
SERVICING & CUSTOMIZING WINDOWS XP EMBEDDED

GWYNETH SALDANHA

APRIL 2007

Department of Mathematics and Computer Science
Algoma University College, Sault Ste. Marie, Ontario

© Gwyneth Saldanha

ABSTRACT

This thesis will attempt to examine the issues involved in customizing or modifying the hardware or the underlying operating system for thin client devices based on Microsoft Windows XP Embedded and offer a solution for working through the required process.

TABLE OF CONTENTS

CHAPTER 1 : INTRODUCTION.....	8
1.1 EMBEDDED OPERATING SYSTEMS	8
1.2 OBJECTIVE OF THE THESIS	9
CHAPTER 2 : XP EMBEDDED: AN OVERVIEW.....	10
2.1 THE MICROSOFT EMBEDDED PLATFORM	10
2.2 WINDOWS XP EMBEDDED	11
2.3 DIFFERENCE BETWEEN WINDOWS XP AND WINDOWS XP EMBEDDED	13
2.4 SYSTEM REQUIREMENTS AND COSTS.....	14
2.4.1 <i>Development System Requirements</i>	15
2.4.2 <i>Target Device Requirements</i>	15
2.4.3 <i>Licensing Costs</i>	16
2.5 SUMMARY.....	17
CHAPTER 3 : THE EMBEDDED STUDIO TOOLS	18
3.1 TARGET ANALYZER.....	18
3.2 COMPONENT DESIGNER	20
3.3 COMPONENT MANAGEMENT INTERFACE	22
3.4 COMPONENT DATABASE MANAGER	23
3.5 TARGET DESIGNER	24
3.6 EMBEDDED ENABLING FEATURES.....	26
3.7 THE COMMAND LINE TOOL	29
3.8 SUMMARY	30
CHAPTER 4 : XP EMBEDDED IMAGE DEVELOPMENT CYCLE	31
4.1 CREATING AN XP EMBEDDED IMAGE	31
4.1.1 <i>Analyzing the Target Computer</i>	31
4.1.2 <i>Create a Component for the Target Device</i>	32
4.1.3 <i>Creating a New Configuration in Target Designer</i>	34
4.1.4 <i>Update Configuration Settings</i>	35
4.1.5 <i>Check Dependencies</i>	35
4.1.6 <i>Build the Run-Time Image</i>	36
4.1.7 <i>Deploy the Run-Time Image</i>	37

4.2 BOOTING FROM USB FLASH	38
4.2.1 Overview	38
4.2.2 Format USB Media	39
4.2.3 Add USB Boot 2.0 Component	40
4.2.4 Build Target Image and Transfer to USB Media	41
4.2.5 Boot with the USB Media	41
4.3 DEMONSTRATING XP EMBEDDED WITH VIRTUAL PC	42
4.3.1 Setting up the Virtual Machine	42
4.3.2 Capturing the Hardware Information	43
4.3.3 Building and Deploying the Image	43
4.3.4 Running XP Embedded	43
4.4 SUMMARY	44
CHAPTER 5 : COMPONENTS AND COMPONENTIZATION	45
5.1 COMPONENT DEVELOPMENT	46
5.2 ADVANCED COMPONENT DEVELOPMENT	47
5.2.1 Tools Used on the Development PC	48
5.2.2 Tools Used on the Target Device	49
5.3 COMPONENT SUPPORT FROM THIRD-PARTY VENDORS	50
5.4 SUMMARY	51
CHAPTER 6 : MANAGING AND SERVICING RUNTIME IMAGES	53
6.1 UPDATES OVERVIEW	53
6.2 DATABASE UPDATES	54
6.2.1 Creating a Database Update	55
6.2.2 Advantages of a Database Update	56
6.2.3 Issues with Database Updates	56
6.3 DESKTOP UPDATES	56
6.4 SERVICING EMBEDDED DEVICES	57
6.4.1 Recovery CD	58
6.4.2 Run-Time Image Replacement	59
6.4.3 Remote Management	60
6.4.4 Device Update Agent	60
6.4.5 Software Update Services (SUS)	60
6.4.6 Systems Management Server (SMS)	63
6.5 DEVICE UPDATE AGENT	64
6.5.1 Using Device Update Agent	65
6.5.2 Web Server Configuration	66

6.5.3 Security Issues.....	67
6.6 SUMMARY.....	68
CHAPTER 7 : SECURITY CONSIDERATIONS.....	69
7.1 ASSESSING YOUR SECURITY RISK.....	70
7.2 REDUCE YOUR SECURITY RISKS	72
7.3 BUILDING IN WINDOWS SECURITY	72
7.4 SECURING PHYSICAL MEDIA BY USING ENHANCED WRITE FILTER.....	74
7.5 PROTECTION FROM COMPUTER VIRUSES	75
7.5.1 EWF is NOT an Anti-Virus Solution	75
7.6 SUMMARY.....	76
CHAPTER 8 : COMPARISON WITH OTHER EMBEDDED OPERATING SYSTEMS.....	78
8.1 WINDOWS CE	78
8.2 WINDOWS EMBEDDED NT	80
8.3 EMBEDDED LINUX	81
8.4 SUMMARY.....	84
CHAPTER 9 : CONCLUSION	85
9.1 WINDOWS VISTA FOR EMBEDDED SYSTEMS	85
9.2 SUMMARY.....	86

LIST OF FIGURES

FIGURE 1: WINDOWS XP EMBEDDED IS A COMPONENTIZED VERSION OF WINDOWS XP	13
FIGURE 2: LICENSING COSTS	16
FIGURE 3: EMBEDDED STUDIO TOOLS	18
FIGURE 4: THE COMPONENT DESIGNER.....	21
FIGURE 5: THE COMPONENT DATABASE MANAGER.....	24
FIGURE 6: THE TARGET DESIGNER.....	25
FIGURE 7: SET THE COMPONENT VISIBILITY LEVEL IN TARGET DESIGNER'S 'TOOLS > OPTIONS'	26
FIGURE 8: EMBEDDED ENABLING FEATURES	28
FIGURE 9: PARTIAL OUTPUT WHEN RUNNING TAP.EXE ON TARGET DEVICE.....	32
FIGURE 10: IMPORTING THE TARGET PMQ FILE AS A COMPONENT.....	33
FIGURE 11: ADDING THE SELECTOR PROTOTYPE COMPONENT AS THE PROTOTYPE	34
FIGURE 12: RESOLVE DEPENDENCIES BY CLICKING ON THE TASK NAMES.....	35
FIGURE 13: BUILDING AN IMAGE.....	37
FIGURE 14: FORMAT THE USB MEDIA WITH UFDPREP.EXE	40
FIGURE 15: ADD THE USB BOOT 2.0 COMPONENT.....	40
FIGURE 16: BUILD THE USB BOOTABLE IMAGE.....	41
FIGURE 17: FILE RESOURCE DETAILS.....	47
FIGURE 18: SUS OVERALL PROCESS.....	62

LIST OF TABLES

TABLE 1: UPDATE TERMINOLOGY	53
TABLE 2: UPDATE TYPE BASED ON DEPLOYMENT METHOD.....	54
TABLE 3: SERVICING OPTIONS	58
TABLE 4: AUTOLOGON LEVELS	68
TABLE 5: RECOMMENDED WINDOWS EMBEDDED OPERATING SYSTEM BY DEVICE CATEGORY	79

CHAPTER 1 :

INTRODUCTION

1.1 Embedded Operating Systems

Imagine being able to rebuild Windows exactly how you want, with nothing more or less than what you need to do what you want. Imagine having a Windows so customizable that you can simply drag and drop the components you want--components that you may have tweaked or built from scratch – into a complete OS of your making. Sounds like a dream operating system, doesn't it? Well, that's the basic premise of Microsoft's Windows XP Embedded – the ability to custom make Windows in your image. It has everything XP Professional has: all the compatibility and, unless you specifically put it in, none of the bloat.

Traditionally, XP Embedded has only been used in things like dumb terminals that don't need the full XP experience, but it works just as well (if not better) on the desktop than traditional XP Professional since it's custom built and has no bloat.

This paper attempts to analyze issues with servicing and customizing Microsoft's embedded operating system – Windows XP Embedded.

1.2 Objective of the Thesis

An embedded operating system is quite different from the traditional operating system. Many are skeptical about taking on the daunting task of using Microsoft's Windows XP Embedded (often referred to as XPe). Some of the common perceptions that consumers seem to have about Windows Embedded operating system range from; 'The operating systems are too big', 'The operating systems are not secure', 'It's too hard to configure an operating system for my needs', 'There's no driver support for my hardware.'

This thesis aims to educate the reader about working with XP Embedded and explore some of the main issues that consumers face. For this thesis Windows XP Embedded Service Pack 2 Feature Pack 2007 has been used. The thesis is divided into 3 sections. First an introduction to Windows XP Embedded is presented to familiarize the reader with the process of creating and deploying XP Embedded images. Next the issues of customizing, updating and servicing the deployed images are addressed. Lastly a comparison between competing embedded operating systems is explored.

Although XP Embedded was released in 2001 there are a lot of consumers with confused faces. It's been an elating experience clearing the confusion off of my face.

CHAPTER 2 :

XP EMBEDDED: AN OVERVIEW

2.1 The Microsoft Embedded Platform

Microsoft's mission in this thriving embedded market is to deliver adaptable and scalable platforms for 32-bit, connected devices that enable rich applications and services.ⁱ

Microsoft officially entered the embedded marketplace in November of 1996 with the release of Windows CE 1.0. Windows CE was designed from the ground up to provide embedded developers with the ability to extend the “sophisticated software environment of today's personal computer into the embedded world,” according to Craig Mundie, then Senior VP of the Consumer Platforms Division at Microsoft.

After the release of Windows CE, Microsoft quickly discovered that many embedded developers were building a wide range of non-PC devices that were neither small nor resource constrained and could benefit from a PC-based architecture, an enhanced set of features, richer functionality and greater scalability than what Windows CE could provide at the time. In 1999, to compliment its embedded offerings, Microsoft delivered Windows NT Embedded to the market, thereby providing

embedded developers with greater choice and flexibility as well as access to the rich Windows feature set.

In 2001, Microsoft released Windows XP Embedded, the successor to Windows NT Embedded, which provides a wealth of new features created for the mainstream operating system, and is available for the embedded marketplace in componentized form.

The Windows Embedded family of operating systems provides developers with the building blocks to create a wide variety of embedded devices, for example: industrial control systems, mobile and handheld devices, set-top boxes, retail point of sale devices, and thin clients. Developers can choose only those components of the operating system needed to satisfy device design requirements.

Windows Embedded customers can be categorized within two groups: original equipment manufacturers (OEMs) and their partners who customize an operating system to meet specific design needs, and groups within Microsoft that are building specific device platforms, such as Pocket PC.

2.2 Windows XP Embedded

The embedded version of Windows XP is a componentized version of the well-known Windows XP Professional operating system. Instead of everything being wrapped tightly into a single package, XP Embedded breaks the OS down into more than 10,000 individual components,

allowing developers to create systems that have the functionality and familiar features of XP. One of the most attractive features of XP Embedded is that it is much smaller than XP for desktop systems — so small, in fact, that it can fit on a 512 MB flash card, and still leave room for system applications and data backupⁱⁱ.

There are 2 service pack releases for XP Embedded, namely SP1 and SP2. Microsoft launched Windows XP Embedded SP2 Feature Pack 2007 in November 2007. This feature pack extends Windows XP Embedded Service Pack 2 providing developers with new embedded enabling features, re-optimized OS components, enhanced servicing options and powerful embedded-specific development toolsⁱⁱⁱ.

Also of great interest for the embedded system developer are XP Embedded's choices of boot methods, its wide range of communications options, and the array of tools available for system configuration. XP Embedded uses the same application programming environment as XP, which makes application development quick and easy, and allows the same application to run on desktop and embedded machines. And it has improved code protection for critical kernel structures, file protection and more.

A few of the many possible consumer and commercial applications of XP Embedded would include set top boxes and terminals such as kiosks and ATMs.

2.3 Difference between Windows XP and Windows XP embedded

Both Windows XP Embedded and Windows XP are built on the same stable code base of Microsoft Windows NT® and Windows 2000. This code base offers a protected memory model and pre-emptive multitasking, both of which contribute to system stability. Starting from this proven code base, Windows XP Embedded is fundamentally reliable starting at the kernel level.

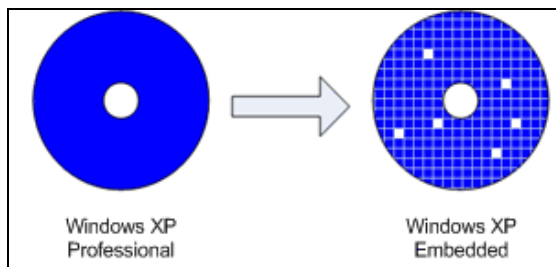


Figure 1: Windows XP Embedded is a componentized version of Windows XP

The difference between the Windows XP desktop operating system and the Windows XP Embedded operating system is that Windows XP Embedded is modularized into components. Componentization makes it easy to build fixed-function devices, and to reduce the footprint of a target run-time image. You can use Windows XP Embedded to build a run-time image that includes only the Windows components that your device requires. As long as you include the required dependencies in your run-time image, you can maintain the application compatibility that you need to run your applications. Componentization also makes it easier to reduce security risks by designing run-time operating systems with limited surface area. The smaller the footprint of your run-time image, the smaller the surface area of the operating system. Less surface area means less risk of intrusion. For example, if you do not require networking capabilities, you can exclude networking components from your run-time image.

Even though Windows XP Embedded is built from the same binary files that Windows XP Professional uses, Windows XP Embedded does not share all the features of Windows XP Professional. Some features in Windows XP Professional are not appropriate for embedded device scenarios. For example some Windows XP Professional features that are not included in Windows XP Embedded are: the Windows File Protection (WFP), Windows XP Tour, Windows Setup, Online product activation, Out-Of-Box Experience (OOBE), Windows Update and MSN® Explorer^{iv}. These are not included in Windows XP Embedded because of the highly customized nature of Windows XP Embedded-based operating systems.

