Windows Embedded

Windows Embedded 8 Standard

Positioning White Paper

Custom Branding

This paper provides an overview and a lab on how to custom brand your device with Windows Embedded 8 Standard.

Introduction

In the embedded world of purpose-built devices, identifying the device through its name, design, and/or functionality is a powerful way to create differentiation and value. Having a consistent user experience is one key aspect of creating a differentiated device. An embedded device typically does not want to show any branding other than that of the original equipment manufacturer (OEM). In this paper we will give guidelines on how to suppress the operating system start, shutdown, and other UI elements while still taking full advantage of the underlining operating system. These are the key focus areas when creating a custom-branded device:

- Boot experience
- Logon experience
- Automatically launch target application
- Hide Windows status messages
- Shutdown

Key Points

1. Control the device experience.
2. Consistent look.
3. Brand recognition.

Boot Experience

The Windows boot process consists of several phases which are explained in more detail by the following blogs:

The Windows 7 Boot Process

Delivering fast boot times in Windows 8

For the purposes of this guide we will simplify things by just talking about what a customer might see during the firmware (BIOS/UEFI) boot process and the operating system boot process before going through to the next step of what is seen during the logon experience.

The normal Windows boot and resume process contains animations and messages shown by the Windows kernel. So the first time Windows 8 or Windows Embedded 8 boots, which is after the Power-On-Self-Test (POST) process, you will see the Windows boot logo, the spinning wheel progress symbol, and possibly a status message, depending on the operating system. See figure 1 below. Obviously this gives away that it's a Windows operating system, which is okay for many embedded devices, but some may still want to suppress this for reasons like brand recognition or the additional security measure of not revealing the underlining operating system.

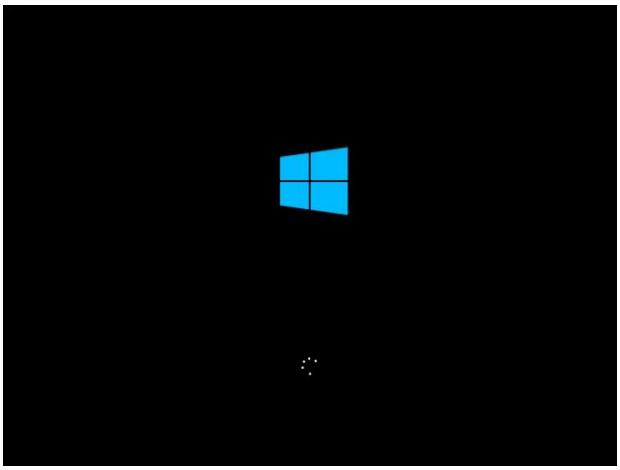


Figure 1

With Windows Embedded 8 Standard you have the ability to selectively remove all these UI elements during the Windows boot screen after the POST process. This functionality is configured using the Image Configuration Editor (ICE), which is one of the tools in the Windows Embedded 8 Standard Toolkit to create and edit configuration files for installing an image on a target device.

The Unbranded Boot module in ICE enables you to suppress traditional desktop Windows UI elements that are displayed when Windows starts or resumes. *Figure 2* below shows the Unbranded Boot module plus the settings that can be configured to suppress the operating system boot process.

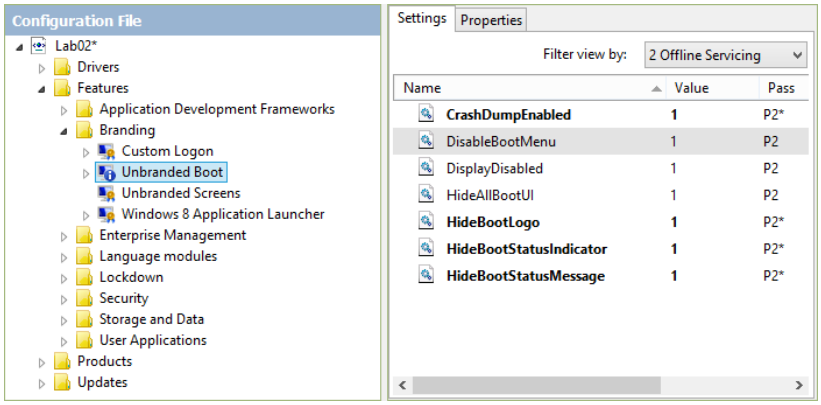


Figure 2

Additional functionality in this module gives you the ability to disable the F8 or F10 keys during boot and you can also suppress the crash screen if Windows encounters an error that it cannot recover from. Also, individual settings for each UI element let you pick which visual indicators you want to suppress. This can be useful if you want to remove the logo, but still want a visual indication that the operating system is loading (that is, you just want the spinning wheel status indicator, but not the Window logo). Note: the HideAllBootUI setting will override individual settings.

