Architectural Comparison of Windows Embedded Standard 7 vs. Windows Embedded Standard 2009

Introduction

This paper describes some differences between Windows Embedded Standard 2009 and Windows Embedded Standard 7 with regard to componentization, tools, and image-building processes. The goal is to give customers a high-level understanding of the differences between the two embedded products, and providing an understanding of some different behaviors in features, tools, and overall user experience in building and deploying embedded-device images.

Componentization Comparison

In this paper we will examine the following areas to compare some differences between Windows Embedded Standard 2009 and Windows Embedded Standard 7:

- Component architecture
- Feature-set packages
- Driver packages
- Language packages
- Component dependencies
- Macro components versus templates
- Settings Management Infrastructure (SMI) settings versus configuration settings
- Embedded enabling features (EEFs)
- Support for customized components

Component Architecture

Both Windows Embedded Standard 2009 and Windows Embedded Standard 7 have a similar componentization concept, namely a set of binaries (files, and so on), specified in a wrapper file together with other information such as registry keys, dependencies, and other resources. All of this data is installed in the run-time image as a group. However, beyond the componentization concept, the implementation and servicing of components is very different in each product.

For Windows Embedded Standard 2009, the implementation of a component was created from scratch because the original binaries it inherits from Windows XP Pro are not componentized. Therefore, Windows Embedded Standard 2009 components are not compatible with future versions of the operating system (OS). This includes Windows Embedded Standard 7. Also, these components were designed to be imported into the Windows Embedded Standard 2009 component database only, in

development computers instead of in the embedded target devices. Servicing the devices requires OEM developers to rebuild the complete run-time image with the updated components.

For Windows Embedded Standard 7, the components are inherited from Windows 7. These are specified in component manifests. In addition to some exceptions needed to satisfy certain embedded requirements, all these Windows 7 manifests remain unchanged for Windows Embedded Standard 7 . Therefore, the design maintains complete compatibility between Windows Embedded Standard 7 and Windows 7. Servicing components is similar to that in Windows 7, requiring only creating an embedded-specific update package that can be imported to the Windows Embedded Standard 7 distribution share on OEM development computers, or installed directly on the embedded run-time images.

Another component design variation of Windows Embedded Standard 7 from Windows Embedded Standard 2009 is the concept of embedded core (or eCore). eCore consists of a set of fundamental OS components (kernel, networking, security, some drivers, and so on) that facilitate the booting up of an embedded device with system security and networking capabilities. eCore is the minimum image of an embedded device that enables OEM developers to add other feature sets, drivers, and language packages. This will be described later in this paper.

Feature Set Packages

For Windows Embedded Standard 2009, OEM developers are required to select feature set packages at the component level when they build their device images. Even certain large features (for example, Windows Media Player, Windows Internet Explorer, Microsoft Remote Desktop Protocol, and so on) are implemented as components because they are an aggregation of a large set of binaries. Therefore, there is no clear difference between a component and a feature. In addition, Windows Embedded Standard 2009 contains over 10,000 components. This makes it challenging for customers to select the correct components to build their images.

For Windows Embedded Standard 7, feature-set packages are created to aggregate relevant components (for example, Windows Media Player, Windows Internet Explorer, Microsoft Remote Desktop Protocol, and so on); OEM developers are only required to select the feature sets they want to deploy. Because the number of feature-set packages is kept to a minimum (around 150), the feature-selection process is simpler. This makes it easier to design and build device images.

Driver Packages

For Windows Embedded Standard 2009, each driver is implemented as a separate component. There are about 9,000 individual drivers. This can present a significant challenge to OEM developers when matching drivers to their hardware devices.

Similar to feature-set packages, drivers in Windows Embedded Standard 7 are also provided at the package level. However, to keep the footprint size small each driver is provided as a single package, except for the USB drivers, which are also aggregated and implemented by using an alternative USB Boot package. Essentially, these driver packages are similar to the individual driver components in Windows Embedded Standard 2009. By eliminating some legacy drivers in Windows 7, Windows

Embedded Standard 7 contains approximately 500 individual driver packages. In addition, there is a list of about 100 drivers that are included in the embedded core (eCore) to facilitate basic needs such as system boot-up, network communication, and so on

Overall, drivers are provided in similar granularity between Windows Embedded Standard 2009 and Windows Embedded Standard 7. However, Windows Embedded Standard 7 contains significantly fewer drivers. A list of drivers is included in the eCore.