2.4 System Requirements and Costs

In addition to over 12,000 feature components, the Windows® XP Embedded toolkit includes the following development software:

- Microsoft Visual Studio
- Microsoft .NET Framework
- Microsoft Windows Embedded Studio, a full suite of development tools including: Target Analyzer, Target Designer, Component Designer, Component Database, Component Database Manager. These are described in detail in the next chapter.

2.4.1 Development System Requirements

A development system is used to assemble and build the run-time image for the target device. The development system is either a true client or a client and server. The Development System requirements are as follows.

Client requirements:

- Pentium III, or equivalent, 256 megahertz (MHz) processor
- 256 megabytes (MB) of memory
- Approximately 2 gigabytes (GB) disk that will contain tools and images

Server requirements:

- Pentium III, or equivalent, 256 MHz processor
- 256 MB of memory
- Approximately 10 GB disk that will contain the component database and repositories

2.4.2 Target Device Requirements

The main requirement for the target device is that it must have an Intel x86 or fully compatible processor. In order to use the Target Analyzer to determine the basic hardware configuration, the target device must be able to boot MS-DOS from a disk or be running Windows 2000/XP^v.

2.4.3 Licensing Costs

To ship a device and bring it to market, runtime licenses and certificates of authenticity are required for each unit shipped. The cost for each runtime license may vary, based on the volume of licenses that are purchased.

Windows XP Embedded	ERP	What is included?
Evaluation Kits (non-commercial use)	No charge	The runtimes and operating systems you build. Usage expires after 120 days.
Development tools	\$995 US	The tools that enable you use to build runtimes and operating systems. Tools do not expire.
Runtime license cost per unit	\$90 US	Runtimes licenses for each operating system that is shipped. Runtimes never expire.

Figure 2: Licensing Costs^{vi}

The Microsoft Support Lifecycle Policy guarantees that Microsoft provides five years of mainstream support and five years of extended support for Windows Embedded products after release, for a total of 10 years of support for the embedded device^{vii}.

2.5 Summary

Now that we know what are the technical requirements of the development and deployment environment we can familiarize ourselves with the Embedded Studio Tools and how they are used to develop a XP Embedded runtime image.

CHAPTER 3 :

THE EMBEDDED STUDIO TOOLS

The architecture of Windows XP Embedded is a complicated beast, too long for this thesis. Instead, this section focuses on the architecture behind the individual tools in a structured manner.

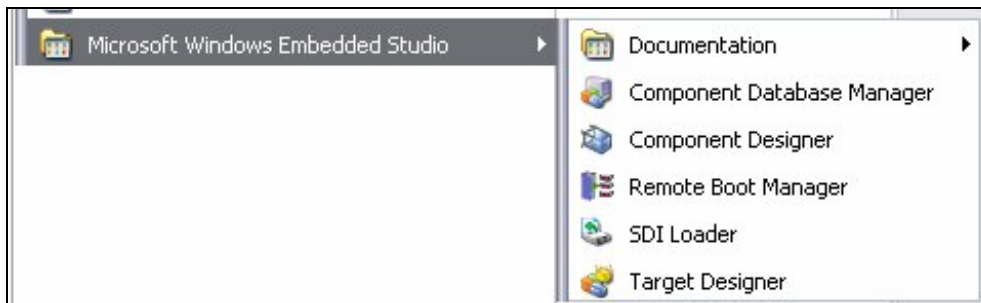


Figure 3: Embedded Studio Tools

3.1 Target Analyzer

The Target Analyzer consists of the Target Analyzer probe utility and the Target Analyzer importer.

There are two flavors of the Target Analyzer probe utility, one for a full or limited Win32 system (called TAP.EXE), and one for DOS systems (called TA.EXE). You can run TAP.EXE on Windows XP, Windows 2000, or a WinPE (Preinstall Environment) CD system. TA.EXE runs best

under DOS, or a Win9X boot disk command prompt. If you can boot your device to Windows XP or Windows 2000, then TAP.EXE is your best bet.

TA.EXE should only be used if you absolutely cannot install or boot a full Win32 environment on your device. The reason being that TAP.EXE gets its hardware information directly from the Win32 registry whereas TA.EXE has to query the hardware directly, and in 16-bit mode as well, so it's limited to asking the BIOS for the hardware list. This limits us to devices that the BIOS knows about, which means we can't enumerate IDE, SCSI, PCI, or other device busses^{viii}.

The Target Analyzer Probe (tap.exe or ta.exe) when executed on the target machine produces a listing of all hardware devices on the target in the form of an XML based .pmq file.

The Target Analyzer Importer is a module of the Component Designer and Target Designer. The .pmq can be imported into either. If the .pmq is imported into the Component Designer a component with dependencies on the identified hardware drivers is created. If the .pmq is imported into Target Designer then appropriate device drivers are added to the configuration. Importing the .pmq ensures that the component or configuration includes all of the resources necessary to support the devices of the target.

Some caveats bear mentioning. TAP.EXE is very good at what it does; sometimes it's too good. TAP.EXE can pick-up ghost devices—devices that have been previously, but are not currently, installed on the system. Usually these are USB devices. So it's important to make sure all the hardware in the component is current and valid. An alternative would be to run TAP.EXE on a clean install of Windows XP or Windows 2000. Lastly, devices gathered from a Windows 2000 system may have different names in Windows XP Embedded. You'll have to use your own intuition to find out what the new devices are if you want to disable them from the Device Manager list.

3.2 Component Designer

The real work (and benefit) to XP Embedded is in creating components that are not part of the database. A Component is a combination of properties, files, registry data, custom resources, dependencies, group memberships, DHTML and build-scripts. The Component Designer is used to create custom components, such as a component based on the .pmq file obtained by running the Target Analyzer on a particular target. These components are stored in Source Level Definition files or SLDs (a.k.a slides) which are XML files that specify the resource files and properties of the component. After you add a custom component to a .sld file, you can modify, test, release, and update it.

In general there are four types of components that are created:

Device Driver Component – Most hardware devices come with Information (.inf) files that instruct the Windows Installer how and where to install the device driver. Component Designer can import most INF data to create a component.

Macro/Platform Component – Macro/Platform components consist of multiple components and other macro components. They save time in creating a configuration and importing the data into Target Designer. Target Analyzer Data (PMQ) files can also be imported to create a platform component.

Application Component – Application components are the best way to get an application installed directly into the image. Application can be third party or your own custom application.

Third party applications usually require some extra work in finding the support resources needed to run. This is covered in the Chapter Four

Primitive Component – A Component that has a single binary file. They are the smallest component elements and usually have a low visibility setting. A DLL or library that is used by multiple applications should be created as a separate primitive component. For e.g. OLE32 and MSVCRT40 are primitives.

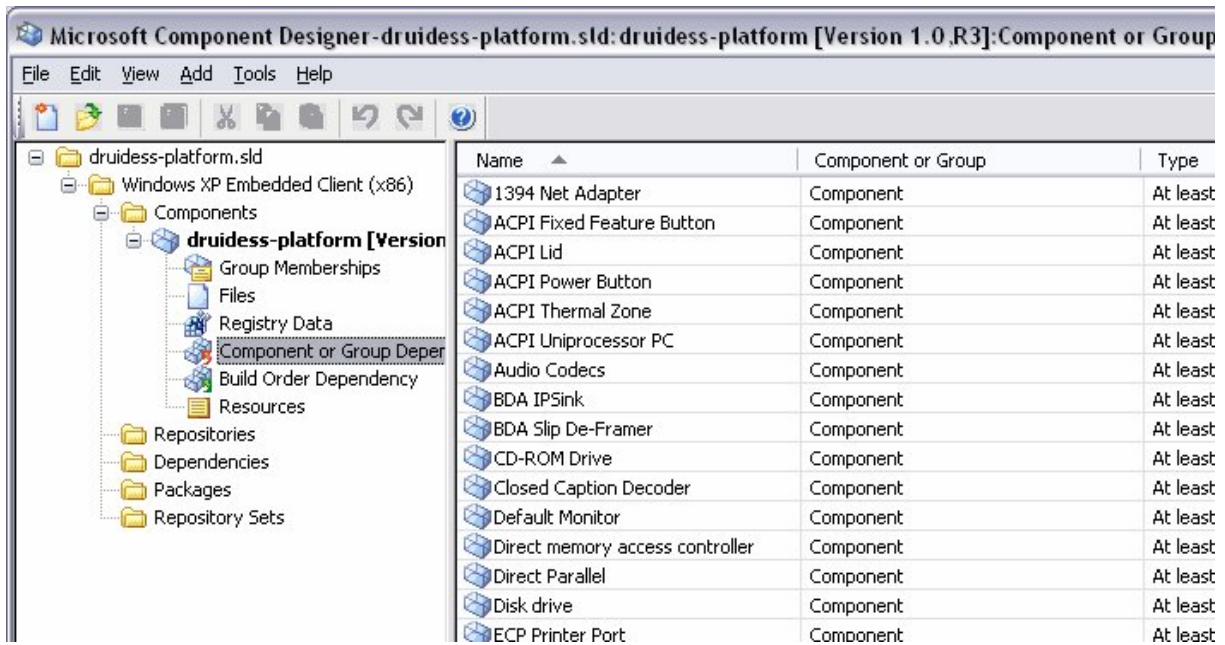


Figure 4: The Component Designer

3.3 Component Management Interface

The CMI, or Component Management Interface, is a part of the tools that is usually never seen or heard from. Windows XP Embedded uses a SQL-based database to hold all components. This database can be a local or remote Microsoft SQL Server installation, or a local Microsoft Data Engine (MSDE) installation –which is provided on the Windows XP Embedded CD.

In order to seamlessly provide access to both local and remote databases from one set of tools, as well as provide for switching databases on the fly, a database communications layer is required in the overall architecture. This layer is the Component Management Interface. Its primary purpose is to provide a standard interface between the Windows XP Embedded tools (Target Designer, Component Designer, and Component Database Manager) and the component database, wherever it resides (local or remote, SQL Server or MSDE). If it's related to anything in the component database, the CMI will be involved.

Since all the tools rely on an active database connection to do any useful work, the first thing any of the tools do is to ask the CMI to provide an active database connection. If no database connection is possible, the CMI returns a failure and the tool reports an error. In short, there is no way to do any meaningful work in Windows XP Embedded without a database connection.

The CMI also provides for some level of asynchronous database access, which is highly likely with a remote SQL Server database and multiple clients. All operations involving database changes are fully transacted in SQL, providing rollback capabilities in the event of a failed operation. The CMI also distinguishes between Read-Only and Exclusive access modes. If any tool wants to delete information from the database (currently only Component Database Manager), it first needs Exclusive Access, and it can't get that if any tool has the database open. On the other hand, if Exclusive Access is granted, no other tool can gain access to the database until Exclusive Access is released.

3.4 Component Database Manager

Components on their own aren't worth very much. They're small XML files that contain a set of data describing a concrete unit of functionality. The Component Database Manager does a number of things for us, but its prime purpose is to take component definitions in the form of SLD files and put them in a component database that all the Windows XP Embedded tools can read.

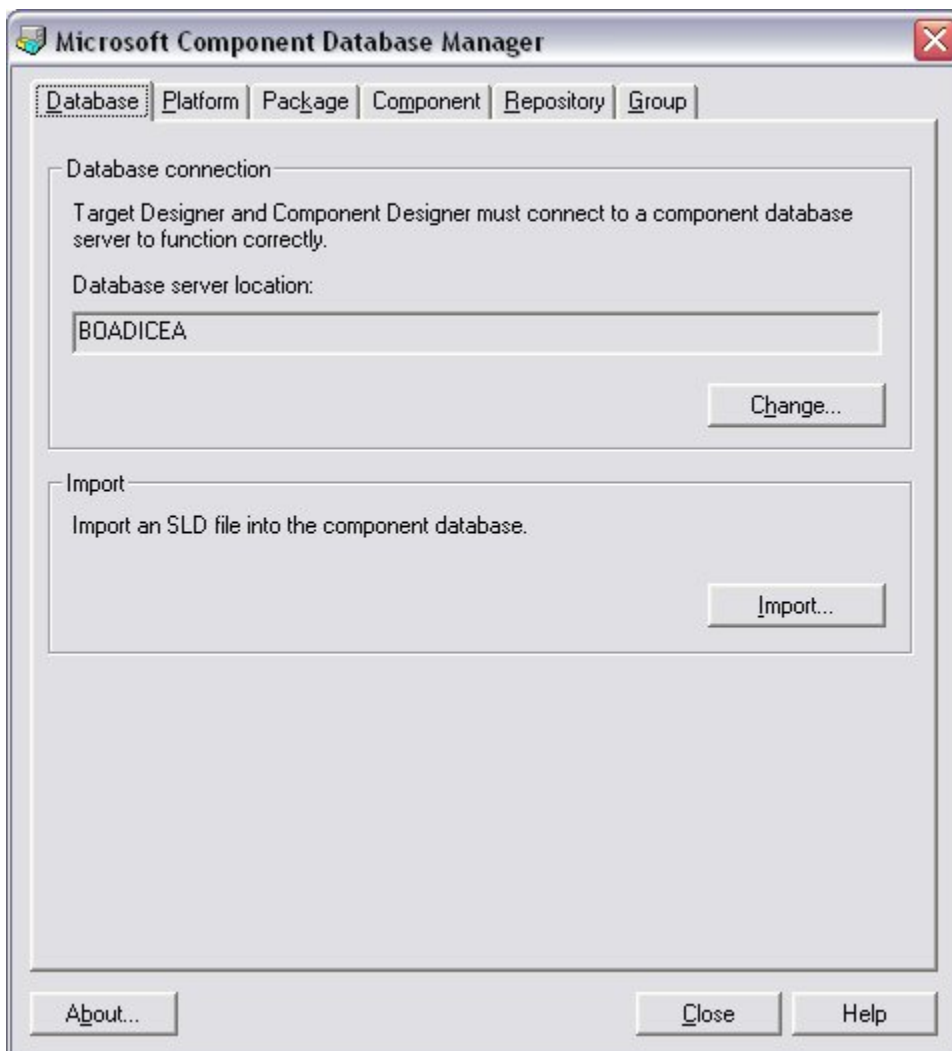


Figure 5: The Component Database Manager

The server listed in Component Database Manager is the server that all Windows XP Embedded tools will use. Its value is stored in the registry on the machine Component Database Manager is running on. You can have multiple databases across your network, and use Component Database Manager to switch between them^{ix}.