Enabling all of these settings as shown above will hide all boot screen elements and suppresses all Windows elements during operating system boot/startup. In this scenario, no messages or animations are shown and the screen remains black until the startup screens are displayed. If the device has a fast boot time this may not be such a big deal. However, if desired, it's now possible in Windows Embedded 8 Standard to replace the black screen with a custom OEM image.

Custom OEM Image

The boot logo itself can only be turned off in the Unbranded Boot module, but not be replaced with a company logo, a feature often desired by customers. This is because of the new boot loader architecture, which targets seamless boot sequences, coming up with the OEM's logo during post and showing this logo until Windows has completely booted. To achieve this seamless experience, the boot logo needs to be part of the UEFI firmware. The Boot Graphics Resource Table (BGRT) is an optional table supported on Unified Extensible Firmware Interface (UEFI) systems that provide a mechanism to indicate whether an image was drawn on the screen during boot, and some information about the image. Boot Graphics Resource Table (BGRT) functionality:

- Enables the seamless boot experience.
- Allows firmware to share the boot logo and its position for use past firmware POST.
- Allows the image to be redrawn if the boot path is interrupted.

Reference the Advanced Configuration and Power Interface (ACPI) specification revision 5 for more information (http://acpi.info/). More background information on the boot process can be found in this blog post of the Windows 8 team.

Unbranded Screens

To take the custom branding efforts a level higher, you are able to include the Unbranded Screens module, which replaces bitmaps and strings that display Windows Embedded 8 branding on the startup screen, the shutdown screen, and the system properties page with empty bitmaps and strings.

Logon Experience

The next area in creating a custom branded device is the logon experience. Again, the Windows default background is a straight giveaway that the underlying operating system is Windows based. Another branding module called Custom Logon in Windows Embedded 8 Standard provides the functionality to control the user logon experience. With this module you can still take advantage of the benefits of Windows logon like security and multiple-users but suppress the logon UI elements to provide a custom look and feel to your embedded device, in combination with the other embedded enabling features. The functionality in *Figure 3* below shows the main capabilities of the Custom Logon module.

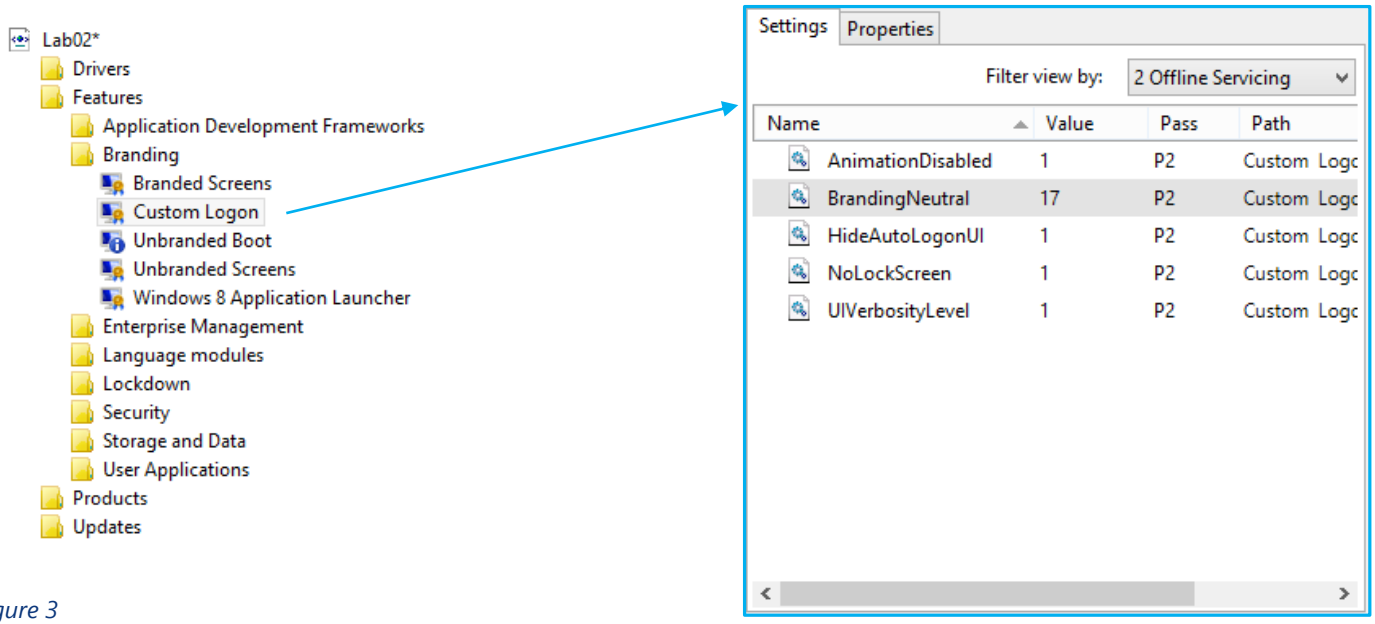
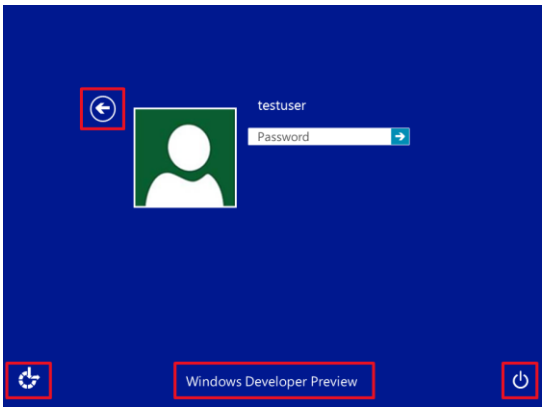


