IT 09 061
Examensarbete 30 hp
November 2009

Java Platform Adaptation for On-Board Computers

Daniel Gille

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Java Platform Adaptation for On-Board Computers

Daniel Gille

Embedded devices such as Mobile phones, PDA's etc. are getting more and more common in everyday life and professionally. The gain in mobility is lost in hardware power such as CPU speed, memory and battery life to mention a few. To be able to run software on those devices the common strategy has been to use a stripped down version of the software language. One of these languages is Java where Java 2 Micro Edition (J2ME) is a subset of the Java language. The purpose of this document is to provide details on the implementation of the full Java language, namely Java 2 Standard Edition 1.5, on embedded devices. With a full J2SE implementation the possibilities of using existing application or porting applications to Java will be broader.

This research investigates the most common used environments and virtual machines that are used in the market today. The different JVM's are compared to each other and listed with positive and negative aspects regarding adaptation to an embedded device. Interviewing some costumers interested in this Thesis work has provided useful information on the usage and needs of the Java software language.

The target device for this Thesis is the CCP XS embedded computer manufactured by CC Systems. CCP XS typically uses the Windows CE operating system and comes equipped with 256Mb RAM memory. By the use of PERC Virtual Machine, PVM, I have demonstrated that J2SE can execute with acceptable performance on the CCP XS. A demo application written shows that execution of J2SE code, using feature not available in J2E, is possible on the device.

Handledare: Anders Svedberg
Ämnesgranskare: Arnold Pears
Examinator: Anders Jansson
IT 09 061
Tryckt av: Reprocentralen ITC

SAMMANFATTNING

Handhållna enheter såsom micro dator och smarta mobiler blir allt mer vanligt i vår vardag. Hårdvaruprestandan har fått lida till fördel för mobiliteten i dessa produkter. CPU kapacitet och minne är sådant som har bantats ner. Applikationer och program för dessa enheter är typiskt enklare versioner. Programmeringsspråk såsom J2ME som används till bl.a. mobiltelefoner är nerbantade versioner av sina motsvarigheter, Java Standard Edition i detta fallet. Detta dokument kommer att utröna möjligheterna och även visa på en metod för att implementera ett fullt fungerande programmeringsspråk på mobila enheter, nämligen Java 2 Standard Edition. Med fullt stöd från J2SE utökas möjligheterna att använda existerande programvaror och mer avancerade funktioner kan användas.

I denna undersökning jämförs de vanligaste virtuella maskinvaruprogrammen på marknaden idag. De olika JVM'erna listas med dess olika för och nackdelar med hänseende på implementering på en mobil enhet. Intervjuer av ett antal användare har utförts för att få en bra bakgrund och underlag till vad som bör ingå i användarmiljön.

J2SE kommer implementeras på en specifik enhet. Denna enhet utvecklas av CC Systems och är en mobil inbyggd dator som heter CCP XS. Normalt körs CCP XS med Windows CE som operativsystem och använder ett ramminne på 256Mb. Genom implementationen av PERC Virtual Machine, PVM, har jag visat att det är fullt möjligt att använda J2SE på en mobil enhet. Jag har skrivit ett demoprogram som visar hur vissa specifika funktioner i J2SE stöds.

Contents

ACKNOWLEDGEMENTS

I would like to thank CC systems for this opportunity to learn more about embedded systems and the environment that it represents. I have learned much about device drivers which I had not experienced before and I thank CC systems especially for the insight of the work within a company. I especially would like to give my thanks to Anders Svedberg who has been my supervisor at CC systems and his indubitable support both with technical issues and supervision of the progress during the studies and writing of the report. He has been my guide helping me to never get trapped with a problem for a long time

Furthermore I would like to thank Arnold Pears at Uppsala University for taking the role as my subject reviewer with all the work which that includes.

I also like to thank Anna Bengtson and Lesley Carlsson for reviewing and provide useful comments on the disposition and content of this paper.

At last I would like to thank David Humphris at Aonix Europe for being my contact person and support regarding the PERC Ultra JVM which became a large part of my Master's Thesis work.