The Component Database Manager uses the Component Management Interface to add new components to the database and display the current components in the database. Once a component has been created, its .sld file must be imported to the component database using the Component Database Manager. Entities may be deleted and the resource file repositories managed from the Component Database Manager.

3.5 Target designer

The Target Designer is used to create configurations, which are stored in .slx files, and build XP Embedded images from those configurations.

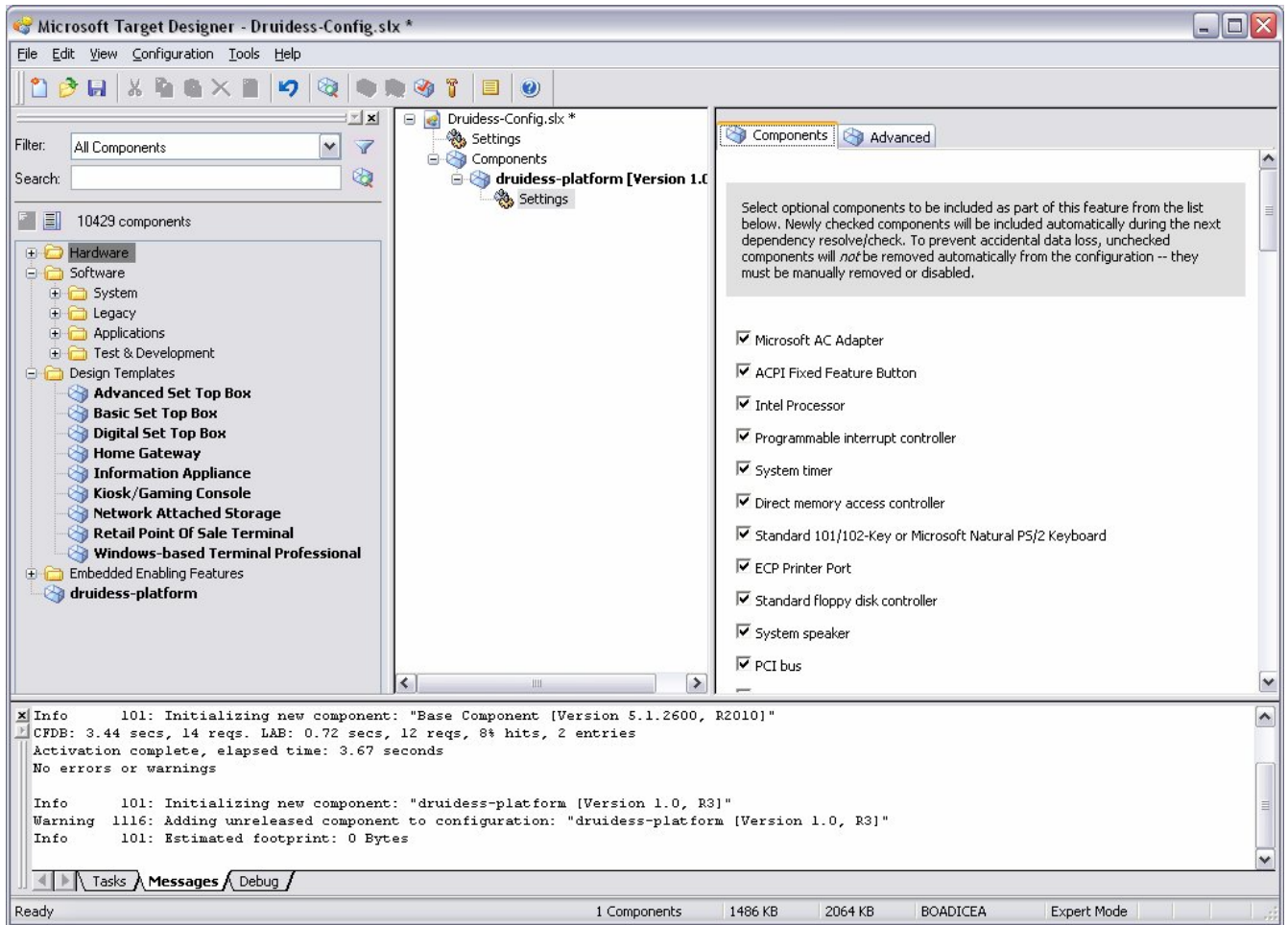


Figure 6: The Target Designer

There are three panes in the Target Designer; on the left is the Component Browser, in the middle is the Configuration Editor and to the right is the Details Pane^x. Components that are available in the Component Database are displayed in the Component Browser and are added to the configuration by dragging them into the Configuration Editor pane.

There are several default Design Templates available to form the base of XP Embedded configurations: Windows-based Terminal Professional, Information Appliance, Basic Set Top Box, Digital Set Top Box, Advanced Set Top Box, Kiosk/Gaming Console, Home Gateway, Retail Point

of Sale Terminal, and Network Attached Storage. Each component has a visibility level property, which is a value from 100 to 10,000. The Target Designer also has a visibility level and will only display a component in the browser if the component's visibility level is higher than the Target Designer's visibility level. The default visibility level for components is 1000, macro components are usually set to 2000, and "hidden" components usually have a visibility level of 500. To view all available components in the Target Designer, set its visibility level to 100 from Tools > Options.

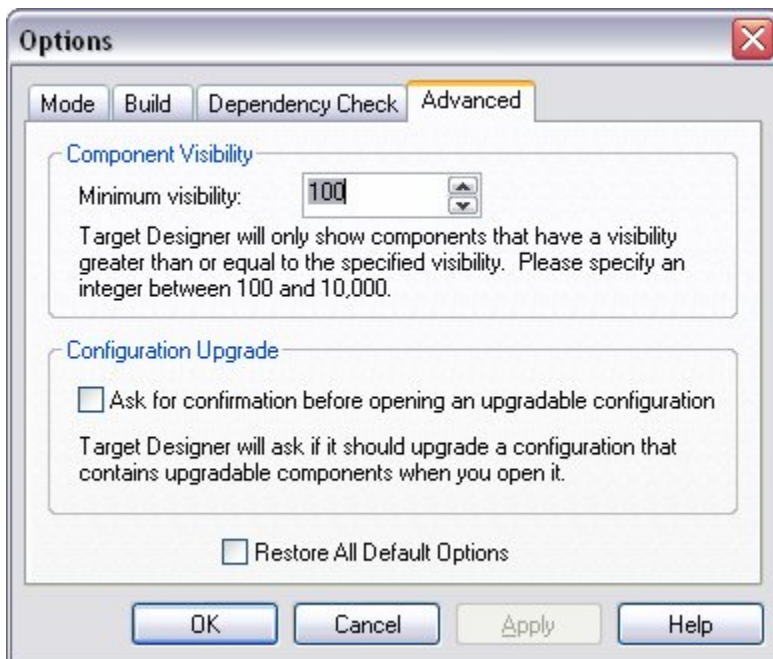


Figure 7: Set the component visibility level in Target Designer's 'Tools > Options'

3.6 Embedded Enabling Features

Embedded enabling features are included to help optimize the operating system for embedded devices. Some of the features are:

Enhanced Write Filter (EWF): This protects the contents of disk volumes by redirecting write operations to a different storage location called 'overlay' for e.g. RAM or a disk partition.

Hibernate Once Resume Many (HORM): Here the state of the machine is saved by copying all the contents of RAM to the drive and thus enables the device to achieve an instant on experience.

Remote Boot: This service uses the Pre-boot Execution Environment (PXE) protocol to boot a device over a network. The PXE communicates with the server and retrieves a boot image over a network.

Update Services: The System Management Server (SMS), Windows System Update Services (SUS), and Device Update Agent (DUA) all address servicing the system when deployed in the field. These three servicing solution address incremental updates and can be handled remotely without having a service technician attend each station.

System Deployment Image (SDI): Windows XP Embedded includes the SDI (System Deployment Image) feature, which enables you to manage your run-time images. SDI tools are used to facilitate the preparation and maintenance of run-time images.

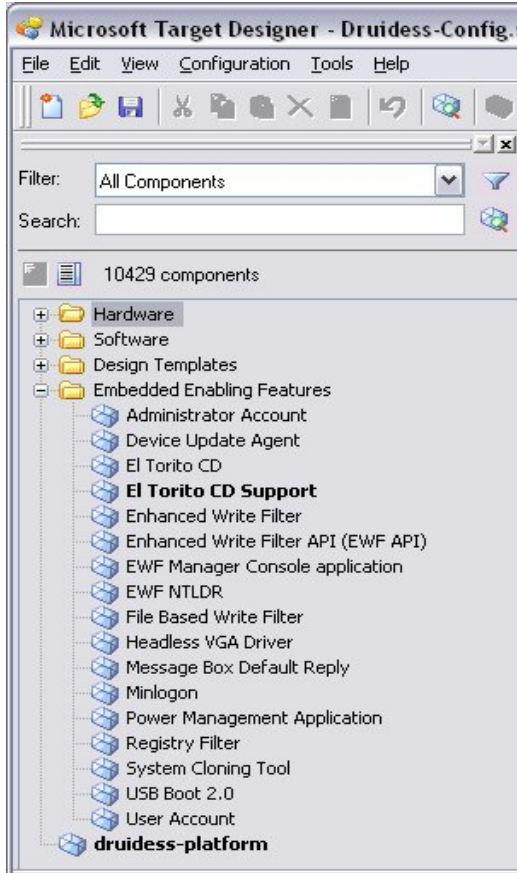


Figure 8: Embedded Enabling Features

Feature Pack 2007 extends Windows XP Embedded providing developers with new embedded enabling features, re-optimized OS components, enhanced servicing options and powerful embedded-specific development tools^{xi}. Additionally, the Feature Pack 2007 enables additional embedded-enabling features such as:

File Based Write Filter: Unlike EWF where the complete disk partition was protected the File Based Write Filter (FBWF) allows developers to selectively protect individual files and folder

USB Support: Native support in the operating system for booting USB mass storage devices (including USB Flash) on systems with USB 2.0 Boot capability. With USB Boot, a device maker can achieve improved diagnostic and servicing. By storing a diagnostics image on a USB Flash key, a field technician can boot a deployed device in the field for maintenance or troubleshooting.

3.7 The Command Line Tool

With the release of Feature Pack 2007, the Windows Embedded Studio tool suite now includes a command line script interface tool designed to automate the end-to-end build processes, increasing developer productivity.

The Windows XP Embedded Command-Line Tool (XPECMD) enables you to perform many of the same tasks as in the GUI tools, including creating and opening runtime configurations, adding, modifying and removing component instances, resolving dependencies, building runtimes, and importing components into the database.

For e.g.: Creating a new configuration can be done by using the `initcfg` command.

```
xpecmd> initcfg cfg
```

This command creates a new configuration object and automatically initializes and activates it.

You can use the new configuration once it has been activated.

You can also create a new runtime configuration using the new command.

```
xpecmd> new cfg Configuration
```

3.8 Summary

The XP Embedded development process is a combination of tools that help perform hardware analysis, the creation of custom components and actually building the XP Embedded images. In this chapter we covered the core tools and concepts used to build Windows XP Embedded images. In the next chapter we explain the actual soup to nuts process of building an XP image and deploying it.

CHAPTER 4 :

XP EMBEDDED IMAGE

DEVELOPMENT CYCLE

There are several deployment options available to developers who wish to create Windows XP Embedded images. This chapter describes how to build images with the development environment that was used to demonstrate this thesis.

4.1 Creating an XP Embedded Image

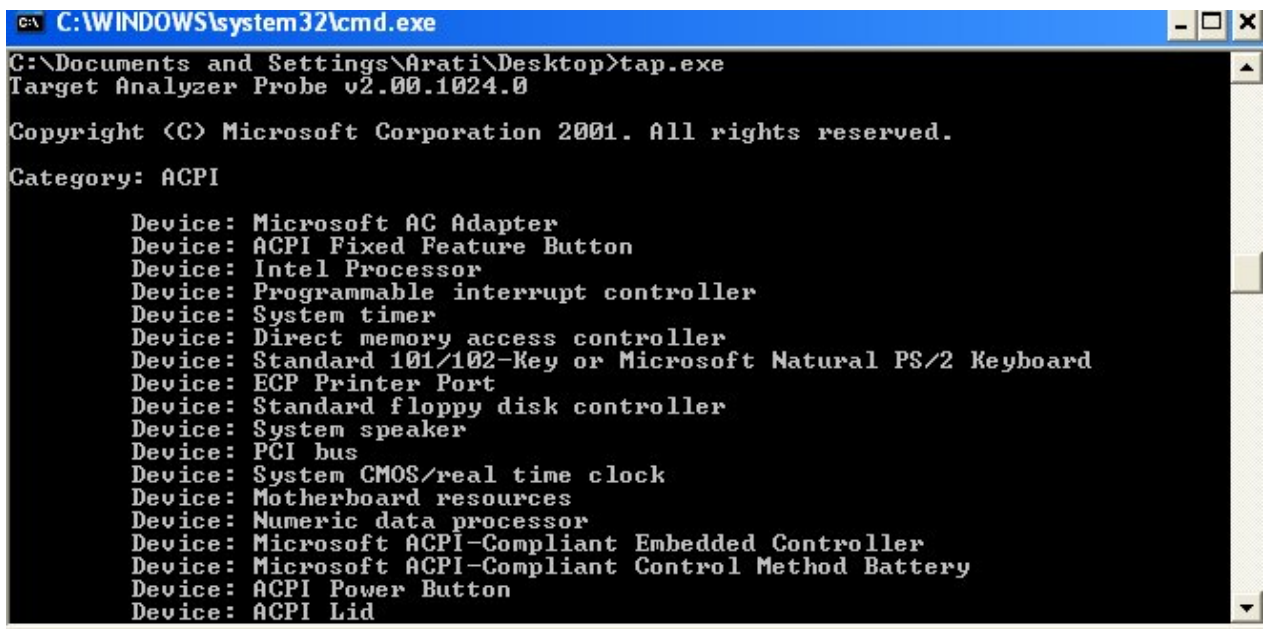
The development environment (Boadicea) was a desktop machine with Windows XP Professional and Windows Embedded Studio SP 2 Feature Pack 2007 installed. The target device (Druidess) was a HP Laptop with an existing Windows XP Professional installation.