Figure 3

- AnimationDisabled. Hides the animation during user logon.
- BrandingNeutral. Provides granular control to remove any or all of the UI elements highlighted in red. This parameter is useful if you still require a traditional logon using the security capabilities provided Microsoft, plus the additional security afforded by removing the UI elements.



- Disable the Shutdown button.
- Disable the Language button.
- Disable the Ease of access button.
- Disable the Switch user button.
- Disable the Blocked Shutdown Resolve (BSDR) screen.

Figure 4

The last parameter in BrandingNeutral is for when there are applications that are blocking the system from shutting down, restarting, or signing off, which would normally display the Blocked Shutdown Resolver (BSDR) screen that asks for the user's attention. When this setting is enabled, restarting or shutting down the system causes the operating system to immediately force close any open applications that are blocking system shutdown. No UI is displayed, and users are not given a chance to cancel the shutdown process. Note: This can result in lost data if any open applications have unsaved data.

HideAutoLogonUI. Suppresses the logon screen during auto-logon. If you want to launch directly into the shell without displaying the logon UI, enable this setting and automatic logon. Automatic logon configured under Products, Embedded Core, AutoLogon.

- NoLockScreen. Disables the lock screen functionality

Many embedded devices auto-boot into their application for use by multiple users. An example would be an airline check-in kiosk. The last thing the airline wants is a lock screen to appear when the device has not been used for a period of time. Disabling the lock screen like the one shown in figure 5 helps the airline's application provide a consistent user experience for their customers.

Figure 5

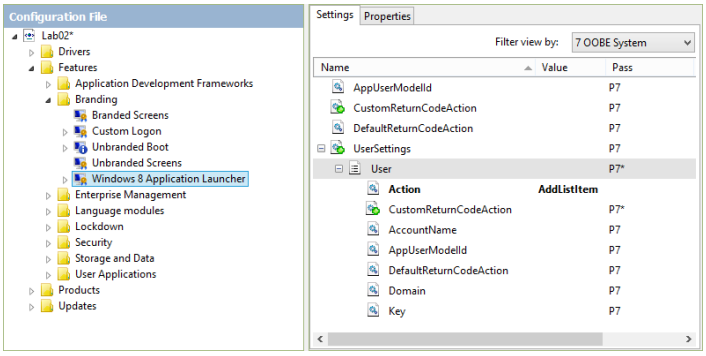
- UIVerbosityLevel. Enables or Disables configuration of Windows status message output during the logon or

shutdown process.

Automatically Launch Target Application

We've discussed how to disable and suppress the boot and logon UI elements, so now we can boot directly into our application to really create a seamless user experience when the embedded device is powered on. There are two main methods to automatically launch an application on Windows Embedded 8 Standard.