1 INTRODUCTION

CC Systems is a company that develops and creates hardware and software for intelligent distributed systems. The main targets for their systems are vehicles designed to operate in harsh environments with extreme temperatures, vibrations, and exposure to dust and water. An example is equipment for rock-drilling machines, construction vehicles, rock crushers, military vehicles, forest machines and container spreader systems. The vehicles are equipped with I/O devices and digital micro controllers' which help the person controlling the vehicle in many different ways as well as providing feedback to both the driver of the vehicle and to other vehicles and data stores to enable more extensive analysis.

The different units typically run with software developed in C/C++ or C#. There is great availability of compilers for a large set of platforms and thus the opportunity to use the hardware in an effective way. To shorten development time of the software and to increase quality a simulation framework has been developed by CC systems. This makes it possible to make test runs and debug the system on a standard desktop PC before it is installed on the hardware that will run the particular application. The use of the simulation technology is successful and shortens the time-to-market for programs. However, there are some features present in the languages that may lead to errors. In C and C++ the programmer manages the memory manually and this may lead to memory leaks and memory fragmentation. The current solution to these problems is coding guidelines which are intended to prevent the programmer from using risky programming constructions.

Furthermore, the endianess is implemented differently in different implementations of C/C++ in contrast to Java where this is a part of the specification of the language. These kinds of problems are handled by the use of coding rules which today are manually implemented to ensure a high level of quality in the resulting code. In a more modern language like Java some of these features have been removed or improved. Portability is improved in Java compared to C/C++. Memory management is handled with the help of garbage collection.

1.1 PROBLEM DESCRIPTION

Java being a relatively young and modern programming language in comparison to C, C++ .Net and C#, for example, has a lot of the features that a modern programming language of today have. Those are object handling, memory management and garbage collection to mention a few. Since the development of Java (1995) the language as well as its implementation and adoption from the community has grown large. What are the gains with Java that we can make use of?

With the escalating demand of simpler language development the pressure of an implementation of Java is increasing. Not only to be able to support more programming languages but also to support a growing range of custom made applications from the market. The amplified performance on the embedded systems today helps to speed up the process of

running J2SE on those systems. Is the performance today enough to accomplish the tasks we demand?

Performance has been known to be worse with Java (because it is interpreted by a virtual machine) in comparison to other languages which are compiled to binary code. Can we improve Java VM performance in order to satisfy timing requirements? Is the performance sufficient to handle systems which control critical processes?

Another aspect of converging towards Java is the development tools and support of the language itself. Since Java has been in industry use as a major language for more than 10 years this should not be of any concern though.

The aim of this thesis and the task that CC systems gave me was to research the possibilities to implement Java (J2SE) on one of their embedded computers, namely the CCPilot XS. This is a computer which runs with the Intel Xscale architecture and also has substantially less memory than a normal desktop computer. How does the ARM Intel Xscale IXC 1100 architecture and the stripped down Windows CE operating system stand up to the demands of the programming language? Are there tools and even a JVM supporting java? These are some of the questions raised and the subject of this Master's Thesis.

1.2 OUTLINE

This Master's Thesis disposition is such that it should be read from start to end with an introduction of the problem broken down into smaller pieces in the beginning. Followed by a background of the environment and describing the background of Java in general. Furthermore the study of the various JVM:s is described and discussed. The Master's Thesis ends with a discussion and conclusion part where I deal with the outcome of my work and what I have achieved versus my initial ambition.

The reader acquainted with this field of work may skip the background reading. However the abstract and problem description paragraphs followed by Conclusion are highly recommended.

1.3 OVERVIEW

This thesis will focus on the implementation of Java J2SE 1.5 on one of CC systems on board computers, namely the CCP XS. The CCP XS computer is based around the ARM (Intel XScale, ixc 1100) architecture and has up to 256Mb of memory (SDRAM). This should be sufficient resources to permit an implementation of a JVM that can handle the "full" J2SE. With a full J2SE implementation the possibilities of using existing application or porting applications to Java will be broader.