4.1.1 Analyzing the Target Computer

Since the Target device already had Windows XP running the Win32 Target Analyzer Probe TAP.EXE was used to generate the .pmq file. The PMQ file contains information about the target's

hardware. Once the inventory list is captured the next step is either to create a component with the Component Designer or create a Configuration in Target Designer.

One of the main reasons to import the PMQ file into Component Designer is to allow you to remove hardware that may be causing footprint issues^{xii}. Creating a SLD makes it easier to delete or disable dependencies; then rebuild and retest the component. Removing hardware from the SLD reduces the footprint of the eventual runtime.



```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Arati\Desktop>tap.exe
Target Analyzer Probe v2.00.1024.0

Copyright (C) Microsoft Corporation 2001. All rights reserved.

Category: ACPI

Device: Microsoft AC Adapter
Device: ACPI Fixed Feature Button
Device: Intel Processor
Device: Programmable interrupt controller
Device: System timer
Device: Direct memory access controller
Device: Standard 101/102-Key or Microsoft Natural PS/2 Keyboard
Device: ECP Printer Port
Device: Standard floppy disk controller
Device: System speaker
Device: PCI bus
Device: System CMOS/real time clock
Device: Motherboard resources
Device: Numeric data processor
Device: Microsoft ACPI-Compliant Embedded Controller
Device: Microsoft ACPI-Compliant Control Method Battery
Device: ACPI Power Button
Device: ACPI Lid
  
```

Figure 9: Partial Output when running TAP.EXE on Target Device

4.1.2 Create a Component for the Target Device

The target's PMQ file was imported into the Component Designer to create a Platform/Macro Component; this may take a few minutes. The 'Selector Prototype Component' was added as the Prototype. This allows you to override the component dependencies in the Target Analyzer. This newly created .SLD file was saved and imported into the Component Database Manage thus adding

the devices component to the database. This customized component will now be available for inclusion in a configuration.

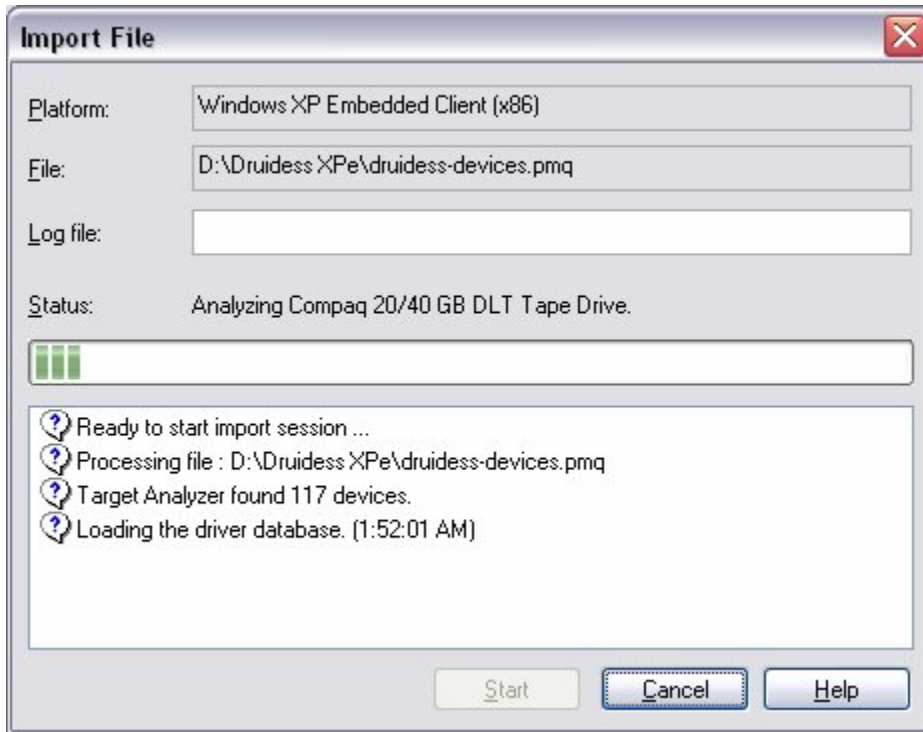


Figure 10: Importing the Target PMQ file as a Component

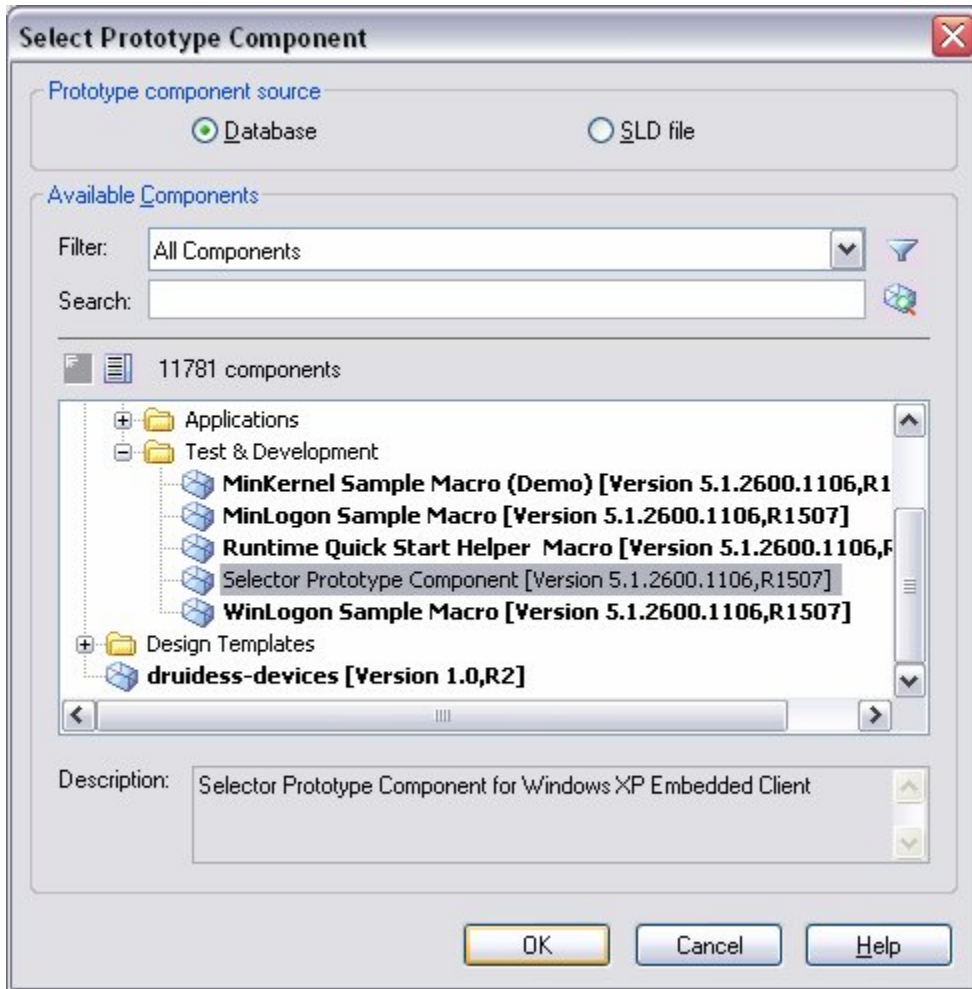


Figure 11: Adding the Selector Prototype Component as the Prototype

4.1.3 Creating a New Configuration in Target Designer

With the devices component created the runtime image can now be built. A new configuration was created in the Target Designer. The component browser lists all the available components. The component we created and other required components were dragged to the configuration editor pane. Additional components can be added to the configuration and if a specific component doesn't appear in the component browser list, try lowering the visibility level setting.

4.1.4 Update Configuration Settings

There are a number of components that have individual settings that need to be selected manually. The Target Designer does not have a one-stop pane that has the entire settings list so you must go through each of the components to check the settings individually. For example, with the Explorer Shell we can select what the end user is going to see by modifying the settings under the User Interface Core Component and check/uncheck the boxes to 'Show Control Panel on Start Menu', 'Show My Computer on Start Menu.', etc.

The Configuration Settings control how the image will boot and behave when running on the target. These settings need to be addressed now, before the image is built.

Once all the settings for the configuration and components are entered it's time to go ahead and run a dependency check.

4.1.5 Check Dependencies

Once the components have been added, removed and/or disabled from a configuration, it is time to run a check of the configuration to verify that all of the components have their individual dependency requirements satisfied. Certain components require the presence of other components to function properly.

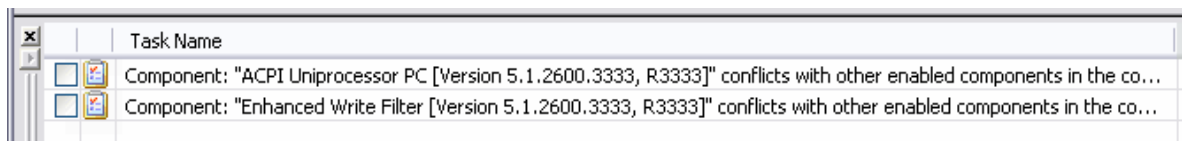


Figure 12: Resolve Dependencies by clicking on the Task names

A missing piece could cause the image or device to not work properly. For e.g. you cannot add 'Monitor' without adding 'VGA' to the configuration and have the image function. A dependency check will find these discrepancies.

Since 12,000 components and their dependencies make locating and including components a challenge. The 'Auto-Resolve Dependencies' was selected to automatically resolve dependencies when performing a dependency check.

4.1.6 Build the Run-Time Image

After resolving any dependencies issues for the configuration we are now ready to go ahead and select 'Build' and build the image. Images can be built as either a Release Image or a Debug Image. The debug image outputs debug information over a serial connection allowing developers to troubleshoot. Images are also categorized as either for deployment or test. To build for deployment you need to have a valid Product Key from Microsoft. Test builds have a live of 120 days.

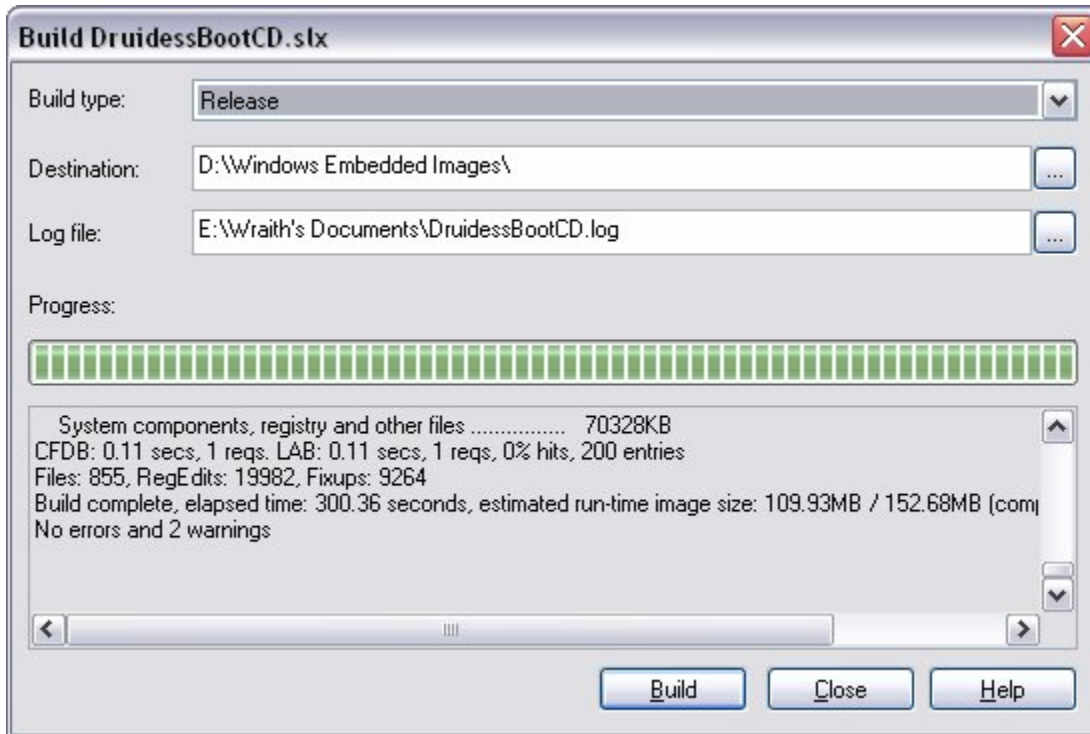


Figure 13: Building an Image

When the build completes an estimated size of the image is provided. The specified destination directory will contain the final image, which is similar to a standard XP Professional file structure. After the successful build the configuration was saved and exited. The final image is now ready to be downloaded to a boot media.

4.1.7 Deploy the Run-Time Image

To deploy the XP Embedded image you basically have to copy the files from the image folder on your development device to your target device's boot media. XP Embedded provides a variety of bootable media. Some of the bootable media include IDE hard drives, Compact flash, CD-ROM, Remote Network Boot, etc.

The main issue is how to physically get the image (with its long file names) from the host machine to the target. Some machines are limited in their I/O support so transfer becomes a bit more of a challenge. Devices like Compact flash and PCMCIA flash are easy to actually transfer but have tricky setup issues. The deployment method you choose will be dictated by the media and I/O support of your target machine.

When the XP Embedded runtime is booted for the first time a set of processes called the First Boot Agent is run. When FBA has finished processing everything, it cleans itself up and reboots the machine. Subsequent boots of the Windows XP Embedded runtime will not execute the FBA instructions again. FBA is a one-time operation, and usually happens on a golden master device before replication.

For this thesis two deployment methods were explored a USB boot and a XP Embedded demonstration using Microsoft's Virtual PC.

4.2 Booting from USB Flash

4.2.1 Overview

Booting from a USB disk has long been sought by the XP Embedded community, and it is now a new out-of-the box boot feature in XP Embedded SP2 Feature Pack 2007. The ability to boot

from small, inexpensive, and readily available flash disks opens up new possibilities to boot XP Embedded faster, deploy images more simply, and provide new, simple means to support systems in the field^{xiii}.

Even though the support is available in XP Embedded, this is only part of the solution. The target hardware, specifically the BIOS, plays a big role whether you can boot from USB 2.0. The target system must support USB 2.0 and the ability to boot USB 2.0 devices.