Language Packages

Because Windows XP was not designed with language-neutral components, it does not include individual language packages to apply in addition to the base-neutral OS. Each OS binary must be fully localized for different languages to meet the needs of different countries and regions. Such a design not only complicates bug fixing, testing, and servicing of OS components, it also significantly increases the OS footprint size for devices that require multiple languages on their images. Windows Embedded Standard 2009 inherits the same language design as Windows XP, except for providing the non-English language resources in separate media. This gives customers a choice to build their device images with one or more languages. However,, this does not eliminate the complication of fixing bugs, servicing OS components, and footprint size issue, as mentioned earlier.

Windows Embedded Standard 7 inherits the same language-neutral design model from Windows 7. Different language packages can be applied over the base-neutral OS. This approach makes it easier to fix bugs, provide service, and manage the size of the memory footprint in Windows XP and Windows Embedded Standard 2009. As with Windows 7, Windows Embedded Standard 7 provides up to 36 fully localized language packages (LPs). However, the number of available language-interface packages (LIPs) will depend on customer demand. Also, the language packages in Windows Embedded Standard 7 will only contain relevant language resources for their corresponding neutral components for each feature set and driver package (and eCore). Therefore, their package sizes will be significantly smaller than those in Windows 7. Smaller language-package size should make it easier for OEM developers to deploy various language packages to their field devices, depending on the specific needs of each.

Component Dependency

Windows Embedded Standard 7 adopts a similar component-dependency concept as Windows Embedded Standard 2009. Unlike Windows Embedded Standard 2009, whose dependencies are expressed at the component level, dependencies in Windows Embedded Standard 7 are expressed at the feature-set package level. The types of dependencies are similar between Windows Embedded Standard 2009 and Windows Embedded Standard 7, as shown in the following table.

Dependency Type	**Windows Embedded Standard 2009**	**Windows Embedded Standard 7**
Direct (Required) dependency	Yes	Yes
Zero or more dependencies from a group	No	Yes
Exactly one dependency from a	Yes	Yes

group		
One or more dependencies from a group	Yes	Yes
None from a group	Yes	Yes
All dependencies from a group	Yes	No (but is covered by direct or "Zero or more")

Macro Components Versus Templates

In Windows Embedded Standard 2009, to satisfy the dependencies to install a certain feature or application, a macro component can be implemented and imported into the component database. The macro component can specify certain configuration settings, and any required or optional dependencies. A macro component is implemented much like a standard component, however, the former does not contain any file. Therefore, OEM developers can use the embedded tool (for example, Target Designer) to change the configuration settings.

In Windows Embedded Standard 7, a similar concept is used. A template is defined to satisfy the installation of a given feature or application. A template specifies a list of feature-set packages that are required for the feature or application. However, a template is not implemented the same way as a standard feature-set package. It does not allow for the specification of configuration settings that can be altered by using the embedded tool (for example, Image Configuration Editor, or ICE).

SMI Settings Versus Configuration Settings

In Windows Embedded Standard 2009, a component can specify configuration settings (for example, defining firewall ports, and so on) and let OEM developers set the desired values for these settings by using Target Designer. Such settings are implemented in the component wrapper file (or SLD) by using HTML as the user interface. The settings can change the behavior of a particular feature (for example, enabling or disabling a firewall port).

In Windows Embedded Standard 7, the only settings that can be manipulated by OEM developers through ICE are the visible and mutable SMI settings in the components inherited from Windows 7. There is no additional setting implemented at the feature-set package level. Therefore, certain behaviors of a feature that are invisible, or not defined as part of the component level SMI settings, cannot be altered (for example, setting a firewall port). This means the OEM development experience will be significantly different from that of Windows Embedded Standard 2009.

