Preface

What is our Book about?

This book is for programmers who want to extend Java's capabilities on Windows XP and/or Vista, but aren't sure where to start. One of the drawbacks of Java's portability is that many Java programmers have a rather sketchy knowledge of Windows-specific programming.

Java and Windows Interoperability Recipes describes over 100 ways that Java applications can utilize Windows application software, OS features, and hardware beyond the reach of Java's standard libraries. A variety of Java/Windows programming techniques are explained, including:

- Java's employment of the Win32 API via C, JNI (the Java Native Interface), and J/Invoke.
- Java's utilization of Window's Command Line Interface (CLI) and batch files, accessed through Java's Runtime, ProcessBuilder, and Process classes.
- Java and Windows object-based scripting, centered around the use of VBScript, Windows Script Host (WSH), and Windows Management Instrumentation (WMI)
- Java interoperability with COM, including hosting of ActiveX controls in Swing containers using jacoZoom.

Our examples cover a wide range of common situations where Java has to 'reach out' to the underlying Windows platform. For example, we show how Win32 process management functions can be used to monitor, schedule or terminate processes. We explain how Window's animation effects can be employed by Java applications while copying or moving large files. Hardware devices such as foot-pedals can provide input to Java programs through the use of Window's USB Human Interface Device API. Java windows can be made snazzier with custom window shapes, layered windows, ActiveX controls, and per-pixel transparency. With Window's multimedia capabilities, Java applications can merrily sing and dance.

Our book uses a question-and-answer format, with all the examples tested on Windows XP and Vista, and available online at http://fivedots.coe.psu.ac.th/~ad/winJava/.

The rest of this preface will explain the Java/Windows programming techniques mentioned above in more detail, answer some common questions about the book's aims, and end with a preliminary table of contents.

The Win32 API

The Win32 API is the primary application programming interface for the Windows platform, consisting of over two thousand functions, implemented across multiple dynamic link libraries (DLLs). Win32 was introduced with Windows NT, and has grown and changed with each

successive release of Windows. For instance, Windows Vista added over 160 functions to the API.

As Figure 1 shows, Win32 is a bridge from user-mode applications to the Windows OS kernel. Virtually all Windows applications use the API, either directly, or through frameworks such as MFC, COM, Visual Basic, or .NET; Win32 is a foundational part of Windows.

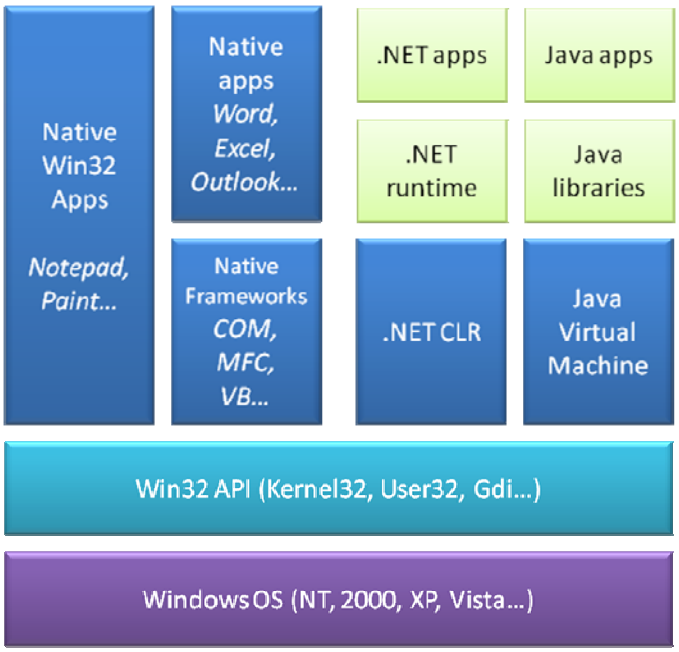

Figure 1. A simplified Windows application stack

The Windows software development kit (Windows SDK) provides the authoritative reference for the Win32 API . The Windows SDK can be downloaded free from MSDN at http://msdn2.microsoft.com/en-us/windowsserver/bb980924.aspx. It includes tools, samples, documentation and C/C++ header files and libraries for linking to the Win32 DLLs.