Like any other OS, XP Embedded is constantly writing data (e.g. registry data) to the disk. USB flash disks have the same flash life issues as CF devices. 200,000 erase cycles is a typical life-time for a flash device. Typically the Enhanced Write Filter (EWF) or the File Based Write Filter (FBWF) is implemented to protect the flash from constant writes by re-directing the writes to a RAM overlay.

Let's walk through the steps taken to implement this feature.

4.2.2 Format USB Media

Format the USB media using the `ufdprep.exe` tool found in the utilities folder under Windows XP Embedded Studio Tools. You can use `ufdprep /?` for more information on the utility. To use USB Boot in combination with Enhanced Write Filter (EWF), leave room on the USB flash disk for EWF to create the partition for the EWF^{xiv}.

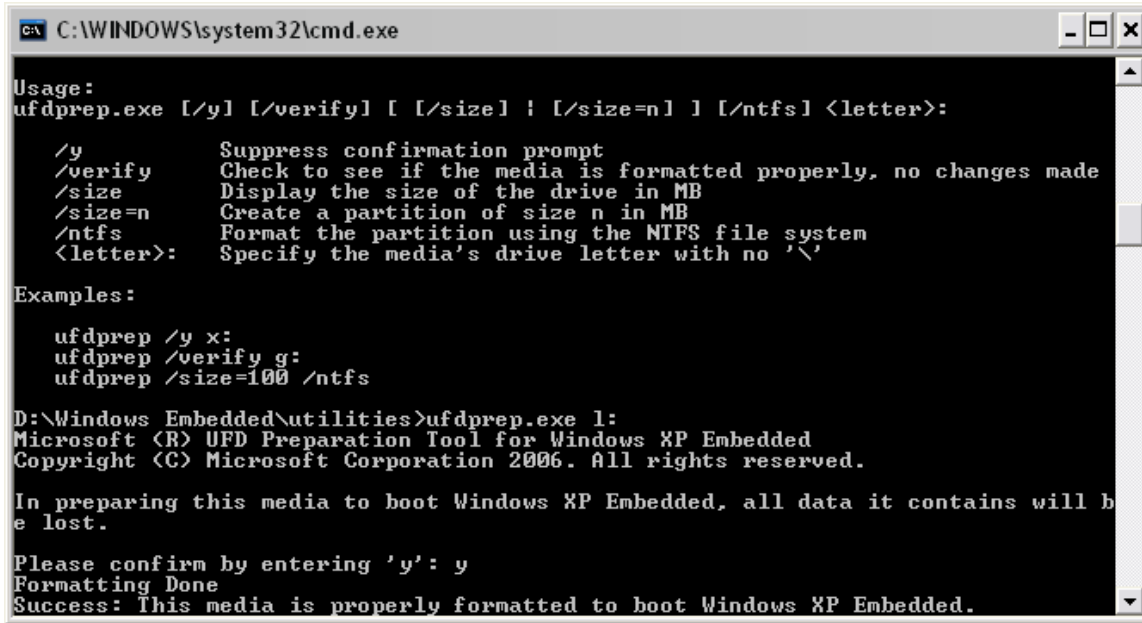


Figure 14: Format the USB Media with ufdprep.exe

4.2.3 Add USB Boot 2.0 Component

After generating, transferring and importing the .pmq file from the target device into the Target Designer we need to add the USB Boot 2.0 component to the run-time image. Resolve Component Dependency procedure until all dependencies are resolved.

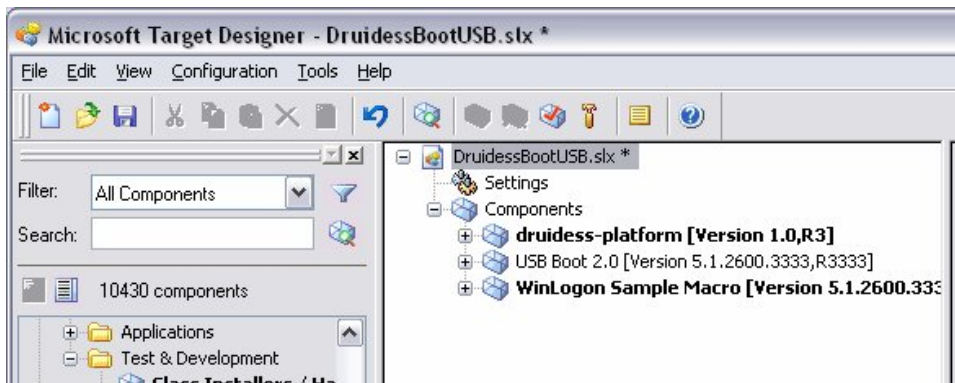


Figure 15: Add the USB Boot 2.0 Component

4.2.4 Build Target Image and Transfer to USB Media

Finally, build the target image and then transfer it to the USB Media (a simple copy-paste in Windows Explorer).

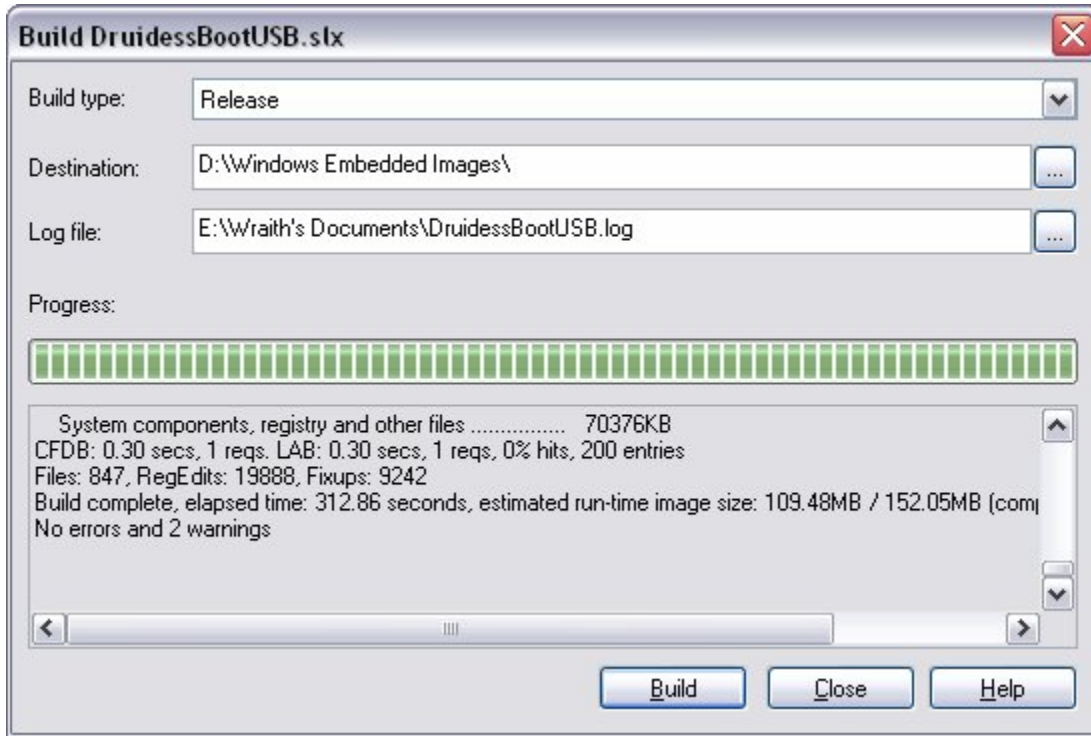


Figure 16: Build the USB bootable image

4.2.5 Boot with the USB Media

Set the BIOS in the target to boot USB 2.0, insert the USB flash disk into the target and let the system boot from the flash disk. The system will run through the First Boot Agent and then XP Embedded build will boot.

4.3 Demonstrating XP Embedded with Virtual PC

In a presentation demonstrating Windows XP Embedded's power and usability, it's common to only have a single machine available. You can use two partitions; however, you must then reboot your machine to switch from Windows XP Professional to Windows XP Embedded. In a presentation, this is not the ideal scenario. With Microsoft Virtual PC one can host a Windows XP Embedded image on a Windows XP Professional machine.

4.3.1 Setting up the Virtual Machine

- Create a new virtual machine with 300 MB HDD space, 128 MB RAM and Windows XP as the OS for the VPC machine. (XP and XPe have the same binary files). Boot the virtual machine with use the Windows PE CD (CD 1 of the Windows Embedded tools). Now we need to partition the disk so run `diskpart.exe` to partition the disk and type the following commands to create

a partition.

```
SELECT DISK 0
```

```
CLEAN
```

```
CREATE PARTITION PRIMARY
```

- Ensure the partition is active with the following command.

```
SELECT PARTITION 1
```

```
ACTIVE
```

- Once that's done type `Exit` twice to exit Diskpart.exe and reboot Windows PE.

After rebooting format the partition with `FORMAT C: /FS:NTFS /q`.

4.3.2 Capturing the Hardware Information

Run Target Analyzer tool (tap.exe) to automatically capture the information about the virtual machine's hardware. Tap.exe (Target Analyzer 32 bit version) is available on the Windows PE CD in the \XPe folder. To save the results of tap.exe in the host machine, create a mapped drive from the host machine inside VPC.

4.3.3 Building and Deploying the Image

Now that we have the devices.pmq we follow the regular process to build an image. Import the PMQ file into the Component Designer. Select the Prototype Component. Save the SLD. Open up the Target Analyzer and add in the components needed. Check the dependencies and then build the image.

We must now copy the XP Embedded image from our host operating system to the Virtual PC, using the networking of VPC in the same way we created the TAP output in the shared folder.

4.3.4 Running XP Embedded

Detach the CD and the virtual machine will boot the Windows XP Embedded image. The first boot will go through the First Boot Agent (FBA) process, which builds the machine registry, enumerates the device drivers, registers all components, and so on. At the end of FBA, the machine will automatically reboot. When the machine restarts, it will be running Windows XP Embedded

4.4 Summary

This chapter presented the process to create and deploy the Windows XP Embedded Images that were explored during this thesis research using the XP Embedded Development Studio tools. The next chapter will explain the customization, Servicing and Security Issues with XP Embedded.

CHAPTER 5 :

COMPONENTS AND COMPONENTIZATION

Why should you even use Component Designer? I mean, if Windows XP Embedded can run anything that will run on Windows XP, then why don't I just create an install program, and install it? That can work—as long as your run-time image contains all the support DLLs necessary to run the installer, and you're running on a Read-Write media, and you've got enough media space to handle the temporary files, and you've got enough memory and a big enough pagefile to handle the process. Since most embedded devices can't meet all these requirements, we need another way to put that functionality into a runtime. That's where components come in.

Components are easily defined: They are the smallest individually selectable pieces of functionality that can be included or excluded from an embedded run-time image, and are comprised of files, registry entries, and dependency information. In short, a component is a block of functionality. Windows XP Embedded comes with a database full of components—they're the most basic element in your configuration. The process of turning your application into a component is called componentization.

To exemplify componentization lets first look at the process to create a simple component for the Microsoft Paint application.

5.1 Component Development

Open up a new SLD file in the Component Developer and right click component to add a new component. Name the component 'MSPaint'. Under Group Memberships, right-click and add new group membership under the category: Category>Software>Applications>Other. Under Files right-click to add a new file. Browse to the mspaint.exe program in the system32 folder. Change the other details as shown in the figure below.

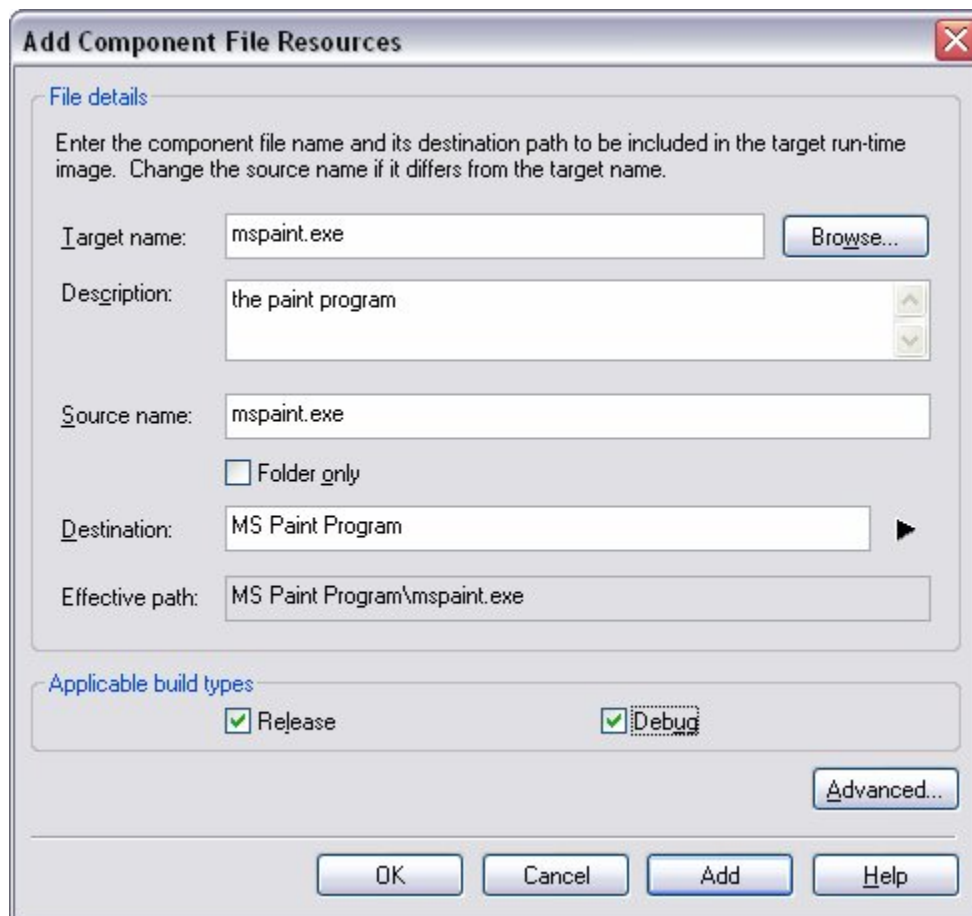


Figure 17: File Resource Details

MS Paint needs two other resources to run successfully so let's add them. Under 'Component or Group Dependency' add a new component dependency from the database: Software>System>System Services>Base>Primitive: OLE32, and click OK. Next, add the second component dependency from Software>System>System Services>Application Support>Microsoft Foundation Class Libraries (MFC), and click OK.