Figur 1: The CC Pilot XS from different angles

The reason for not implementing the J2ME (micro edition) is that it is believed that the compatibility and performance of the pilot XS is fully capable of handling a full featured J2SE JVM (Java Virtual Machine) or at least that is what this Master's Thesis will try to demonstrate.

Through implementation of Java on the CCP XS, CC systems can target a larger market with the new support, both by easy implementation of existing applications and also by the possibility to develop new applications with the extended choice of language selection. This will all help to keep a high quality standard on developed code and make it easier for the developers to avoid certain pitfalls otherwise addressed by for example coding guidelines.

1.4 SCOPE

The operating system of the CCP XS is Windows CE version 5.0, as that is the most common OS on this machine. Today there is a Linux version, but this is somewhat outdated and is currently scheduled to be reimplemented. Since there is no current demand from customers to run J2SE with Linux I will only investigate the implementation for the WinCE operating system.

There are currently two larger "fractions" of graphical libraries which I can choose to use, the AWT/Swing or SWT, more about the graphical libraries under the Graphical User Interface discussion in section 2.1.3. AWT/Swing is inappropriate for footprint and performance reasons. 256Mb is insufficient for Swing. I will focus on the SWT libraries as a GUI interface.

Regarding the different API:s (see 2.1.3.1) I will try to implement all current available devices on the CCP XS machine. Those are, Ethernet, CAN, Video, Bluethooth, WLAN, GSM/GPRS, GPS, USB, COM and Backlight support. This should be straight forward to implement because of the JNI support which is in the java language specification.

2 JVM STUDY

The intention of the study is to find and define the special needs which an embedded system must pocess to be able to run the fully fledged Java 2 Standard Edition. I also had to scan the market to find out which goals were realistic. I had the opportunity to get in contact with some costumers to CC systems which I interviewed for their requirements.

To locate the most suitable Java virtual machines I first needed to define the different requirements of the system. It is an embedded device but not in the sense of today's mobiles or handheld computers but a more powerful device intended for more advanced deployment like interaction and information support in vehicles. The CCP XS computer runs with the operating system Windows CE 5.0 and it has the capability of 256Mb ram memory. The JVM for this kind of machine that I was set to find should be able to handle Java 2 Standard Editon 1.6 or at least 1.5 and be able to handle graphical libraries.

Today there are two supported GUI libraries supported by J2SE, the inbuilt AWT/Swing library and the SWT library developed and maintained by the Eclipse Foundation. Swing being the more advanced version of the two with more graphical support and usability but also with the setback of a larger footprint may not make it suitable for the striped down embedded device.

2.1 Java

Java in the early stages was contemplated to be developed for embedded devices. A team originally consisting of Patrick Naughton, James Gosling and Mike Sheridan, engineers at Sun Microsystems, considered using C++, but it was rejected for several reasons. Because they were developing an embedded system with limited resources, they decided that C++ demanded too large a footprint and that its complexity led to developer errors. The language's lack of garbage collection meant that programmers had to manually manage system memory, a challenging and error-prone task. The team was also troubled by the language's lack of portable facilities for security, distributed programming, and threading. Finally, they wanted a platform that could be easily ported to all types of devices.

The project first run under the name "The Stealth Project" which quite soon was renamed to "Green Project" 1992. An entirely new language was developed called Oak. The first demonstration of the new language was made in September 1992

In 1994 the language was retargeted towards the World Wide Web and renamed Java. Patrick Naughton wrote a small browser as a prototype first called WebRunner but later renamed to the more known HotJava. Java 1.0a was available for download from the internet in 1994 but the first public release of Java and the HotJava browser was in May 1995. On 9 January 1996, the JavaSoft group was formed by Sun Microsystems in order to develop the technology.

Since J2SE 1.4, the evolution of the Java Language has been governed by the Java Community Process (JCP), which uses Java Specification Requests (JSRs) to propose and specify additions and changes to the Java platform. The language is specified by the Java Language Specification (JLS); changes to the JLS are managed under JSR 901. Below I will introduce the different Java standards that have evolved.