The first one is the Windows 8 Application Launcher, which is a module under the Branding feature within ICE. See *figure 6* below.

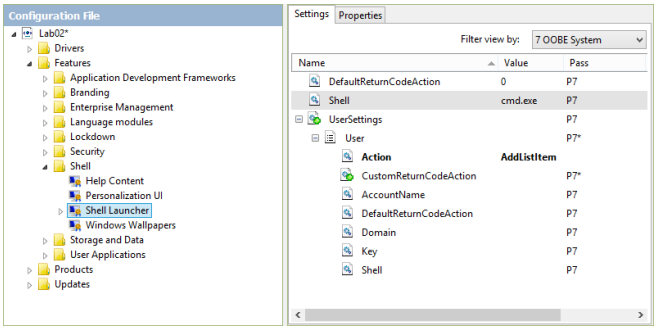


This module enables your device to automatically launch a specified Windows 8 app, and to take a specified action when the app exits. For example, you can relaunch the app, restart the operating system, shut down the operating system, or do nothing. This requires Application User Model Identifier (AUMID) of the Windows 8 app to be added to Windows 8 App Launcher module. See ICE help for more details.

Figure 6

For more information about how to automatically launch a new Windows 8 App, reference the lab on how to create a kiosk-style device. Note: With Windows 8 Application Launcher, the new Start screen with be displayed briefly before the app starts. This can be minimized by unpinning all the tiles on the Start screen and changing the Start screen background color to black or a matching color of the app background.

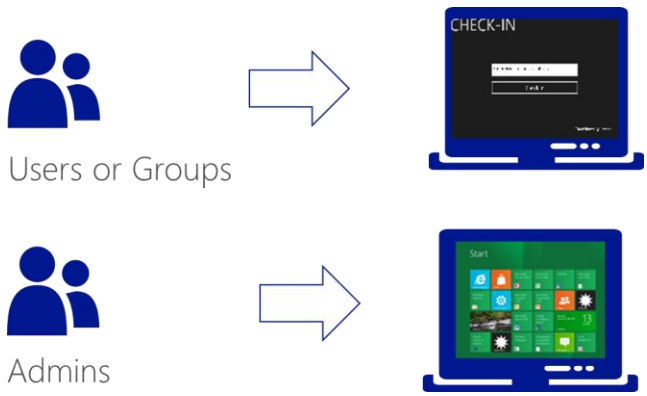
The second method is to use the Shell Launcher module to launch an application as a custom shell. See *figure 7* below.



The Shell Launcher module is for Win32.NET applications (.exe), or traditional desktop applications as they are sometimes known. The Shell Launcher has the ability trigger a custom action like restarting the shell if something goes wrong.

Figure 7

Both the Shell Launcher and Windows 8 Application Launcher can be configured to support multiple users so an administrator gains access to the device with the default explorer.exe (New Windows 8 Start screen) and users or user groups get a custom experience.



Hide Windows Status Messages

In many embedded usage scenarios, the user experience must not be interrupted by a dialog pop-up originating from the operating system or other applications. When a dialog pop-up happens on an embedded devices like the one in figure 6 below, not only does this create a poor user experience, it gives away the underlining operating system.