Now let's create a new repository. Create a new folder on the HDD and call it 'app' and copy the mspaint.exe program into the folder. In the Component Developer, under the Repositories branch as a new repository and in the details pane fill in a name for the repository and then set the Source Path for the repository to the 'app' folder you created on the HDD. Save the SLD file. Our component is now ready to be added to any configuration.

5.2 Advanced Component Development

Installation programs come in all shapes and sizes, from a simple xcopy, IExpress, through GUI installers that require registration information from the user and may ship on multiple CDs/DVDs—installation programs may install database engines or operating system services, device drivers or support DLLs, and can make changes to the operating systems registry or INI files—in some cases the installed files may be installed to multiple folders on your PC, Program Files for the core application, "\\Windows\System32" for device drivers, or other installer dependent

folders—determining what should be included into your Windows XP Embedded component would at first appear to be a non-trivial job.

Unfortunately, there are not any tools that can automate the process of building a Windows XP Embedded component for applications—there is some detective work that needs to take place. One of the tricks to componentizing an application or driver is simulating dynamic setup process data.

Most installers do three things:

- Copy files from a flat to a final location.
- Create registry keys based on install options, user input, or environment.
- Register COM objects and DLLs.

In order for Windows XP Embedded to accomplish these, we need to know a few things about the application we're trying to install:

- The files in the application.
- The registry keys used by the application.
- Any files not part of the application that are needed to run properly (such as runtime libraries, existing system services, and so on.)
- Any DLLs that need to be registered.
- Anything that requires user input or analysis of the run-time environment

5.2.1 Tools Used on the Development PC

The behavior of the application's installation program cannot be changed, but the operating system can be monitored to figure out what changed as a result of installing an application. There

are a number of tools available that can assist with this process. Since some installation programs require a reboot during the installation process, ideally a tool should be used that can monitor changes in the PC's file system and registry across reboots. Applications such as InCtrl5 (ZDNet), SysDiff (Microsoft), and others 'snapshot' the file system and registry before installing an application, then snapshot the file system and registry after the installation and provide a report of the differences—this can assist with determining which files should be included in the component. Don't forget that applications and services already running when the application installed may also make changes to the file system and registry.

Here's a list of tools that can be use on the development PC to determine which files/registry entries should be included in a custom component

- InCtrl5—used to snapshot the file system and registry and provide a list of differences (ZDNet)
- Dumpbin—used to determine which DLLs are used by an application/DLL (Visual Studio .NET)
- Depends—can also be used to determine which DLLs (and APIs) are used by an application
- RegMon—dynamically monitor changes to the registry (NT System Internals)
- FileMon—dynamically monitor the file system (NT System Internals)

5.2.2 Tools Used on the Target Device

Using the above tools to determine which files and registry entries are needed is only one piece of the puzzle. Once the XP Embedded image has been built, the application will also need to be

tested to ensure it behaves as expected—it's possible the application may be dynamically loading DLLs or creating instances of COM objects, but this can't be caught using Dumpbin or Depends. Therefore some debug tools on the target device are needed. Here's a list of tools that can be used on the target device (remember to remove them before deploying the final image):

- RegMon—dynamically monitor changes to the registry (NT System Internals)
- FileMon—dynamically monitor the file system (NT System Internals)

5.3 Component Support from Third-Party Vendors

Componentization is quite a task for larger applications. Even after building the image there needs to be a testing phase to ensure that you got all the little bits and aren't missing anything. Life would be a lot easier if you could find a pre-developed SLD from the application developer along with the application so that you have a stable component. There are a few third-party vendors providing software drivers or applications for XP Embedded.

Computer Associates' eTrust Antivirus for Microsoft Windows XP Embedded is the industry's first virus protection solution designed specifically for XP Embedded^{xv}.

Sygate's (now Symantec) Security Agent for Windows XP Embedded offers enhanced virus protection and anti-application hijacking, a dramatically reduced single-agent footprint, seamless integration with Microsoft's Target Designer and the availability of a run-time version^{xvi}.

Then there's Trend Micro's Network VirusWall 300 which integrates with Windows XP Embedded solutions to protect a spectrum of devices such as ATMs, Retail Point-of-Sale Terminals, Thin Clients and other network-connected systems from network viruses and internet worms^{xvii}.

The Windows Embedded Partner program lists more partners, products and services for XP Embedded at: <http://www.msweb.com/>. Besides vendors providing componentized versions of their applications there is the online XPe community which makes available pre-developed SLD files and components for hardware device drivers and software applications. The binary files have to be downloaded separately from the individual vendors. Some websites are:

<http://www.seanliming.com/>

<http://blogs.msdn.com/embedded/>

<http://www.xpefiles.com/>

5.4 Summary

Components are at the center of XP Embedded, and at some point you will probably have to build a custom component for your application or device drivers. Anything that will be in the final image should be placed in a Component.

XP Embedded's import features have shortened the development time needed to create components. What used to take days now only takes a few hours. There is still some analysis that needs to be performed, but there are various third party tools that make it easier to collect the

necessary resource information. And of course there are always companies and individuals that post XP Embedded components on the internet.

CHAPTER 6 :

MANAGING AND SERVICING RUNTIME IMAGES

6.1 Updates Overview

Windows updates are classified according to their purpose. The following table reviews Windows update terminology for Windows XP Embedded.

Term	Description
Update	A security bulletin that is issued to a wide audience.
Hotfix	A fix that is designed to resolve a specific customer issue. Hotfixes are not released to the public.
QFE (Quick Fix Engineering update)	A fix that is not related to security and that is issued to a wide audience.
SP1 XPE QFE	Any type of fix that is not associated with a major software release.

Table 1: Update Terminology^{xviii}

Updates for Windows XP Embedded are further classified by how they are deployed. The following table describes these types of updates.

Type	Description
Desktop update	An update that is applied, in part or as a whole, to an embedded device. Desktop updates are updates that are released for Microsoft® Windows® XP Professional. A servicing strategy that is based on desktop updates is a strategy of incremental updates to the run-time image.
Database update	An update that is applied to the component database from which the OEM takes Windows components to create a run-time image. A servicing strategy that is based on database updates requires that a run-time image be rebuilt or "reimaged," and then redeployed to apply the update to the embedded device.

Table 2: Update Type based on Deployment Method

6.2 Database Updates

A Windows XP Embedded database update is run on the development machine so that the Windows Embedded Studio component database is updated with the new or changed binary files. A database update ensures that all future run-time images that are created on the development computer include the update by default.

Updates that are made to the database cannot be uninstalled from a run-time image that was created to include them.

Updates should always be tested for the specific device to which they will be deployed before deployment is made, regardless of whether the update is implemented in a reimaged run-time image or by deployment directly to the device.

Database updates are made only for critical and important updates.

6.2.1 Creating a Database Update

To deploy a database update, the run-time image must be re-built after the update is made to the database. Installing a reimaged run-time image is the only way to apply updates to devices that boot from read-only media. The basic process that device developers use to re-image a run-time image is:

- In Target Designer, open the .slx file for the run-time image configuration to be updated.
- On the Tools menu, select Update to update the configuration with the changed component.
- Complete a dependency check and build a new run-time image that includes the new or changed component.

Even if the run-time image will not be reimaged and redeployed to distribute the update, the component database should be updated with the update to ensure that run-time images that are generated in the future use the correct components.

6.2.2 Advantages of a Database Update

There are some practical advantages to distributing an update by re-imaging. For example, you can more effectively test a reimaged run-time image for missing dependencies and other problems. It can also be more practical to re-image when you have a number of updates to apply and want to avoid the time involved in scripting and administering a series of incremental updates.

6.2.3 Issues with Database Updates

The technique of re-imaging to include an update limits the means of deployment that are practical to use to install the update on devices. Servicing solutions are not generally intended to be used to deploy entire run-time images and the mass deployment of a run-time image over a network demands a lot of resources, even under the best of circumstances. Distributing an update in a reimaged run-time image also means that the boot media must be replaced or reformatted.

6.3 Desktop Updates

Desktop updates are updates that are released by Microsoft for Windows XP Professional. Since XP Embedded is a componentized version of the Windows XP, these Windows updates can run on Windows XP Embedded run-time images that include the required dependencies.

Desktop updates are used to apply incremental updates to a run-time image. Combined with good testing, the incremental application of desktop updates reduces the size of the update package and the cost of deploying them.

6.4 Servicing Embedded Devices

Device servicing is the deployment of new and changed binaries to embedded devices. A Windows XP Embedded-based device can be serviced by:

- Re-imaging: Updating the Windows Embedded Studio component database with the new and changed binaries, and then rebuilding and redeploying the run-time image.
- Making incremental updates: Updating the run-time image on the device by deploying the new and changed binaries.

Servicing plays a critical role in any device security strategy but is not the sole component of a security strategy. The design process should include a means for run-time management and servicing. Servicing solutions include mechanisms for patching and updating embedded devices, as well as mechanisms for deploying reimaged XP Embedded images.

The servicing solutions that Windows XP Embedded supports for making incremental updates to a run-time image include:

- Microsoft Windows XP Embedded Device Update Agent (DUA)
- Microsoft Software Update Services (SUS)
- Microsoft Systems Management Server (SMS)

When a run-time image is deployed on a device, the run-time image can be managed remotely and serviced with security patches and Quick Fix Engineering (QFE) updates. Before you can use any of the automated managing and servicing features, they must be added to your run-time

The following table provides a quick reference and comparison of the Windows XP Embedded servicing options^{xix}.

Solution	Best suited to...
Device Update Agent (DUA)	Any networked device. Ideal for small-footprint devices or smaller environments.
Software Update Services (SUS)	Any device on a closed network. Ideal for medium to large enterprise environments.
Systems Management Server (SMS)	Well-suited to RPOS, WBT, and ATM devices. Ideal for enterprise environments that require management capabilities.

Table 3: Servicing Options

You can use one of the supported servicing solutions (DUA, SMS, or SUS) or you can create your own servicing solution. Below is an overview of various solutions and a detailed look at the Device Update Agent.

6.4.1 Recovery CD

In some severe circumstances, such as when a run-time image is corrupted, it may be necessary to start your device from a bootable CD. To create a recovery CD, you can package Microsoft Windows Pre-Installation Environment (Windows PE), a System Deployment Image (SDI) file, and SDI tools. With this recovery CD, you can use Windows PE to boot the device, and then restore the image on your device by installing an image from an SDI file. Once the recovery image is installed,

you can reboot the device from the recovery image. This recovery process does not repair individual files; rather, it installs the complete image as it was on the device when it left the factory. All existing files, data, and settings on the device, as well as any changes made since it left the factory, are replaced by the image.

With a few modifications to this process, you can use Windows PE and SDI to deploy Windows XP Embedded to a device or upgrade an existing device in the field. The recovery CD is not a method for deploying or upgrading individual components, settings, or QFEs.

6.4.2 Run-Time Image Replacement

You can service a run-time image by replacing the entire run-time image with an updated version. The updated run-time image includes all of the fixes, additions, and updates that are required to service the run-time image. For some types of devices, such as bootable CD-ROMs or network-hosted images, this is the only available method of servicing a run-time image.

In Target Designer, add Quality Fix Engineering (QFE) updates, component updates, or other additions to your configuration, and then rebuild and test the configuration. Once the run-time image is verified, you can distribute it to your devices.

One of the benefits of completely replacing a run-time image is that all dependencies are met, updates are applied, and other than updating the device with the new run-time image, there are no additional steps.

6.4.3 Remote Management

In some cases, you cannot gain physical access to the device once it has been deployed. Windows XP provides a comprehensive set of features to help administer a target device, including Remote Desktop (Terminal Services) and Telnet, which helps to manage your system from a remote location. Other services and features include File Transfer Protocol (FTP), Windows Management Instrumentation (WMI), and Simple Network Management Protocol (SNMP)^{xx}. You can also create a Windows-based Terminal (WBT) that you can manage remotely. A remotely managed system must recover automatically from an error. You can use message interception to acknowledge system errors without requiring user interaction.

6.4.4 Device Update Agent

Device Update Agent (DUA) is a lightweight management solution for small footprint devices. It enables organizations to update a run-time image remotely. DUA is a service that runs on your run-time image and performs administrative tasks, such as copying files, creating registry. The DUA is discussed in detail later in this chapter.

6.4.5 Software Update Services (SUS)

Software Update Services (SUS) allows you to configure your own intranet update server that downloads Windows Updates from Microsoft. Implementing SUS servicing requires a separate SUS intranet server that manages the updates. This SUS intranet server is set up by an administrator to poll the public Microsoft Windows Update web site for updates for Windows XP Professional systems. If there are new updates available, these updates are downloaded to the SUS intranet server.

Before these updates are applied to the XP Embedded devices, they must be approved by an administrator.

Because the updates are specific to Windows XP Professional or Home editions, an administrator must inspect the update, verify that it applied to the Windows XP Embedded device, and test the update. After the update has been successfully tested, it can be made available to the Windows XP Embedded clients^{xxi}.

Although it is possible to configure client devices to directly download updates from the public Microsoft Windows Update web site, doing so may potentially corrupt the device. Because Windows Updates are specific to Windows XP Pro or Home editions, some of the updates may not apply to Windows XP Embedded.

The following illustration shows the overall process flow for delivering and approving updates using SUS.