Embedded Enabling Features

In Windows Embedded Standard 2009, EEFs are implemented the same way as other components. In Windows Embedded Standard 7, similarly, EEFs are implemented the same way as other feature sets. Windows Embedded Standard 7 is designed to keep parity with Windows Embedded Standard 2009 for EEFs, with several exceptions as listed in the following table.

EEFs	**Windows Embedded**	**Windows Embedded**

	Standard 2009	Standard 7
Enhanced Write Filter (EWF)	Yes	Yes
File-Based Write Filter (FBWF)	Yes	Yes
Registry Filter	Yes	Yes
Message Box Default Reply	Yes	Yes
Custom Shell Support	Yes	Yes (Improved)
RAM Disk Controller	Yes	Yes
USB Boot	Yes	Yes
CD/DVD Boot	Yes	No
SD Boot	No	Under planning
Dialog Box Filter	No	Yes
Edition Branding	No	Yes
Custom Logon Desktop Background	No	Yes
Hide Boot Screens	No	Yes
Web Services on Devices for .NET	No	Yes
Device Update Agent	Yes	No
Power Management Application	Yes	Power Management feature set

Support for Customized Components

In Windows Embedded Standard 2009, OEM developers can implement their own customized components by creating the appropriate SLD files that have the appropriate binaries, and then importing them into the component database. This process gives customized components the capability of extending features and integrating them into the development platform to provide a seamless user experience using the embedded toolkit (for example, Target Designer).

In Windows Embedded Standard 7, OEM developers cannot create customized components or feature sets. Therefore, their experience with the embedded toolkit and development platform will not be the same. However, OEM developers can create customized features (including third-party drivers) that can be put under the "OEM" folder in the distribution share. This enables OEM developers to include customized features in the embedded run-time images, or redistribute them as a configuration set. However, by using this approach the customized features cannot express any dependencies to Windows Embedded Standard 7 feature sets. As a workaround, OEM developers may create templates as mentioned earlier to include necessary Windows Embedded Standard 7 feature sets as dependencies. There is no tool-based mechanism to change settings using this approach.

Modified Windows 7 Behavior for Embedded Needs/Scenarios

In addition to the differences in componentization between Windows Embedded Standard 2009 and Windows Embedded Standard 7 as mentioned above, certain behaviors for Windows 7 features have also been modified to better suit embedded needs. Most of these behaviors are modified by using SMI settings. These are listed in the following table.

Feature	Windows 7 Behavior	Windows Embedded

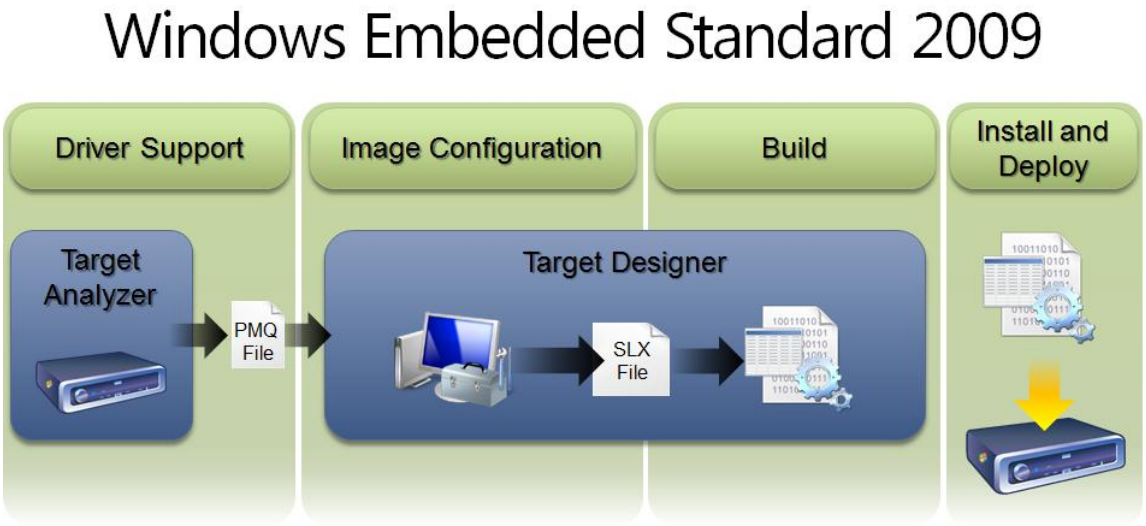
		Standard 7 Behavior
Firewall Notification	Enabled by default	Disabled by default
UAC	Enabled by default	Disabled by default
Paging Files	Enabled by default	Disabled by default
Automatic Update	Download the updates automatically and notify when they are ready to be installed	Disabled by default
Hibernation Enable	Enabled	Disabled

