product. The jump to Windows 10 IoT Enterprise is a big step for them.

How Did We Get Here?

Let's go all the way back to Microsoft's entry into operating systems. MS-DOS and Windows 3.1 were the top operating systems when I joined Annabooks/Annasoft in 1995. MS-DOS was small enough to be put into a PROM card, DiskOnChip, and there were a couple of MS-DOS ROM versions available. Windows 3.1 added the GUI but was still relatively small in size at the time. Windows 95 was soon released; and even though Windows CE was on the horizon, Windows 95 started to gain momentum for those using embedded PC systems. The only problem was the size. Back then, flash drives were in their infancy and very expensive, so getting the footprint down was a critical design priority. Developing a build tool was not practical back then, so I wrote the paper: *Minimizing the Windows 95 Footprint for Embedded Systems Applications*. It was a manual process to remove unneeded files from the OS. The paper is still floating around the Internet. About 20MB in size was the low footprint size I was able to achieve at the time.

Windows CE was launched in 1996. As a small 32-bit ROMable OS, Windows CE took off in popularity; but it had a completely different kernel than Windows desktop. This led to different development processes between Windows CE and Windows desktop. Windows CE was smaller in footprint and componentized to support onboard flash drives. As Windows desktop moved to support ARM processors, around 2008, Windows CE's 17-year run came to an end.

Soon after Windows CE's release, Microsoft acquired Component Integrator from VenturCom, which became Windows NT Embedded. I remember one Embedded System Conference in Boston where attendees would walk by the booth and chuckle at the idea of using Windows NT in an embedded system, but NT Embedded found a home in a few devices. NT Embedded was broken into about 230 components. There was no-PnP at the time, so building and deploying the image was relatively simple. The following embedded features were added to NT Embedded: write filter, headless support, CD-ROM boot, serial remote administration, and system message intercept. You could build workstation or server images. The image size ranged from 9.5 MB to 40MB, which was large but still fit the drive sizes of the time. With the modest success of Windows NT Embedded, Microsoft put a large effort into creating the next offering: Windows XP Embedded.

XP Embedded took componentization to the extreme by breaking Windows into 10,000+ components. 9,000 components were for device drivers, and 1000 were OS components. Many of the OS components were for a single DLL file. Knowing how to build the image became a challenge. Go down the wrong path, and the resulting image would end up with a BSOD Stop 7B error. The Stop 7B problem was so pervasive that I had to write a white paper and update it a few times to help developers to resolve the issue. A streamlined image development process took me several years to develop. Much of this streamlined process I still use today with Windows 10, and it keeps evolving.

The development process came as a result of having to deal with so many components. The biggest problem was trying to figure out what components were needed to get a feature to work. I had a client who wanted the screen saver working, but they also wanted a small image. One would think that dropping in the screen saver component was all that is needed; not even close. I had to build a full OS image to prove that the screen saver was working in XP Embedded, and then compare the component list between the big full-featured image and the smaller desired image. The final answer after many OS rebuilds was that the "Windows XP Visual Styles" component had to be included, which I would have never figured out based on the name. Locating many of the features became tribal knowledge within the community.

The experience made me create some tools and solutions to speed up the process. First, I created the Full XP Pro-like Solution macro component set. This allowed you to build a full XP Pro image from XP Embedded tools. Second, I created a few tools to help with development. One tool being the SLDXAssistant, which compared SLX and SLD files. SLDXAssistant could compare the build files from the Full XP Pro-like Solution with a smaller build file to locate the differences. From there a little sleuth work was needed to find the missing components. Sometimes it took some iterations, but eventually, features were found.

Regardless of the design challenges, Windows XP Embedded was popular and successful. One could get an image under 75MB to simply talk to serial ports. The Full XP Pro-like Solution would become a big problem, though. Developers started using the solution to create their images and never bother to look at shrinking the image. There were many times I was hired to help a development process that went afoul by developers using the Full XP Pro-like Solution as is. Having programs like Messenger, Outlook Express, IIS, and Games was not desirable in the embedded OS, and the approach to start big and go small wasn't cleaning up the mess. The Full XP Pro-like Solution was only intended as a means to help to track down features, and it was not intended for being the bases for the image. The ramifications to Windows componentization would come later.

XP Embedded's componentization was difficult for some companies, like point of sale vendors, just needed a Windows OS to quickly install onto their devices. Microsoft came out with Windows Embedded for Point of Services. Rather than building an image from components, WEPOS was a simple DVD install with a Wizard that had some selectable features. WEPOS created a fork in the embedded offering, and its success influenced future decisions on componentization.