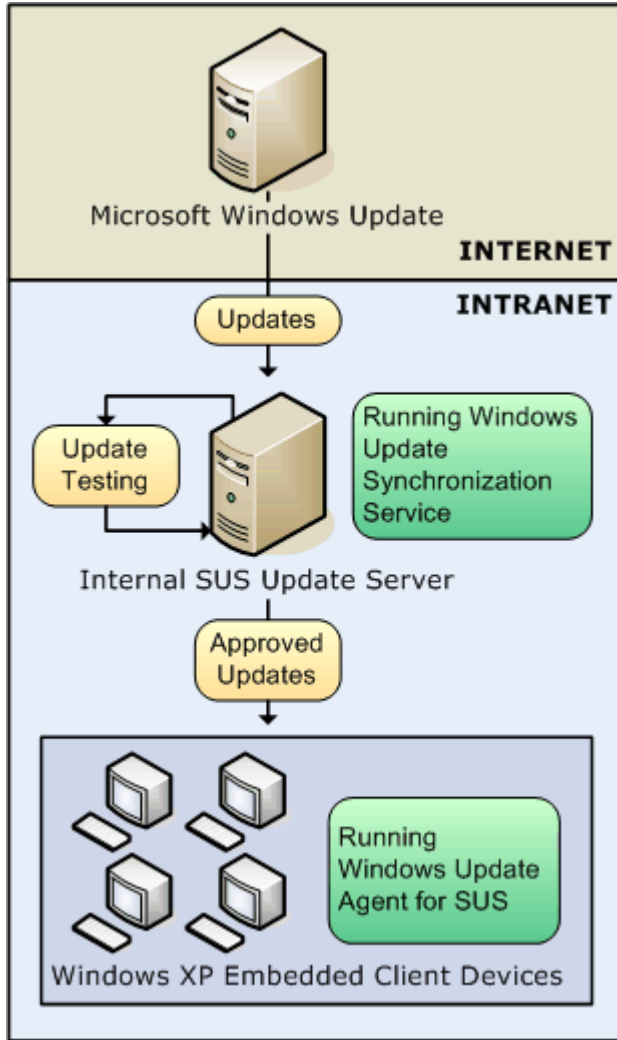


Figure 18: SUS Overall Process

The basic process flow is detailed in the following list:

- The SUS intranet server polls the public Microsoft Windows Update site for Windows XP Professional updates.
- The SUS intranet server downloads any new updates.
- After new updates have been downloaded, an administrator must verify that each update applies to the Windows XP Embedded device. The administrator must then test each update on a sample device.

- After the update has been verified and tested, it is made available on the SUS intranet server to be downloaded by the Windows XP Embedded clients.
- At the next scheduled polling time, the Windows XP Embedded clients download the updates from the internal SUS server.

6.4.6 Systems Management Server (SMS)

You can use Microsoft's Systems Management Server (SMS) 2003 to manage networked Windows XP Embedded-based devices alongside Windows desktop, Windows server, and other Windows Mobile systems. SMS supplies an administration console that can be used on a Windows Server 2003 system to remotely perform management functions.

Client and server components for SMS are not included in the Windows Embedded Studio component database and must be separately obtained. The SMS Advanced Client component runs on the Windows XP Embedded device and reports system status to the server. Windows XP Embedded requires the SMS 2003 Advanced Client; earlier versions of SMS are not supported.

The SMS 2003 Advanced Client must be added to the configuration in Target Designer before the run-time image is built and deployed.

SMS provides flexible support for applying any kind of update package that the run-time image can execute, including desktop updates, custom scripts, and application-specific executables. If the Windows Installer Service component is included in the run-time image, you can also use SMS to deploy Microsoft Windows Installer (MSI) packages for installation.

6.5 Device Update Agent

The Device Update Agent (DUA) is available with XP Embedded SP1 onwards and so, unlike the other two solutions, it's free. This thesis explores the DUA component.

The Device Update Agent enables you to update a run-time image remotely. DUA is a service that runs on your run-time image and performs administrative tasks, such as copying files, creating registry keys, or executing processes. DUA polls a specific remote or local path for a script file. When this script file is found, the script immediately runs.

Device Update Agent includes a script compiler, `duisc.exe`. This compiler is used to compile script files to a program that the DUA service can execute.

In some cases, you cannot gain physical access to your device after it has been deployed. You can place commands in the DUA script, run the scripts through the script compiler, and then place the resulting file into a folder that is monitored by the Device Update Agent service. In this manner, you can send commands remotely to your device.

The DUA uses simple command sequences that are pulled from the file. The registry is configured with the name of the folder from which the Device Update Agent gets the command file. You can also manually copy the command file to the folder that is monitored by the Device Update Agent service. Typically, this file is delivered to the DUA over a network or on physical media. If the

device running the DUA is connected to a network, the DUA can download additional command files.

6.5.1 Using Device Update Agent

Let's quickly go over the steps needed to create a DUA updateable run-time image.

1. Create and Compile the Device Update Script: First create a Device Update Script (a .DUS file) using a text editor which, when run by DUA, will create a new directory on the device, copy files, and run the required executable for the update. A sample DUS file (helloworld.dus):

```
//Delay 60 seconds.
```

```
2,0,60
```

```
//Create a new DUA_Test Directory.
```

```
4,,,c:\DUA_new
```

```
//Copy helloworld files to the new directory.
```

```
7,,,c:\DUA_polling\helloworld.exe,1,C:\DUA_new\helloworld.exe,
```

```
7,,,c:\DUA_polling\msvbvm60.dll,1,C:\DUA_new\msvbvm60.dll,
```

```
//Execute HelloWorld.
```

```
15,0,0,2,C:\DUA_new\helloworld.exe,0,,0,0,0,1,0,0,,1,0,0,,0,2,C:\DUA_new,1,0,WinSta0\Default
```

Create a directory to house update's file and its dependencies. Save the helloworld.dus in that directory. Compile the Device Update Script with: `dusc helloworld.dus` which will generate a

file called `commands.dup`. When DUA polls local media, it deletes the DUA update file. Saving a backup `.DUP` file allows you to edit and recompile the `.dup` file when you troubleshoot DUA.

When done your DUA polling directory should contain the script file (`.dus`), the compiled script (`.dus`) and the required file and its dependencies (`.exe` and/or `.dll`).

2. Creating the DUA Run-Time Image: In your runtime image you will need to include the Device Update Agent and configure the DUA component settings depending on your scenario (HTTP Polling or Local Polling, etc). For this scenario we'll use local polling so set Complete Path to the Command File including Filename and Extension: to `C:\DUA_Polling\helloworld.dup`.

3. Deploy the DUA Run-Time Image: Build the runtime image. Copy the DUA Polling directory created in step 1 to the root of the runtime image. Deploy the image to the target. Boot the target. After 30 seconds the `helloworld` executable will run and DUA will update the runtime image.

6.5.2 Web Server Configuration

You can host DUA updates from any type of Web server, however when you configure your Web server, it is important to note the following design considerations:

- You must include the `.dup` extension as a registered MIME type.
- It is strongly recommended that you use HTTPS to improve security when transferring the update. You should create a user account on the Web server that is used exclusively by DUA to access the update file. By using an exclusive user

account, you can reduce the risk of tampering with, or interception of, the update.

- The files on the server must have the appropriate access permissions.

6.5.3 Security Issues

Security is an important consideration when you deploy the Device Update Agent (DUA) component. DUA runs under a user context with sufficient privileges to perform commands that can damage the device run time. For this reason, it is extremely important that DUA only acquire and process command scripts that are intended for the device by the device manufacturer.

DUA offers the following configuration settings to enhance security when it transfers and executes command scripts. Consider the following security features and issues when you use the Device Update Agent (DUA):

- Poll locations: DUA can be configured to poll local or remote command paths. In either case, ensure that poll locations cannot be compromised. Command files should come from a known source and should not be tampered with once they are deployed to the command file path poll location.
- HTTPS transfer: Command files can be retrieved from a remote server using HTTPS. The HTTPS option allows devices to open secure transfer sessions with a specified update server. To configure DUA for HTTPS, select the HTTPS option under Advanced Settings. When you use remote HTTP or HTTPS command paths, it is important to evaluate the communication channel between the device and the remote update server. There may be cases when HTTPS is

not the desired method of channel security. Alternative methods can be used to secure the channel between the device and the server, such as virtual private network (VPN) connections.

- **AutoLogon Options:** DUA can be configured to authenticate with a remote update server. If HTTPS is not specified, credentials are transferred in clear text. To configure the AutoLogon options, expand the Security Settings section. The following table shows the AutoLogon levels to specify when DUA should provide credentials to the server.

Value	Level	Description
0	Medium	Use default credentials for intranet requests only.
1	Low	Use default credentials for all requests.
2	High	Never use default credentials.

Table 4: AutoLogon Levels

6.6 Summary

Many developers try to use Install Shield or standard device management tools to update applications and device drivers. Componentization is a powerful feature in XP Embedded, but where smaller images are concerned, componentization limits the usefulness of these installation methods for updates.

Device Update Agent with its small footprint and powerful command set is a great way to update applications and device drivers for systems in the fields.

CHAPTER 7 : SECURITY

CONSIDERATIONS

When you create a run-time image for an embedded device, you must consider how your device will be used, and the security threats it is vulnerable to. It is imperative to add safety measures to guard against network attacks, such as worms or viruses, as well as local attacks, such as unauthorized access. Configuring security mechanisms can increase your protection against such attacks and reduce the amount of downtime when devices are deployed in the field.

You can improve the security of your system by adding additional security components to your run-time image. By increasing the security of your run-time image, you can prevent attacks from hackers, viruses, or other unauthorized access. Before you deploy your run-time image, it is important to understand the security threats that your device is vulnerable to, and to add the appropriate security and servicing components to your configuration.

Updating your devices frequently can be expensive; however, servicing can prevent even more expensive security compromises. We have already discussed various servicing options in the earlier chapter. Let's have a look at some general guidelines on how to reduce the risk of a security compromise.

7.1 Assessing your security risk

Some factors to consider when you assess your potential security risk are as follows:

Network environment: Devices that are connected to a network might be vulnerable to network-based attacks, especially if these devices have unrestricted access to the Internet. You help to mitigate this risk by connecting your devices to a corporate network, or—even better—by restricting both incoming and outgoing Internet traffic. For more information about technologies that can help you to mitigate the risk of exposure to network-based attacks, see "Building in Windows Security" and "Securing the Network" later in this document.

Physical environment: Any kind of direct physical access to your devices by a malicious user—a user who intentionally accesses a system with the intent to cause harm to the system or to use it in an unauthorized manner—presents an obvious risk. For more information about technologies that can help you to reduce this risk, see "Securing physical media" later in this document.

Data storage: Because embedded systems run operating systems that have a small footprint, it is best not to store critical data on them. Instead, store critical data on a different computer, a server that is connected to the network, or on embedded devices whose operating systems have a larger footprint. Limit the amount of data that you store on a device running Windows XP Embedded so that the device works normally and achieves your performance goals.

A number of security-related factors are taken into account during the design of an embedded operating system, including:

Footprint: The larger the footprint of the embedded operating system, the more surface area that is vulnerable to attack. It is recommended that you choose an operating system that has the smallest footprint possible and can still meet your needs. Devices running operating systems that have small footprints also tend to perform faster due to the small size of the media that they use, and the small number of files that they must process, load, and catalog.

Services and features: The more services and features that you enable on a device, the more surface area that is vulnerable to attack. Again, the minimum set of features and functionality that meets your needs is recommended.

Built-in security features: You can use Windows Firewall, WinLogon, Group Policy, and Access Control Lists (ACLs) to secure Windows XP Embedded. In Windows-based systems, an ACL is a list of access control entries that apply to an entire object, a set of the object's properties, or an individual property of an object, and that define the access granted to one or more security principals.

7.2 Reduce your security risks

Some ways that you can help reduce security risks to your devices include the following:

Building small images: Devices with small footprints have less surface area exposed to attack.

Managing non-essential services: Some services are unnecessary on certain devices and should not be built into the operating system, or should be turned off or disabled. You can also configure services to start either automatically or manually. You can configure a service to start manually, or to be managed by the device itself, if the service must be installed on the device but poses potential security vulnerability.

Using Windows Firewall: The Windows Firewall is a port-based firewall service that blocks incoming traffic to your device on specific ports. Windows XP Embedded contains a Windows Firewall component that implements this functionality

7.3 Building in Windows Security

Security features in XP Embedded can help reduce potential data loss or compromise by either communicating directly with a device, or by communicating with a device over the network. XP Embedded offers two distinct logon base components, with distinct security models:

The MinLogon component, which is unique to Windows XP Embedded, provides faster boot times and a smaller operating system footprint at the expense of built-in security features. There are no users on a device that use the MinLogon component: Programs run in the Local System context, which provides all users with complete control over the operating system. Security features such as Group Policy settings, logon rights, and ACLs are not necessary in this context, because there are no users.

The Windows Logon (Standard) component, also referred to as the WinLogon, component embodies the same standard logon mechanism as used in Windows XP Professional. Devices that use WinLogon are somewhat larger and slower to boot than devices that use MinLogon; however, WinLogon uses the full spectrum of Windows security features. Security features such as Group Policy settings, user logon rights, and ACLs are implemented in this context.

Some additional security considerations are:

Microsoft Active Directory service: Active Directory provides a centralized, distributed computing infrastructure with built-in security. Devices running Windows XP Embedded can participate in an Active Directory infrastructure by including the appropriate Active Directory components.

Group Policy: If your devices run the WinLogon service, you can manage users and security groups by configuring Group Policy settings in XP Embedded.

Credential management APIs (Application Programming Interfaces). Windows XP and Windows XP Embedded provide the APIs that you need to implement custom credentials

management applications. You can use these applications to manage user credentials instead of relying on users to type their user names and passwords.

Smart cards: Windows XP Embedded supports smart cards, including integrated Smart Card security management and Smart Card reader device support.

7.4 Securing physical media by using Enhanced Write Filter

Protecting the physical storage media of your devices is critical to avoiding data corruption from outside sources and computer viruses. Windows XP Embedded provides the Enhanced Write Filter (EWF) component to help protect your physical storage media.

EWF helps to protect the contents of a volume on the physical media by redirecting all writes to a different storage location, called an overlay. Used in this context, an overlay is similar to a transparency overlay on an overhead projector. Any change made to the overlay affects the picture as seen in the aggregate, but if the overlay is removed, the underlying picture remains unchanged. EWF can protect one or more bootable and non-bootable disk volumes, including but not limited to hard drives, flash ROMs, and CDs formatted in the El Torito format.

EWF presents a servicing challenge, however. To service the underlying operating system or application that EWF helps to protect, you must first disable EWF. This challenge is reduced by the availability of the EWF API, which provides programmatic control of EWF from inside your own applications.

7.5 Protection from Computer Viruses

A number of Microsoft partners provide antivirus functionality for Windows XP Embedded. As mentioned earlier, Computer Associates offers a software-based solution and was first-to-market with a Windows XP Embedded-based solution. Trend Micro followed shortly thereafter with a hardware-based solution.