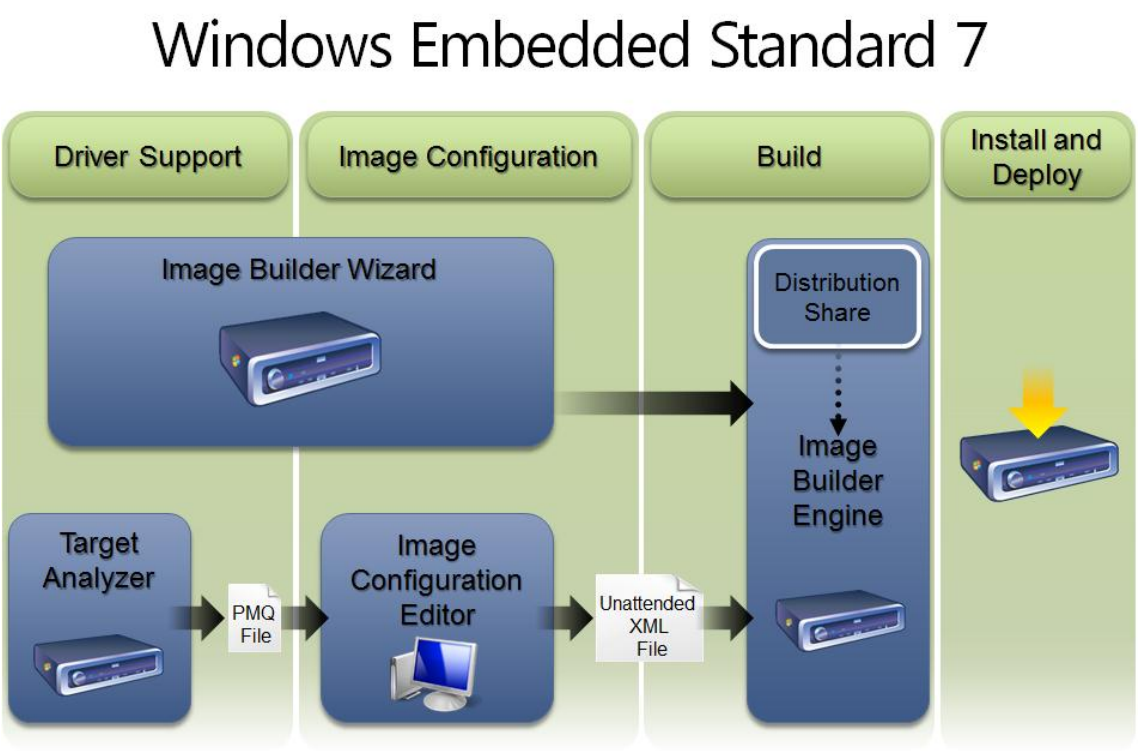
Comparing the Image-Building Process

The process of creating a Windows Embedded Standard OS can be broken down into the following steps:

1. Device Analysis – Determining what devices are on the target computer and adding drivers to support those devices.
2. Configuration – Adding the desired packages for the OS and configuring any associated settings.
3. Building – Forming the OS for the device.
4. Customization – Adding any third-party files or programs to the OS.
5. Capture – Bundling the OS into one cohesive unit for redeployment on other devices.
6. Deployment – Taking the preconfigured OS and installing it on one or more devices.

Shown below are two diagrams that give an overview of the Windows Embedded Standard 2009 and Windows Embedded Standard 7 image building process.





In this paper, each step in the list above is examined for the differences in the development process between Windows Embedded Standard 2009 and Windows Embedded Standard 7.