2.1.1 J2ME standard

One of the three main Java specs from JCP/Sun is J2ME. The other two are J2SE and J2EE which are for web applications or servers ('E' stands for enterprise), desktop applications ('S' stands for standard) and portable devices ('M' stands for micro) respectively. J2ME is a subset of the J2SE Java platform and is used for small portable resource-constrained devices such as mobile phones and PDA's. There are some different versions of J2ME which have evolved from various JSRs. Sun Microsystems have not developed any JVM themselves. They rely on third party developers to provide implementations of the standards.

2.1.1.1 J2ME CDC

J2ME CDC is a Java specification for bigger portable devices (RAM of 512 kB or more; typically PDA's). CDC stands for Connected Device Configuration. In short, CDC spec is roughly Java 2 spec without Swing but with many other packages found only in Java 2 specs.

2.1.1.2 J2ME CLDC

J2ME CLDC is a Java specification for smaller portable devices (512 kB RAM or less; typically phones and embedded devices). CLDC stands for Connected Limited Device Configuration. While Sun's documents suggest that CLDC is appropriate for PDA's, it is really far too restrictive (in terms of available GUI API) for PDA's. CDC is much more appropriate.

2.1.1.3 Personal Profile

Personal Profile (PP), replace the PersonalJava platform, which is no longer under active development. Like PersonalJava, they provide the classes necessary for building interactive applications.

2.1.1.4 Foundation

The Foundation profile is based on CDC. However, FP does not provide any API for building interactive applications.

2.1.2 J2SE standard

The J2SE standard is the fully implemented Java platform technology from Sun Microsystems. Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can

then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

2.1.3 Graphical User interface

2.1.3.1 AWT/Swing

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.

2.1.3.2 SWT

SWT is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. SWT is developed and maintained by the Eclipse foundation at eclipse.org/swt.

2.1.4 Application Programming Interface

2.1.4.1 JNI

The Java Native Interface (JNI) is a programming framework that allows Java code running in the Java virtual machine (JVM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly.

The JNI is used to write native methods to handle situations when an application cannot be written entirely in the Java programming language such as when interacting with device drivers written, for example, in C.

2.1.4.2 JNA

Java Native Access provides Java programs easy access to native shared libraries (DLLs on Windows) without using Java Native Interface.

JNA is comparable to .NET Framework P/Invoke. Contrary to JNI, no boilerplate C glue code (wrapper) has to be written to be able to access the native library.

2.2 JVM

A Java Virtual Machine (JVM) is a set of computer software programs and data structures which use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode. This language conceptually represents the instruction set of a stack-oriented, capability architecture.

Java Virtual Machines operate on Java bytecode, which is normally (but not necessarily) generated from Java source code; a JVM can also be used to implement programming languages other than Java. For example, Ada source code can be compiled to Java bytecode, which may then be executed by a JVM. JVMs can also be released by other companies besides Sun (the developer of Java) -- JVMs using the "Java" trademark may be developed by other companies as long as they adhere to the JVM specification published by Sun (and related contractual obligations).

The JVM is a crucial component of the Java Platform. Because JVMs are available for many hardware and software platforms, Java can be both middleware and a platform in its own right — hence the expression "write once, run anywhere." The use of the same bytecode for all platforms allows Java to be described as "compile once, run anywhere", as opposed to "write once, compile anywhere", which describes cross-platform compiled languages. The JVM also enables such unique features as Automated Exception Handling which provides 'root-cause' debugging information for every software error (exception) independent of the source code.

The JVM is distributed along with a set of standard class libraries which implement the Java API (Application Programming Interface). The virtual machine and API have to be consistent with each other and are therefore bundled together as the *Java Runtime Environment*.

2.2.1 Open source JVM

Open source JVM's are Java Virtual Machines which are developed by an independent organization and where the source code is made available free of charge. In order to more easily compare the JVM's with each other the following paragraphs are written as bullet points.