The Computer Associates' eTrust is well componentized and so depending on the bells and whistles you want you can get the local scanner for 5.4MB all the way up to the full product at 21MB which gives you dual engines and management by an enterprise server^{xxii}.

7.5.1 EWF is NOT an Anti-Virus Solution

A caveat worth mentioning, EWF as an anti-virus solution is NOT recommended.

The usual security plan is: start with SP2 FP2007 then, add the firewall, antivirus and servicing. Some developers however skip the antivirus due to footprint requirements and decide to use EWF as an AV solution. Recall that the Enhanced Write Filter (EWF) feature of Windows XP Embedded makes it possible for you to write-protect your run-time images. This is not always a secure or good idea.

Consider a machine running EWF RAM gets infected and the device is protected only in the sense that the system files are not permanently corrupted. In the meantime, until you reboot the device it could be:

- Consuming resources, trying to write to disk which fills up the ram overlay and eventually the machine runs out of memory and dies.
- Acting as a 'zombie' or host, infecting other machines on the net.

Now after you reboot this device, the machine is no longer infected, but more than likely it's going to be infected again and the same issues above apply until the next reboot.

While this is bad, this scenario could get far worse. Consider the machine is infected, you don't realize it yet but you need to commit some changes to disk. You commit the changes in the overlay to disk and you've now permanently written infected files. Rebooting will still leave you in an infected state. Now you either need to re-image the device or install AV software, clean the disk, commit those changes and cross your fingers.

7.6 Summary

In this chapter we looked at various security considerations. Security should be easy to implement. XP Embedded provides a toolset to help in building a more secure runtime image. A security plan is crucial and security continues after deployment with device servicing as no device design is complete without a servicing option.

CHAPTER 8 : COMPARISON WITH OTHER EMBEDDED OPERATING SYSTEMS

An important consideration when comparing Windows XP Embedded and other Embedded Operating Systems is that XP Embedded is designed to run on a PC-based architecture. Besides XP Embedded, the other Embedded Operating systems that fall into the PC-Based architecture are Windows CE, Windows Embedded NT and Embedded Linux.

8.1 Windows CE

On initial inspection, Windows CE and Windows XP Embedded may seem to be similar, since both are componentized operating systems, both expose similar programming interfaces (Win32, MFC, ATL, and support for .NET applications), and both expose similar operating system technologies, which include support for networking, internet browsers, media players, and so on. Using the table below, depending on the target device you can select the right embedded OS.

Targeted Device	Windows XP Embedded	Windows CE .NET
Mobile Clients		
PDAs		✓
Smartphones		✓
Internet and Media Appliances	✓	✓
PC Companions		
Digital Cameras		✓
Printers and Scanners	✓	✓
Thin Clients		
Retail Point-of-Sale (RPOS) Devices	✓	✓
Windows-Based Terminals	✓	✓
Connected Clients		
Basic Set-top Boxes		✓
Advanced Set-top Boxes	✓	
Basic Residential Gateways and Servers		✓
Advance Residential Gateways and Home Servers	✓	
Industrial Controls	✓	✓
Voice Over Internet Protocol (VoIP) Phones		✓

Table 5: Recommended Windows Embedded Operating System by Device Category

Devices such as mobile handhelds and basic residential gateways require a small footprint, efficient power management, and remote management capabilities along with the ability to deliver rich user experiences, making Windows CE .NET the recommended operating system for smart, connected and small-footprint devices.

Devices such as advanced set-top boxes and retail point-of-sale clients require the latest security and reliability features, familiar and powerful Windows features, and are less restricted in terms of footprint.

Windows XP Embedded is the recommended operating system for delivering the power of Windows in componentized form.

8.2 Windows Embedded NT

Microsoft Windows NT Embedded was a reintroduction of a Microsoft desktop operating system for the embedded market. Windows XP Embedded builds on the momentum of Windows NT Embedded with improved features and better performance.

With Windows NT Embedded, you had to make a guess as to what platform components were required in the OS image. The best way to ensure that you had the correct computer and other driver components was to install the Windows NT OS on the target systems. Sometimes, you would have to do this several times to gather all of the necessary information. Windows XP Embedded simplifies this process substantially with Target Analyzer.

Creating components has greatly improved in Windows XP Embedded. Device driver components took up to two days to develop in Windows NT Embedded. Windows NT Embedded had a very small number of device drivers, which forced developers to create three to five device driver components per system. The latest drivers for most hardware are already included in

Windows XP Embedded, thus saving time on the number of device driver components that you must create. If there is a driver that is not in the database, the INF import feature can create a new component from scratch, which lowers component development time from days to hours.

With a dramatic number of additions, enhancements and improvements in the core OS, tools and platform features, Windows XP Embedded offers superior performance than Windows NT Embedded. As PC technologies continue to evolve, Windows XP Embedded will support the latest.

8.3 Embedded Linux

The power, reliability, flexibility, and scalability of Linux, combined with its support for a multitude of microprocessor architectures, hardware devices, graphics support, and communications protocols have established Linux as an increasingly popular software platform for a vast array of projects and products.

Because Linux is openly and freely available in source form, many variations and configurations of Linux and its supporting software components have evolved to meet the diverse needs of the markets and applications to which Linux is being adapted. There are small footprint versions and real-time enhanced versions. Despite the origins of Linux as a PC architecture operating system, there are now ports to numerous non-x86 CPUs, with and without memory management units, including PowerPC, ARM, MIPS, 68K, and even microcontrollers.

Since Linux is open source there is a caveat worth mentioning. It is not public domain software. It is licensed according to the GNU General Public License, which has a strict set of rules for use.

If you take the Linux kernel or any Linux utility and modify it, port it, or add features to it, you must make the source available to anyone who asks for it. If you are not careful you may give up the rights to your proprietary software unintentionally. It is best to always consult your attorney on all open-source copyright issues.

Microsoft had published documents pitting Windows XP against Embedded Linux with the emphasis on the technical and business inferiority of Linux. Embedded Linux distributors argue that the document not only distorted the value of Linux, but contains inaccuracies. While it's beyond the scope of this thesis to verify the validity of the claims there are some points that can be commented on about Embedded Linux that are mentioned in another document by Microsoft: "The Windows Embedded Advantage-Comparing Windows Embedded with Linux"^{xxiii}

Kernel Size – The smallest footprint configuration of XP Embedded is 5MB^{xxiv} giving extremely limited functionality. An average-sized configuration is about 40MB^{xxv}. The footprint size for LynuxWorks' BlueCat Linux is 259KB and LynxOS real-time operating system product is 254KB. An "extremely limited functionality" version of LynxOS is at about 150 KB^{xxvi}.

Development Environment – Contrary to Microsoft's claims Embedded Linux does indeed have an integrated tool set. Some commercial Embedded Linux vendors sell Windows tools supporting Embedded Linux, capitalizing on those companies reluctant to throw away their

Windows development environment. LynuxWorks VisualLynux is a Visual Studio plug-in that allows developers familiar with Visual Studio to program for a Linux target within a Windows.

In the Embedded Linux market, the Eclipse IDE has become very popular. Commercial Embedded Linux vendors such as Timesys and Wind River build their own tools around the Eclipse framework. Then there's MetroWerks Code Warrior an embedded IDE that has been available for Linux for a while.

Security & Reliability– Linux by design and heritage is a lot more secure than Windows and Microsoft's 'Install now, Patch Later' remedy model. Linux is also reliable and stable and has been proven on the Internet. Windows XP on the other hand has greatly improved stability but still doesn't measure up to the standards set by Linux.

Linux is a versatile and cost effective operating system for embedded systems It can be embedded in a surprisingly small system to handle simple tasks and scaled up to handle more complex tasks. Linux can run on most microprocessors with a wide range of peripherals and has a ready inventory of off the shelf applications. Development cycles are shortened through the use of mature tools, open source code, substantial documentation and available support services.

The learning curve required to setup a Linux machine is no longer as steep as it used to be, but it's still not close to the Windows learning curve which gives Windows a big boost over Linux. Microsoft definitely scores big points on familiarity too.

8.4 Summary

The choice of OS from Microsoft's Embedded Platform depends largely on the target device and requirements. Windows CE is geared towards handhelds and XPe is geared towards PC-based devices. Linux can be installed on a variety of devices and has a lot of pros as well as cons. A thorough survey of needs to be done before choosing an operating system for your embedded device since each OS has its own strengths and weaknesses.

CHAPTER 9 :

CONCLUSION

In this thesis we looked at a soup to nuts development process to develop and deploy an XP Embedded Image. Developing a product, of course, is more than just creating and building an image. What goes into the image, what the image runs on, and how the product is manufactured are all points to consider. So, we looked at creating customized components and then servicing and security considerations. Lastly we looked at other embedded operating systems available.

In conclusion let's have a look at the future offerings of Microsoft's Embedded Platform.

9.1 Windows Vista for Embedded Systems

Windows Vista for Embedded Systems was released December 2006^{xxvii}. The Windows Vista for Embedded Systems family includes two product choices: Windows Vista Ultimate for Embedded Systems and Windows Vista Business for Embedded Systems.

Unlike the Windows Embedded operating systems Windows Vista for Embedded Systems is not a componentized offering and the image is not customizable by the developer. Windows Vista for

Embedded System was designed for the desktop but is available for fixed-function or dedicated embedded systems. So in essence you get the whole of Vista, with no specific embedded features. Microsoft is also working on a componentized version of Vista to come out in the future but no release date has been set.

Recently Feature Pack 2008 for XP Embedded was announced. Many Vista desktop technologies have been back-ported to run on XP embedded so developers building embedded devices today can take advantage of Vista technologies.

9.2 Summary

The goal of this thesis was to bring together the different options and considerations that needed to be taken into account to customize and secure an embedded device running on XP Embedded. Projects are full of choices and the most overlooked step in any project is taking the time for adequate planning and documentation.

XP Embedded offers a different approach to the traditional embedded OS. Using a popular OS software package with a large support community opens a world of possibilities and options for development. Although Microsoft Vista for Embedded Device is now available, until the componentization version of Vista is released, Windows XP Embedded will still be at the head of the Embedded Platform line.

REFERENCES

- ⁱ David Ursino, Windows Embedded Family (Microsoft Corporation White Paper, September 2003), 1
- ⁱⁱ Patric Dove, 'Windows XP for Embedded Applications' (Advantech Corporation White Paper) ,
<http://www.automation.com/sitepages/pid2602.php>
- ⁱⁱⁱ Windows XP Embedded Service Pack 2 Feature Pack 2007 Evaluation,
<http://www.microsoft.com/downloads/details.aspx?familyid=9BDF1DEA-A37E-4D25-83DF-AABBAA78914F&displaylang=en>
- ^{iv} Differences between Windows XP Embedded and Windows XP (Microsoft Corporation White Paper), 4
- ^v Developing an Embedded Run-Time Image from Start to Finish (MSDN Library),
<http://msdn2.microsoft.com/en-us/library/ms838337.aspx>
- ^{vi} License and Ship Your Device (MSDN Library), <http://msdn2.microsoft.com/en-us/embedded/aa731305.aspx>
- ^{vii} Windows XP Embedded (Embedded System Engineering Magazine),
http://www.esemagazine.com/index.php?option=com_content&task=view&id=190&Itemid=2
- ^{viii} J. Fincher, Running Free: Runtime Basics (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa459165.aspx>
- ^{ix} J. Fincher, Component Database Manager (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa459159.aspx>
- ^x J. Fincher, Target Designer, Inside and Out (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa459162.aspx>
- ^{xi} Windows XP Embedded Overview and Benefits (Microsoft Website),
<http://www.microsoft.com/windows/embedded/eval/xpe/default.mspix>
- ^{xii} J. Fincher, Running Free: Runtime Basics (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa459165.aspx>
- ^{xiii} Sean D. Liming and John R Malin, Booting XP Embedded from USB Flash (CA, SJJ Embedded Micro Solutions, LLC., September 2006), 4
- ^{xiv} Booting XPe from USB (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa940960.aspx>

^{xv} CA Delivers Industry's First Virus Protection Solution For Microsoft Windows XP Embedded With eTrust Antivirus (Computer Associates Press Release), <http://www3.ca.com/Press/PressRelease.aspx?CID=63405>

^{xvi} Sygate Introduces Next Version Of Sygate Security Agent For Windows XP Embedded (Sygate Press Release), <http://www.securitypronews.com/news/securitynews/spn-45-20040913SygateIntroducesNextVersionofSygateSecurityAgentforWindowsXPEmbedded.html>

^{xvii} Trend Micro Launches Network VirusWall™ 300 (Trend Micro Press Release), <http://www.trendmicro.com/en/about/news/pr/archive/2004/pr102504.htm>

^{xviii} Supporting Windows XP Embedded-based Devices, (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa460107.aspx>

^{xix} Katherine Enos, Servicing with Windows XP Embedded with Service Pack 2 (Microsoft Corporation, December 2004), 4

^{xx} Remote Management (MSDN Library), <http://msdn2.microsoft.com/en-us/library/ms932519.aspx>

^{xxi} SUS Overview (MSDN Library), <http://msdn2.microsoft.com/en-us/library/aa940929.aspx>

^{xxii} Comprehensive Virus Protection for Microsoft Windows XP Embedded (Computer Associates Website), <http://ca.com/channel/oem/eav.htm>

^{xxiii} The Windows Embedded Advantage-Comparing Windows Embedded with Linux., <http://www.microsoft.com/windows/embedded/comparisoncalculator/notavailable.msp>

^{xxiv} Footprint (MSDN Library), <http://msdn2.microsoft.com/en-us/library/ms912928.aspx>

^{xxv} Mike Hall, I will NEVER use Windows CE or Windows XP Embedded, <http://blogs.msdn.com/mikehall/archive/2004/12/22/331034.aspx>

^{xxvi} Embedded Linux Towers Over Windows XP Embedded, <http://www.linuxworks.com/products/whitepapers/xp-vs-linux.php3>

^{xxvii} Windows Vista for Embedded Systems was released. (DTS Asia), http://www.dst-asia.com/about/05_02_view.asp?id=31&vgoto=&flash_bcate=4&flash_mcate=5&flash_scate=2