During the growth of Windows XP, Microsoft was going through some legal issues. In the late 1990s. Microsoft was sued by the US Government for predatory practices, which stemmed from the Internet Explorer browser and other software being included in Windows. The EU soon filed a similar antitrust case in 2000. In the EU case, the issue focused on Media Player. RealNetworks, with its RealPlayer®, felt they were at a disadvantage. Microsoft argued that Media Player couldn't be removed. RealNetworks used Windows XP Embedded to build a Windows image without Media Player. Microsoft was proven wrong using their own product. Since Windows Embedded sales were very tiny compared to Windows desktop sales, the leadership at Microsoft at the time started to look unfavorably at Windows Embedded.

The code modifications required to take Windows XP to the next version of Windows, Windows Vista, turned out to be a herculean effort. Major changes were required to the kernel, file system, foreign language support, security features, device drivers, and the boot process just to name a few. There was an effort to make an embedded version of Windows Vista, but there were so many problems with the rollout of the desktop version that Windows Vista Embedded was never released. The work that was done attempting to create a Windows Vista Embedded, however, paved the way for future releases of embedded Windows.

After a long wait, WES7, Windows Embedded Standard 7, was released in 2009. This time, Windows was broken down into bigger packages. The Stop 7B issue was resolved with a foundation package that had the core functionality for a successful boot-up each time. The image development process was much easier, but the image sizes were bigger. The smallest image size was 550MB and the largest image size was around 3 to 4GB. Developers could build a full Windows 7-like solution minus games, but this time Microsoft supplied the Application Compatibility macro component. The rationale for the increase in packages and image size was that flash drive sizes were getting bigger and cheaper, so there was no need to break the OS down into smaller components like XP Embedded. This didn't make some folks happy at the time who wanted to move their current XP Embedded system to WES7. The increased size complaints were not too loud, since Windows 64-bit support was now available. Windows XP Embedded only supported 32-bits. The expectation of 64-bit platforms requiring new hardware, more RAM, and larger drive size was understood.

Besides WES7, POSReady 7 was released as the successor to WEPOS. You could still install the OS from a DVD as before, but you could also use the same System Image Manager from the ADK that was used for regular Windows Desktop and Server to create a custom OS installer.

Around this time, Apple's iOS and Google's Android for smartphones and tablets started to change the game in the mobile space and how consumers purchased software via application stores. Microsoft scrambled to play catch up with a move away from Windows CE OS to real Windows on ARM for smartphones.

Eventually, Microsoft's internal reorganization merged the Windows Embedded team with Windows Desktop. The Windows Embedded folks faded away as the Windows Desktop team took over. One of the last products from the Windows Embedded team was the ill-fated Windows Embedded 8 Standard, which was more closely aligned with WES7 in package size. Attempts to build the tools and allow developers to create packages were a nice idea, but this was Windows 8. The desktop team's dominant influence came in the form of requiring activation for each of the units shipped, which didn't go well with almost anyone working on Embedded Systems. WES8 was the last componentized Windows OS.

With pressure to catch up to Apple and Google, the Windows desktop development team was working on what would become Windows 10, was given several goals for this next version of Windows. The first was to create an OS that could scale from small devices like the Raspberry PI to phones, to desktops, to servers, and to HoloLens. The second was to deliver Windows as a service (Waas). The third was to create a unified application model that would allow applications to run on all these devices. There are over 15,000 developers at Microsoft who work on Windows 10. The coordination with such a massive group requires a clear development process with several checks before a release is made. The Windows Embedded team took Windows, did the component break down, and had to do all the testing independently, which had a cost. Offering a componentized Windows 10 version would only add to the development cost. Offering the full release as Windows 10 Enterprise LTSC with all of the embedded lockdown features makes it easier for Microsoft to develop and test. Also, the image development process is the same for all desktop versions and easier since there is no guesswork on what components to include.

When asked if there would be a componentized Windows 10 release, the responses were as follows:

- We found out that developers were building full releases of Windows XP Embedded and Windows Embedded Standard 7, so why should we componentize Windows Embedded? Windows POS Ready is doing great without components.
- Flash drive sizes are in the gigabytes, now; so there is no need to break the OS down into components.
- The extra cost of componentizing didn't add up to be worth doing.
- We are never going to consider components again, so don't ask. (This response appears to be the result of the lawsuit.)
- Azure is the breadwinner now.