2.2.1.1 KaffeCE

- Supports J2SE Java 1.3.2 (Clean room implementation, AWT but no Swing support).
- Supports the StrongArm and XScale platforms.
- Developed for the following OS: Windows Mobile 5.0, WinCE 5.0

2.2.1.2 Mysaifu

- Project objective is to support J2SE Java 1.5 (No final release yet. Still some bugs).
- Supports the StrongArm and XScale platforms.

- Developed for following OS: Pocket PC 2003, Pocket PC 2003 SE, Windows mobile 5.0/6.0, WinCE 5.0.

2.2.2 Commercial JVM

2.2.2.1 CEE-J/Skelmir

- J2SE (Clean room, awt support, few libraries)
- Ports to several platforms have been made: PPC, x86, MIPS 32, MIPS 64, ST20, ST40, ARM, StrongARM, XScale, SH2, SH3 & SH4, Equator BSP-15, Equator MAP-CA, TI OMAP & DM642, DSPs & VLIWs, SPARC
- Also many OS are supported: Windows 98/NT/2000/XP, GNU Linux, pSOS, WindRiver VxWorks, Apple MacOS/X, Windows CE, PocketPC, Solaris, Chorus, ATI Nucleus RTOS.

2.2.2.2 Jamaica

- J2SE
- JVM bytecode programming
- Uses Judoscript, low level programing

2.2.2.3 Jbed/Jeode

- J2ME
- Platform support: ARM-core, MIPS-core, SH3, SH4, PowerPC, x86, others on request
- Supported OS: Windows CE, Windows Mobile, Linux and Qtopia, Unix, VxWorks, others on request

2.2.2.4 Perc

- J2SE Java 1.5 (AWT/Swing for linux, SWT/PEG+ for WinCE).
- Platforms supported are: Power PC, Intel x86, Coldfire, MIPS, ARM, XScale
- Several OS are supported: VxWorks, VxSim, LunxOS, LynxOS-178, Linux, Windows NT, WinCE, OSE, OSE Softkernel, QNX, Nucleus, ETS, PikeOS.

2.3 Discussions and Conclusions

My task to find a JVM rigid to run the latest J2SE with the latest API and GUI support on the ARM Xscale architecture was somewhat more difficult than I first realized. Since SUN does not provide its own JVM for embedded devices the alternatives were products in the open community as well as the commercial market. The last few years there have been several open source projects aiming to provide some sort of JVM for an embedded device. Most of these projects were aimed at smaller embedded devices such as mobile phones and PDA's.

Since the XS I had as target has a better performance than a normal PDA running Windows CE

I was still eager to find a JVM with support for the full J2SE standard as well as a GUI interface. The open source project Mysaifu JVM which tries to conform to the J2SE standard for Windows CE and also the ARM architecture has come a long way, but still needs some work before it can be relied upon for commercial applications.

PERC Ultra is the Viritual Machine that I chose to work with. It is a commercial VM developed by the company Aonix. Beside the fact that it is a fully functional VM, it is provided with a lot of tools and functionality like adapting and tweaking it with different compilation techniques for extended runtime performance. For example, you can set things like the GC thread frequency, slice and priority, the timer thread frequency and initial heap size. These can all have an effect on the performance and responsiveness of the application. An open source JVM would have been cheaper but I could not find anyone that fully supported the Windows CE platform. The PERC JVM was known to be stable, it had already been tested in several projects and it supported SWT for graphical applications. Since the environment requires a simple GUI for the users a good graphical library was one of the requirements from CC Systems. Swing requires too much memory and footprint usage but would otherwise been a good choice. With the SWT library CC Systems will have a good ground to build their graphical applications.

Furthermore PERC Ultra is continuously developed by Aonix and I will be surprised if they do not conform to the Java 1.6 standard in a near future (Java 1.6 gives an approximately speed gain of 15%). If extended usability or some special support is needed Aonix offers to port the JVM after costumer specifications with a professional service contract. With this in mind I think that the PERC JVM offers a stable solution to integrate “full” Java support on an embedded device.