Device Analysis

In Windows Embedded Standard 2009, device analysis is performed by using the Target Analyzer (Tap.exe) program. OEM developers are advised to boot Windows Preinstallation Environment (PE) on their device and run Tap.exe. By default, Tap.exe will generate a Devices.pmq file, which lists all hardware on the target device.

As soon as the Devices.pmq file is obtained, it can be imported by using Target Designer or Component Designer. Devices will be automatically mapped to drivers, and the components that contains these drivers will be added to the configuration.

In Windows Embedded Standard 7, device analysis is peformed in a similar manner, but is somewhat streamlined. Similarly, Target Analyzer is used to create a Devices.pmq file. The process for running Tap.exe, however, is made easier.

If you are developing images by using the Image Builder Wizard (IBW), Tap.exe is run automatically in the background, and devices are automatically mapped to driver packages. As IBW is executed on the target device, the target hardware can be analyzed directly before build time, instead of as a separate step before you creates the configuration.

If you are developing images by using ICE, the device-analysis procedure is almost the same as with Windows Embedded Standard 2009. Tap.exe can be run from Windows PE, or it can be run from the

IBW disk. The OEM developer can then import the Devices.pmq file into ICE, where each device will be mapped to a driver package and added to the configuration.

Configuration

In Windows Embedded Standard 2009, image configuration is performed in the Target Designer tool. Target Designer is an application that runs on an OEM developer's computer and provides an IDE for creating the Windows Embedded Standard 2009 OS. OEM developers can create a new configuration and add drivers, software, and Embedded Enabling Feature components to create their own customized OS. Developers can also add macro and template components which can contain groupings of component binaries and settings. Developers can also decide to edit settings associated with the OS or its components.

After the basic components have been added, OEM developers must resolve dependencies. This will automatically check the components that were added to the configuration, and add in any additional components that are required for the OS to function. After dependencies have been resolved, the configuration can be saved as a .slx file and used to build OS images.

In Windows Embedded Standard 7, two interconnected development experiences are available. IBW is a wizard-based development tool that lets users quickly and easily prototype device configurations. ICE is similar to Target Designer. It enables the OEM developer to fully control every aspect of their OS.

IBW is a setup wizard that is run on the target device. It can automatically detect devices on the system and install the appropriate drivers. OEM developers can start from a template configuration, or manually select the feature packages they want in their OS. Dependencies are resolved, and when the configuration is completed, IBW installs the OS directly onto the device.

ICE is similar to Target Designer because it is an IDE experience that runs on the OEM developer's computer. Developers can add drivers, software, and EEF packages to their configuration. They can also add templates to configurations to serve as starting points, or they can add a collection of functionality. The OEM developer can also modify settings for the OS and resolve dependencies.

When an OEM developer decides a given configuration is complete, it can be saved to an answer file. This file lists all the packages to be installed, and settings to be set. The answer file is passed on to IBW so that the OS can be built.

Building

In Windows Embedded Standard 2009, building the OS is performed on the OEM developer's computer. After the configuration is finished in Target Designer, the OEM developer can build the image. Target Designer creates the OS folder structure at a build location that is specified by the developer, and populates the folder structure with the binaries and registry hives for the target OS. The OEM developer can then copy these files to the disk on the target device. When the target device is booted, Windows Embedded Standard 2009 goes through the First Boot Agent (FBA). This finalizes the installation of the OS. After FBA is complete, the OS is ready to use.

In Windows Embedded Standard 7, building the OS happens completely on the target device. In both the IBW and ICE configuration methods, the configuration is completed in the IBW tool, and the OS is built. IBW installs the base OS on the device first, then installs the selected packages and applies any configured settings. After the basic installation is complete, the device will restart into the installed OS. Then, much like in FBA, the finalization of the OS will occur, and the OS will be ready to use.

Customization

In Windows Embedded Standard 2009, OEM developers have several ways to add third-party applications to their configurations. The first method is by using the Component Designer tool. This lets developers create custom components that contain the custom files and registry keys for their applications or driversin addition to listing the dependencies on other components. These components are imported into the Component Database and are visible in Target Designer. In this manner, OEM developers can create permanent components which function like any other part of the OS and can be shared between multiple configurations. Custom components are also versioned to allow for revision control. If developers want to add third-party files or registry keys to a single configuration without using a custom component, Target Designer also allows for additional files and resources to be manually added to the configuration using the Extra Files, Extra Registry Data, or Extra Resources nodes. These files will be installed to the device when using this configuration, but will not be available in other configurations, and cannot be version controlled.