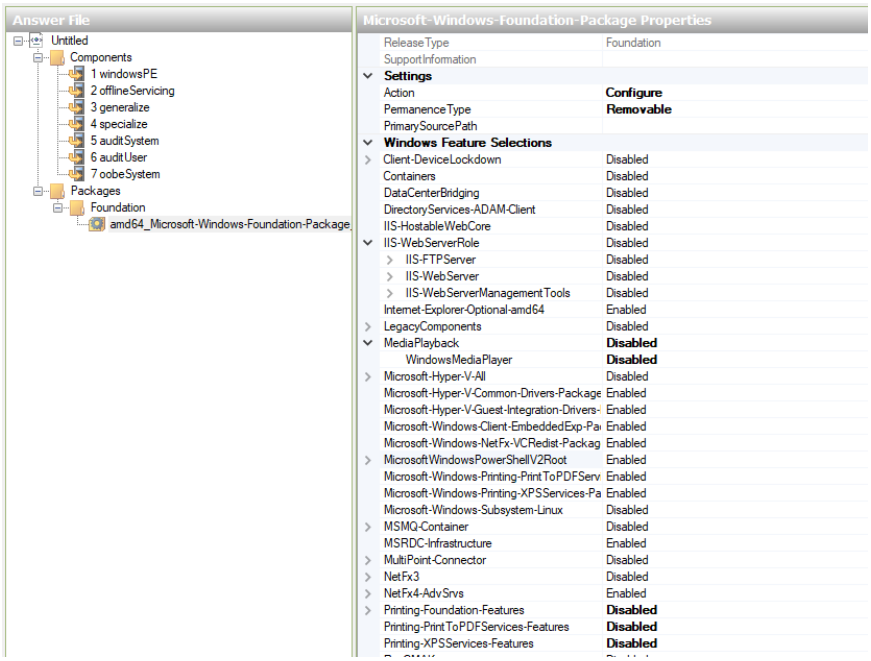
If you didn't know the history of or the effort that goes into releasing Windows, the final decision not to componentize Windows 10 comes across as being made by a few folks at Microsoft who did not have experience with the Windows Embedded market nor the building of Windows Embedded images, listening to a small number of inexperienced Windows Embedded users and using that information to justify not to componentize Windows moving forward. There might be some truth in this from a certain point of view; but looking back, there are a number of factors that forced development priorities to where we are today.

The core challenge is, as Microsoft releases new features, the operation system grows in size. For example, Mixed Reality was a big deal in one of the Windows 10 releases. Mixed Reality is a cool technology, but it is not needed in ATM machines, cash registers, and most embedded systems. After a couple of years, the component issue was brought back up again, and this time Microsoft has heard the feedback. Many new features and applications are being put into the Features on Demand (FOD) and the in-box applications disks. This way the core image doesn't have these optional features. It doesn't solve the problem, but it slows down image size growth.

Can anything be done for Windows 10 Enterprise LTSC? Here are a few tips:

The ability to reduce image size and remove features is very limited. There are a few things that can be done to help reduce the image size:

- Disable Features – Using System Image Manager to help create a custom Windows Installer, add the Foundation Package to the answer file. The Foundation Package provides options to enable/disable Windows features. Internet Explorer, Windows Media Player, XPS Print, Hyper-V, etc. can be enabled or disabled via the Foundation Package settings. This will reduce the image size by not installing the feature, but disabling a feature doesn't remove the installation package for that feature from the image, which is buried in the \windows subfolder.



- Turn off Hibernation and Disable Virtual Memory – the hiberfil.sys and pagefile.sys files can take up several gigabytes on a disk. If these are not needed, then disabling them can save on space.

- Run **DISM /Online /Get-packages** to get a list of packages in the OS. There will not be many, but use the list to remove unneeded packages.

- Remove a copy of the Windows custom installer – When installing from a USB flash drive, a copy of the installer is put into c:\windows\ConfigSetRoot. You can remove the copy of the installer using the following sync command in the answer file:

  cmd.exe /C rd /S /Q c:\Windows\ConfigSetRoot

- The compact utility can be used to free up space:

  compact /compactos:always

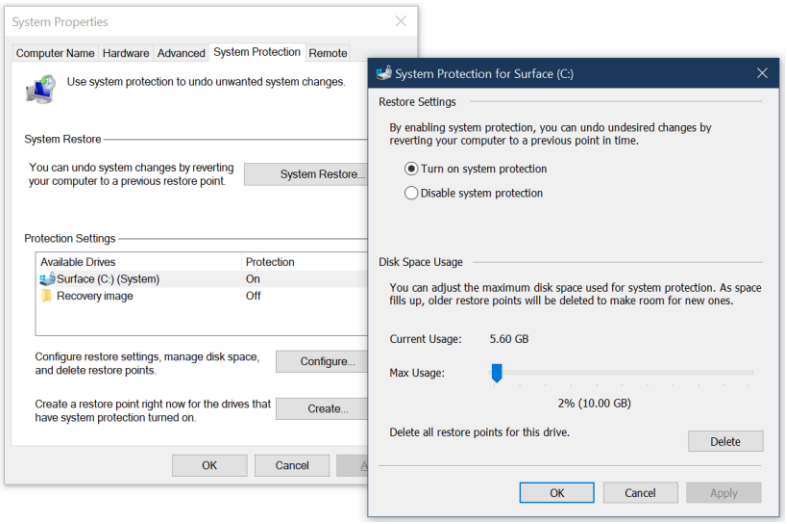
- Sideload UWP (Windows Store) Apps – You can optimize after these applications have been installed via DISM by running the following command on an offline image:

  DISM <path to offline image> / Optimize-ProvisionedAppxPackages

Any duplicate APPX file resources are converted to hard links.