The API's overall functionality can be divided into eight broad categories:

- Base services (e.g. file systems, devices, processes and threads, error handling)
- Advanced services (e.g. Windows registry, user accounts)
- Graphics Device Interface (GDI) (sending graphical content to monitors, printers, etc)
- User Interface (e.g. manage screen windows, GUI parts of Windows).
- Common Dialog Box library (e.g. dialog boxes, colors, and fonts)
- Common Control library (e.g. status bars, progress bars, toolbars)

- Windows Shell
- Network Services (e.g. Winsock, NetDDE, RPC)

Evolution of the Win32 API

The API has gone through many changes over the years. Knowing something about the evolution of Windows provides some context for the API's idiosyncrasies.

Windows began as a graphical subsystem on top of MS-DOS. The Windows API, introduced with Windows 1.0 in 1985, contained less than 450 functions and interfaced with the 16-bit OS. The Windows OS continued to gain traction, and by the time Windows 3.1 was released in 1992, the 16-bit Windows API had firmly established itself.

Meanwhile, a team of programmers led by Dave Cutler at Microsoft were creating Windows NT ("New Technology"), designed from the ground-up as a 32-bit, multi-tasking, multi-user operating system. It became the foundation for later releases of Windows, including Windows 2000, XP, and Vista.

The primary programming interface for Windows NT was intended to be the 32-bit OS/2 Presentation Manager API, but Microsoft changed its mind. Due to the growing popularity of the 16-bit Windows operating systems, Microsoft decided to make OS/2 and POSIX optional subsystems, and focused on developing the Win32 API – the 32-bit programming interface for Windows programming. To ease the porting effort for existing 16-bit Windows applications, the Win32 API was designed to be backwards compatible with the 16-bit API. This included retrofitting of older function names and stretching of 16-bit parameters into the 32 bit type system.

More recently, Windows has started moving onto 64-bit platforms, leading to a 64-bit version of Windows API, with the pointer data type extended from 32-bits to 64-bits. The result was called the Win64 API for a while, before Microsoft decided to rename it and the Win32 API to simply "Windows API". However, the Win32 API name has stuck, and that's how we refer to the Windows API in this book. The Win32 name also helps to distinguish the API from the OS.

Finding out More about the Win32 API

There's a massive amount of information on the Win32 API, both online, and in books and magazines. This version of the API has been around in various forms since Windows NT and 95, so it's been fully explored, debugged, and its quirks and oddities documented.

Unfortunately, much of the API is explained in the context of general Win32 programming using C or VB, with an emphasis on the basics, such as how to create a window, build GUI components, and draw pictures. Most of that isn't required by us, since Java supports those features already, in a more high-level form. The separation of the API wheat from the chaff is

one of the aims of this book: we point you to the useful parts of the API without burdening you with C/VB programming introductions.

The best place for comprehensive Win32 API information is the Microsoft Developer Network (MSDN), which is overviewed at http://msdn2.microsoft.com/en-us/library/aa383750.aspx. The complete API is online, along with programming articles, and can be searched easily. However, it's sometimes faster to use a search engine (e.g. Google) including the keywords "MSDN", "winapi" or "win32 api".

It's convenient to have API documentation on your own machine. The latest release of the Windows SDK can be downloaded for offline use from:

- Windows SDK for Windows Server 2008; http://www.microsoft.com/downloads/details.aspx?FamilyId=F26B1AA4-741A-433A-9BE5-FA919850BDBF&displaylang=en

The longevity of the Win32 API has meant that it's been interfaced to a great variety of languages and frameworks, such as MFC and .NET. Their class structuring generally makes it harder to understand the underlying Win32 API functions (which were originally aimed at C), and its probably better to search for examples written in C or VB. There are many, many example sites, including:

- codersource.net (http://www.codersource.net/codersource_win32prog.html)
- codeproject.com (http://www.codeproject.com/)
- codeguru.com (http://www.codeguru.com/cpp/w-p/win32/)
- techrepublic.com (http://search.techrepublic.com/)
- TheScarms.com (http://www.thescarms.com/vbasic/)
- Brad's VB-32 Programs and Samples (http://btmtz.mvps.org/)

C/VB articles from Windows programming magazines can be found online at:

- *Dr. Dobb's Journal* (DDJ); http://www.ddj.com/windows/wd.jhtml
- *MS Journal* (MSJ); http://www.microsoft.com/msj/
- *The Windows Developer Journal* (later merged with DDJ); http://www.ddj.com/windows/wd.jhtml

Windows XP/Vista tweak sites are worth a look, since they often use the Win32 API to implement their tricks. A good one is:

- Troubleshooting Windows XP (http://www.kellys-korner-xp.com/xp.htm)