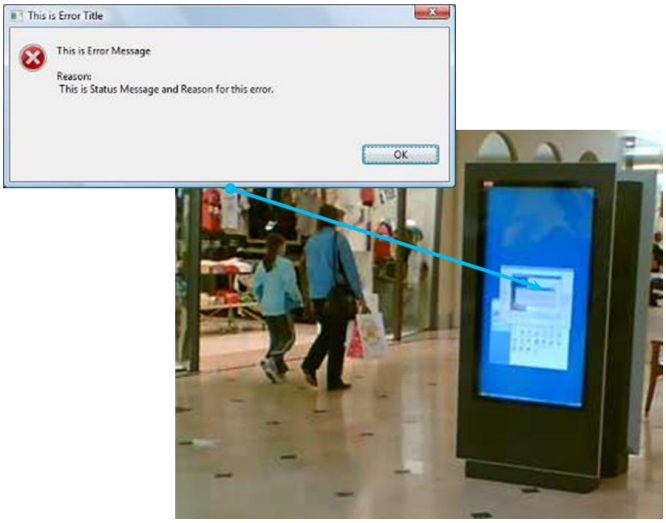


Figure 6

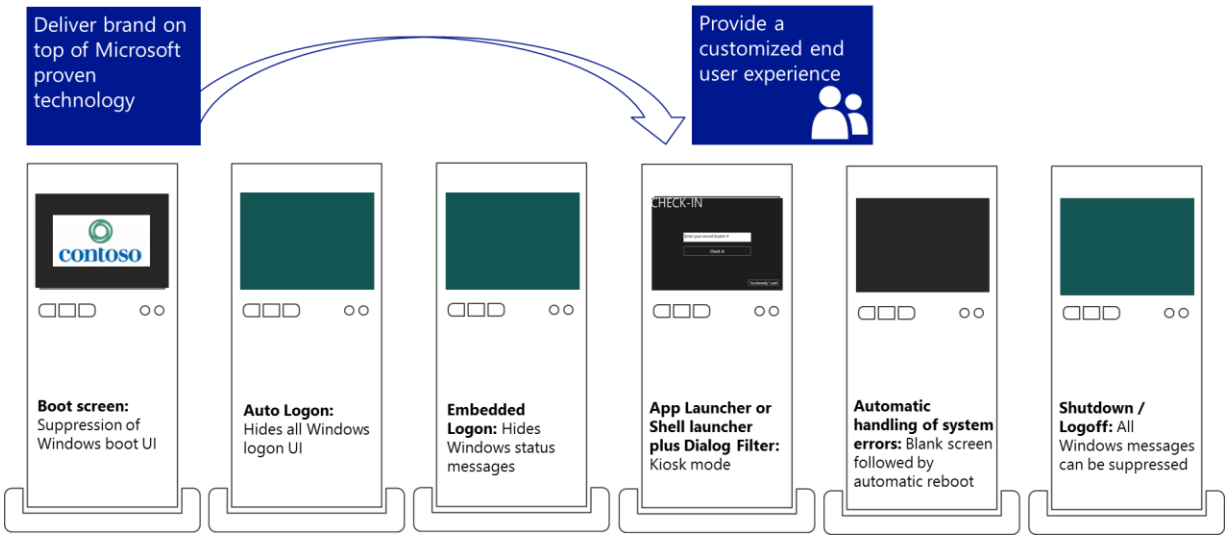
The Dialog Filter module enables you to control which pop-up windows are displayed on the screen, and to automatically handle pop-up windows by taking a default action, such as "close" or "show." Also, in Windows Embedded 8 Standard, you can configure the Dialog Box Filter to always show pop-up windows from specific processes, regardless of the specified default action. Each time a process creates or attempts to show a dialog box, Dialog Filter examines the properties of the new dialog box to determine if any actions need to be taken on the dialog box.

LAB—Remove Windows Embedded Brand Elements from Your Operating System

When you create a specialized Windows Embedded 8 Standard operating system, you can choose to remove nearly all brand elements that indicate that the operating system is based on Windows to create an embedded device with a great look and feel. Windows Embedded 8 Standard provides several modules that can help to remove the operating system branding. This lab demonstrates how to create a Windows Embedded 8 Standard operating system with no visible brand elements. You will use the Unbranded Boot, Custom Logon, Shell Launcher, and Unbranded Screens modules to help create an unbranded device. Visit MSDN for a related lab.

Summary

Windows Embedded 8 Standard has the sophisticated branding infrastructure to deliver a brand on top of Microsoft proven technologies, providing a customized user experience. These granular, configurable features allow removing pretty much everything that would point to Windows as the underlying operating system during boot and logon. When combined with other embedded enabling features (EFFs) like the Dialog Filter, Keyboard Filter, and Write Filters, Windows Embedded 8 Standard can provide a consistent and predictable user experience with increased robustness and security. To find out more, go to [www.microsoft.com/windowsembedded/.](www.microsoft.com/windowsembedded/)



Resources

Below are additional lab exercises posted on MSDN that walk you through some of the additional features of Windows Embedded 8 Standard. Each lab lists any dependencies on other lab exercises in the prerequisites section.

- Create a Kiosk Sample Windows 8 App. Create a basic sample JavaScript Windows 8 app that demonstrates a kiosk-style experience.
- Create a Kiosk-Style Device. Create a kiosk-style device in a complete end-to-end lab, from creating an image in Image Configuration Editor (ICE) to capturing and deploying the image to another device.
- Use Keyboard Filters to Block Keys in Your Operating System Image. Use the Keyboard Filter feature to block specific key presses or key combinations in the Windows Embedded 8 Standard operating system image on your devices.

Microsoft