- Windows Update Process – Running Windows Update to make sure the latest security updates are in the image is important to make sure the OS is secure. The updates will increase the image size. These are some steps to clean up the impact:

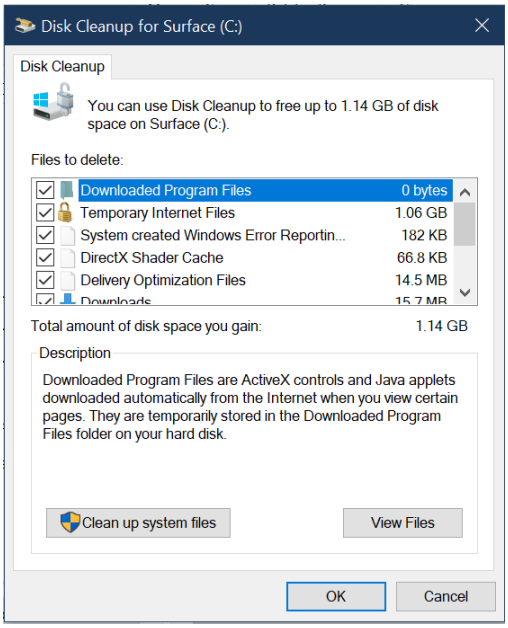
  - In system control panel, you can remove restore points



  - Run dism to perform some package cleanup:

    Dism.exe /online /Cleanup-Image /StartComponentCleanup /ResetBase

  - Run Disk cleanup



Windows 10 IoT Core Conundrum

With Windows CE gone, there was no small footprint Windows OS from Microsoft. Since there was a big push for developers to create UWP applications from the new Microsoft management team, the OS team came up with a solution to help fill the gap: Windows 10 IoT Core. The Core version

is still Windows 10, but the Win32 GUI support has been removed. UWP applications provide the GUI interface, and Win32 applications can only be command line or headless applications. Taking out Win32, the resulting image size is approximately 2 to 3 GB. IoT Core has some components to add features to the image.

The problem with IoT Core is that Microsoft pushed Win32 for such a long time that customers have a large investment in Win32 applications and developers. There was no clear path going from Windows CE to Windows IoT Core. Windows Phone failing and UWP following suite didn't help matters. The UWP only support might have sold Microsoft management, but IoT Core hasn't grabbed the attention of the market like Windows CE did when it was first introduced. There are some really good IoT Core projects that have been done, but the interest has been small. IoT Core really wants to be the new Windows OS for the new generation of small embedded/IoT systems, but it doesn't meet the market needs in its current state.

Not so Green Operating System

Technology has led to CPUs with better power and performance, but the size of Windows 10 IoT Enterprise requires increased amounts of RAM and certainly big storage space on the boot disk. Designing in storage space that is being used to store features and capabilities that are not being used by a dedicated system is not an economical design. The target hardware is not the only issue. There is also an image development impact. Transmitting large images between developers takes upload and download time. Storing large images, storing different image iterations, and backups of the development files require gigabytes to terabytes of storage. Microsoft recently announced that they are going to lower their carbon footprint. One can only hope that their products will also follow this goal.

Maybe Hope for a Change in the Future

Customers would like to continue with their current code base and use Windows, but the size is a factor. Windows 10 IoT Core shows that it is possible to have a small Windows 10 image through the use of components, so there is hope, but the lack of Win32 GUI support is holding the Windows IoT Core product back.

WinUI, combining Win32 and UWP, will be a strong influence to force the rethinking of IoT Core. Windows IoT Core could add Win32 GUI support in the form of components. Start small and grow big, which would bring us back to a more traditional Windows Embedded product approach.

It is good to hear that Microsoft is aware of the size issue and is looking to put features in separate DVDs. How do 15,000 developers develop individual items and then integrate them into a whole? They must be using components internally, but that is only a guess on my part. With Windows development being such a large undertaking with coordination that boggles the mind, could a customizable, componentized version of Windows 10 be made available? Only time will tell.