Windows programming newsgroups are a vast resource. Google can search them with queries including:

group:microsoft.public.win32.programmer.*

or group:microsoft.public.vb.winapi.*

Win32 programming textbooks often spend a lot of time talking about basic GUI coding . Even so, they're worth a browse, especially older books that use C or VB rather than C++, C#, MFC, or .NET. The standard older text is:

- Charles Petzold, *Programming Windows*, Microsoft Press, 5th ed, 1998

Two others that focus on the API, not bothering with the GUI, are:

- Jeffrey Richter and Christophe Nasarre, *Windows via C/C++*, Microsoft Press, 5th ed, 2007
- Johnson M Hart, *Windows System Programming*, Addison-Wesley, 3rd ed, 2004

Two good online tutorials on Windows programming are:

- theForger's Win32 API Tutorial, http://www.winprog.net/tutorial/
- Windows API Notes, http://www.cppworld.com/dante/winapinotes/index.htm

JNI

The most common way of coding with Win32 in Java is with C or C++ functions called via JNI (the Java Native Interface). The C/C++ code needs to be compiled and linked into DLLs, which must be added to the Java library path.

The main JNI forum (http://forum.java.sun.com/forum.jspa?forumID=52) is very popular: 22,000 messages, 6,000 topics, 175,000 views, indicating a widespread need for code that goes beyond the bounds of Java.

A look at the JNI forum reveals that most queries are about basic data conversions between Java and C/C++: problems with pointers, structs, unions, and call-by-reference; this reflects JNI's complexity. Intermediate-level questions, such as how to create keyboard and mouse hooks, have been reviewed over 15,000 times, and the proposed answers are difficult to understand.

The two main JNI books are quite old (dating from the end of the 90's), and not particularly advanced due to the large amount of time they need to spend on explaining data conversions.

JNI's low-level aspects have held back the growth of Java-Windows interoperability, and there's a large audience waiting for an enabling technology simpler than JNI. For this reason, most of our Win32 API examples utilize J/Invoke rather than JNI.

J/Invoke

The majority of this book is about utilizing the Win32 API with the help of the J/Invoke library (http://www.jinvoke.com/). J/Invoke relieves the programmer of all the headaches associated with JNI, requiring no C/C++ programming in order to call Windows API functions. J/Invoke achieves this through a declarative, annotation-based programming style. Native functions are annotated and then called. Behind the scenes, J/Invoke manages the task of loading the specified DLL, automatically converting Java arguments to native types, invoking the target function, and translating the result back into Java.

The resulting code is substantially simpler and easy to understand than comparable JNI implementations. This allows much more elaborate examples to be coded by non-expert Windows programmers than is possible with JNI.

The J/Invoke API contains just a handful of classes, annotations, and enumerations. It doesn't introduce a plethora of classes corresponding to native types. Even so, J/Invoke supports primitive types, arrays of primitive types, strings, structures, and callbacks. It works with both Ansi and Unicode character sets, multiple popular calling conventions, and includes utility functions for managing native memory when needed.

J/Invoke's com.jinvoke.win32 package provides helper classes for the most commonly used Win32 DLLs (see Table 1), and it's very simple to call functions and DLLs not covered by those helper classes.

J/Invoke Win32 Helper Class	DLL Name	Functionality
com.jinvoke.win32.Kernel32	KERNEL32.DLL	Base services
com.jinvoke.win32.Gdi32	GDI32.DLL	Graphics device interface
com.jinvoke.win32.User32	USER32.DLL	User interface
com.jinvoke.win32.Advapi32	ADVAPI32.DLL	Crypto API, event logging
com.jinvoke.win32.Shell32	SHELL32.DLL	Windows shell API
com.jinvoke.win32.Winmm	WINMM.DLL	Multimedia
com.jinvoke.win32.WinInet	WININET.DLL	Internet

Table 1. J/Invoke Helper Classes.

The Win32 API contains numerous constants and structures, many of which are defined in J/Invoke's WinConstants class and the com.jinvoke.win32.structs package. It is easy to add new constants and structs if necessary.

The first author of this book, Gayathri Singh, is the lead developer of the J/Invoke API.

J/Invoke's design was inspired by Microsoft .NET's P/Invoke, but is aimed at Java, and also supports native function calls to the Mac OS, Linux, and Solaris. However, we'll only be using its Windows capabilities here.