In Windows Embedded Standard 7, there is no Component Designer tool. Windows Embedded Standard 7 uses the concept of OEM folders to put third-party files onto the target image. If third-party files must be installed through an installation program, synchronous commands can be used to execute installers during the installation process. OEM folders do not allow for dependencies or built-in version control. However, users can create templates to group feature packages together, and then separate folders for different versions of files.

Capture

After the desired image has been configured and built onto a device, the OEM developer might want to capture the image so that the identical configuration can later be deployed to multiple devices.

In Windows Embedded Standard 2009, OEM developers must use the System Cloning tool, which contains Fbreseal.exe, before capturing their image. During the installation process, each installation is made unique. Fbreseal.exe strips out any unique identifiers so that the installation can be transferred to multiple computers. After running Fbreseal.exe, OEM developers can use third-party tools to capture their images for later deployment. Windows Embedded Standard 2009 also provides limited support for **Sysprep**, which can only be used to prepare an image for use with System Center Configuration Manager's Operating System Deployment method.

In Windows Embedded Standard 7, OEM developers use **Sysprep** instead of Fbreseal.exe. **Sysprep** performs similar functions as Fbreseal.exe, generalizing the image so that it can be captured and then later redeployed. **Sysprep** has the option to force the deployed OS into the Out of Box Experience (OOBE), or Audit mode. This enables either the end-user to configure the OS settings, or enables the

OEM developer to make sure everything is preset before the device is released to the end-user customer. An unattended file can also be passed to **Sysprep** so that additional commands can be executed, or settings configured.

After running **Sysprep**, the image can be captured by using **ImageX** for later redeployment. **ImageX** captures all the contents of a target disk to a Windows Imaging (WIM) file. WIM files are file-based, which allows for significant compression. Duplicate files are only stored in the WIM file one time. Therefore, multiple copies of the same file will not significantly increase the size of the WIM file. Additionally, multiple similar images can be stored within a single WIM file; only the differences between the images increase the overall memory footprint of the WIM file.

Deployment

After the image has been generalized and captured, it can be deployed to production computers. Depending on the environment, OEM developers might have to redeploy the image to one, several, or even thousands of devices. Different deployment methods can be used for each scenario.

In Windows Embedded Standard 2009, no dedicated tools were available to help you in the redeployment process. For low-volume redeployments, OEM developers would just copy the OS files to the disk on the target device. High-volume redeployments would require that you use third-party tools.

In Windows Embedded Standard 7, several deployment options are available from Microsoft. The first, **ImageX**, not only captures files into the WIM file format, but also deploys WIM files to a target disk. OEM developers can use **ImageX** to manually deploy WIM files, or write scripts that use **ImageX** for automated deployment. Contact your Microsoft account team or Distributor for guidance on ImageX usage in recovery scenarios.

The second method for low-volume deployment is through IBW. Using IBW, users can locate a WIM file and complete the installation. In addition to standard WIM deployment, IBW can also add Language Packages to an image being installed, and can even be used to apply an unattended file to an image.

Finally, OEM developers can set up Windows Deployment Services (WDS) or System Center Configuration Manager servers to deploy Windows Embedded Standard 7 images. WDS and Configuration Manager allow for large-scale deployment to multiple devices, and are both fully supported in Windows Embedded Standard 7.

Summary of Tools That Are Used in Each Stage of the Development Process

	Tools used in Windows Embedded Standard 2009	**Tools used in Windows Embedded Standard 7**
Device Analysis	TAP.exe	TAP.exe, IBW
Configuration	Target Designer	ICE, IBW
Building	Target Designer, FBA	IBW

Customization	Component Designer	ICE
Capture	FBRESEAL, Sysprep, third-party tools	Sysprep, ImageX
Deploy	Third-party tools	ImageX, IBW, WDS, Configuration Manager