J/Invoke is available for a 30-day free trial from the J/Invoke web site at http://www.jinvoke.com. There are currently no books on J/Invoke, but plenty of good on-line information at its website. The J/Invoke documentation at http://www.jinvoke.com/docs including a Developer Guide, a Getting Started document and developer scenarios describing how C/C++ can be called from Java on Windows as well as on Unix platforms. The J/Invoke Javadoc (API reference) is available online at http://www.jinvoke.com/www/javadoc/index.html.

There's a user forum at http://www.jinvoke.com/forum, where we can often be found. It's a great place to ask questions, post feedback, and share examples.

Another good alternative to JNI is the open-source Java Native Access (JNA) library, available from https://jna.dev.java.net/. JNA offers a more traditional programmatic approach to native library access, as opposed to J/Invoke's annotations. This design choice increases the complexity of its API, introducing many additional classes and interfaces. However, it enables JNA to be utilized with older JREs while J/Invoke's use of Java annotations requires JDK 5.0 or later.

The CLI and Batch Files

It's surprising how many Java programmers are unaware of the *current* capabilities of Window's command line interface (CLI) and batch files, which have steadily improved since the days of DOS. Windows XP and Vista possess numerous 'power user' CLI utilities, and additional commands can be easily downloaded from Microsoft (e.g. the Windows Server 2003 Resource Kit) and from third-parties (e.g. the GnuWin32 library packages). We describe CLI utilities for controlling the GUI, control panel, the shell, file management, power management, and the network interface. Also, many GUI-based applications (e.g. Windows Media Player) have command line interfaces that makes them very easy to control.

The Windows batch file is often disparaged because of ancient memories of DOS, but XP offers a much better programming experience, including global and local variables, arithmetic expressions, conditional statements (e.g. if-then-else blocks), control flow statements (e.g. for blocks), and procedures. There's support for IO redirection, piping, the manipulation of command line arguments, environment variables, and task scheduling.

Java's interface to the CLI employs its Runtime, Process, and (newer) ProcessBuilder classes. There are several coding 'pitfalls' associated with Runtime and Process, which ProcessBuilder partially resolves. We discuss best coding practice to avoid these, including coding tricks involving XP commands and batch files.

Scriptable Objects

An important part of current Java development is the use of scripting languages, such as JavaScript, Groovy, and Ruby. Unknown to many Java programmers is the availability of several scripting languages on the Windows platform, including the very simple, but powerful and ubiquitous, VBScript.

Much like CLI and batch files, modern-day VBScript is often disparaged because of its poor showing in the 1990's. It was originally a rather lackluster client-side Web scripting language, but Microsoft kept improving it. Our interest focuses on its use with the Windows Script Host (WSH), which provides scripts with direct access to Windows resources represented as objects.

With VBScript and WSH it's very simple to manipulate different areas of the OS, such as the file system, the desktop, the registry, Windows services, printer and disk drives, as well as communicate with many Windows applications (e.g. MS Word and PowerPoint).

VBScript and WSH are ideally suited for the rapid development of scripts that automate Windows tasks, and for the development of small utilities.

In addition, WSH allows the use of COM (Component Object Model) objects. Hundreds of COM objects are supplied with Windows, and hundreds more if you install applications such as Word, Excel, and Visio.

WMI

Windows Management Instrumentation (WMI) is a staple of administrative scripting on Windows – it can access just about every aspect of the OS, including device drivers, system services, and applications. WMI is largely responsible for the surge of interest in administrative scripting over the past few years. Most Java programmers are completely unaware of WMI.

WMI objects are accessible through WSH, and so the same Java techniques we develop for VBScript and WSH can be used with WMI.

WMI is a vast topic, so we concentrate on its use to investigate and control a user's machine. We do not discuss WMI support for remotely accessing other computers on a network.

COM and ActiveX

Java can easily execute most VBScript/WSH scripts, except for ActiveX controls – objects with a GUI interface – which need to be more closely integrated with Java's GUI components. We describe a solution using jacoZoom (http://www.infozoom.de/en_jacoZoom.shtml), which allows you to use ActiveX controls, and other COM objects, with Java. jacoZoom is based on JNI, but hides the very extensive low-level manipulation that JNI carries out.

Why use Java for Windows Programming?

Why should we bother writing Windows programs in Java? Why not call the Win32 API from C, C++, Visual Basic, or .NET?

Let's start with religion! Asking a Java programmer to write in another language is like asking someone to change his religion. This will be met with stiff opposition, and rightly so ☺.

More seriously, the choice of language for a project is often dictated by what the developers are most comfortable with. Many companies have standardized on Java, and the cost of hiring new developers or re-training Java developers is prohibitive.

There are many good reasons why Java is a popular language, including its clean object oriented design, its rich standard libraries (e.g. for GUIs, networking, I/O), numerous high-quality open-source and commercial components, and automatic memory management. Studies have shown Java to be a much more productive language than Visual Basic, C, or C++.

Why 'limit' Java to Windows?

It's prudent to retain a healthy dose of skepticism about directly calling the Win32 API from a Java application. The cross-platform promise of Java ("Write once, run anywhere") has proven its utility. What can be coded purely in Java should be coded that way.

But aren't we killing platform neutrality by tying an application to Windows via the Win32 API? Not if we're careful: it's possible to design cross-platform code that only uses native features when necessary. The result is an application that runs well on multiple platforms, but really shines on Windows – leveraging the Win32 API to do something special.

Not all applications are intended to be cross-platform. You might be writing a disk defragmenter for the Windows NTFS file system, or tools to clean the Windows Registry. It makes sense to closely connect such applications to Windows – platform neutrality isn't relevant in those situations.

Even when a Java package is available for a task, it might still be better to employ the Win32 API instead. For instance, you might have to choose between a non-standard, rather neglected, Java library (e.g. the Java Media Framework) that isn't part of the standard JRE installation, and a DLL which is present in every version of Windows. The Windows choice will reduce your application's footprint, and the library may be faster, easier to use, more reliable, and be better documented.

What this Book isn't

This book isn't an introduction to Java. We assume you're an intermediate-level Java programmer who wants to take advantage of the Windows platform. You're not sure how to start, and want to learn about general techniques and specific solutions.

This book isn't a guide to every aspect of Windows programming; it couldn't be without taking up an entire bookcase. Instead, we aim to discuss core Win32 API capabilities that allow a Java programmer to utilize the main resources on his/her machine, such as the GUI, registry, file system, network interface, attached hardware, audio and video. We include pointers to many other sources of documentation and examples.

Our focus is on Java/Windows programming on a user's machine, often requiring administrative privileges in order to affect the underlying OS or to access hardware. Although we describe ways to access the machine's network interface, and use protocols other than TCP/UDP, we do not discuss more general client-server programming. For example, we don't utilize Java EE, DCOM, or Active Directory Scripting Interface (ADSI) objects.

We don't employ MFC and .NET in this book. Our view is that these frameworks are needlessly complicated for the Windows functionality required by Java programmers. Similar reasoning is applicable to our choice of VBScript rather than Perl as a scripting language, and our use of CLI and batch files instead of the newer PowerShell.

Table of Contents

There are 18 chapters, shared between the two authors. Gayathri is writing 9 chapters, Andrew has 9 chapters.

The book has two sections. The first section introduces the Java/Windows techniques used throughout the rest of the book. Each chapter of the second section looks at a different area of Windows programming using Java.

The following includes a list of possible recipes, along with estimated writing times and the author for each chapter.

8. The Desktop/Shell (Gayathri).

9. Desktop Windows (Gayathri).

10. Hardware Information (Gayathri).

11. Program Installation (Gayathri).

12. Executing Programs (processes) (Gayathri).

13. Process Information (Gayathri)

14. Ghostly, Shapely Windows (Andrew). [Online as chapter 10]

15. Hooking the Keyboard and Mouse (Andrew). [Online as chapter 11]

16. Audio (Andrew).

17. Video (Andrew).

18. Networking (Andrew).

Background and Qualifications

Gayathri Singh is the lead developer of the J/Invoke API (http://www.jinvoke.com/) and a co-founder of ByteBlend, the company behind it. She has over 10 years of software development experience spanning Win32, Java and .NET. Gayathri holds a Masters degree in Urban Planning from Virginia Tech, and an undergraduate degree from Indian Institute of Technology, Kharagpur.

Dr. Andrew Davison is a lecturer and researcher based in the Dept. of Computer Engineering at Prince of Songkla University, Thailand. He has previously authored two popular Java books. In 2005, O'Reilly published *Killer Game Programming in Java*, and Apress brought out *Pro Java 6 3D Game Development* in 2007. The books have accompanying web sites containing their source code and draft chapters at http://fivedots.coe.psu.ac.th/~ad/jg/ and http://fivedots.coe.psu.ac.th/~ad/jg2/.