3 JAVA DEMO APPLICATION

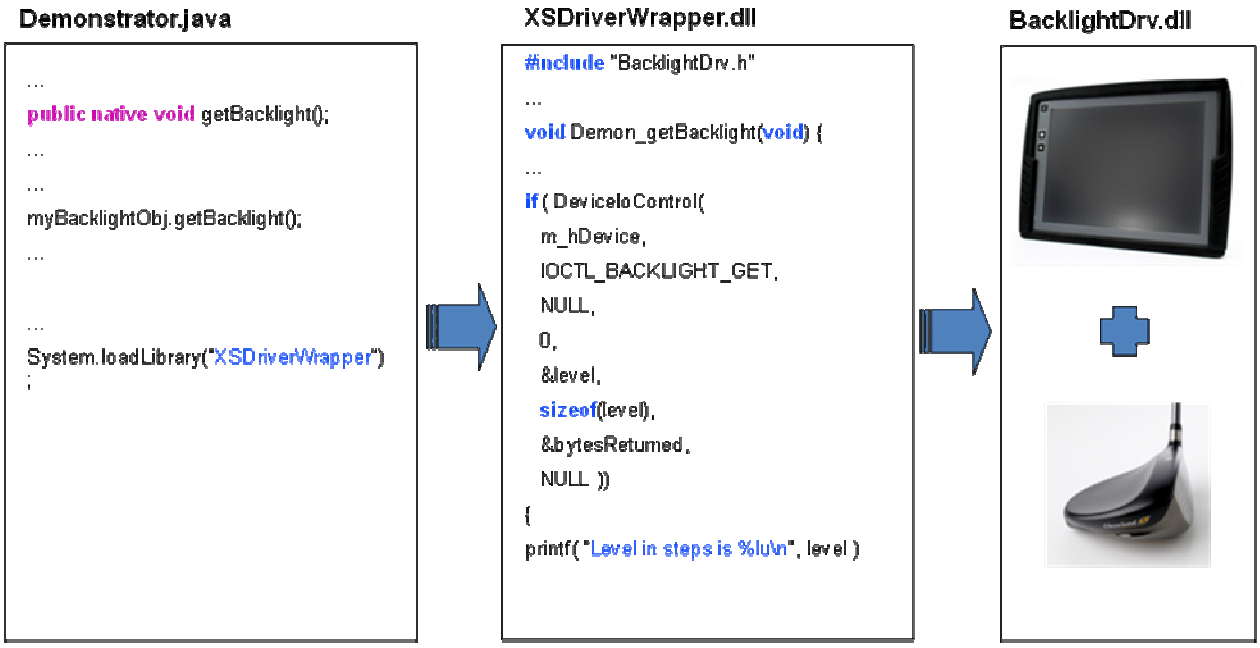
3.1 Purpose

I have provided a simple GUI interface to show that I was able to implement J2SE as a development language on the CCP XS machine. The GUI uses the SWT library for graphical representation. The demo application, demonstrator, provides a visualisation of sliders with which you can change the backlight of the screen. While moving the slider, using either the touch screen or the arrow buttons on a keyboard plugged in to one of the USB connectors, you reduce alternative increase the backlight of the screen. This showing the usage of the native interface providing a tool for interaction with existing device drivers.

It also shows the possibilities for graphical interfaces using SWT on the machine. I show that with my wrapping technique it is possible to wrap already existing native code. For CC Systems this is very useful when they will access, for their Java code, already written device drivers for different hardware on the CCP XS.

3.2 Implementation

It was a necessary to use a device driver in my demo program because the target XS device uses a lot of different hardware and the usability to be able to call legacy device drivers written in native code was essential. This was done by using the **P**erc **N**ative **I**nterface (PNI). PNI gains its speed from being tightly coupled to the PERC VM. While JNI is portable across VMs, and across different versions of the same VM. PNI's one benefit is that it does run faster than JNI. How much faster depends of course on the platform and the usage. Typically, PNI will run about three times faster than JNI.

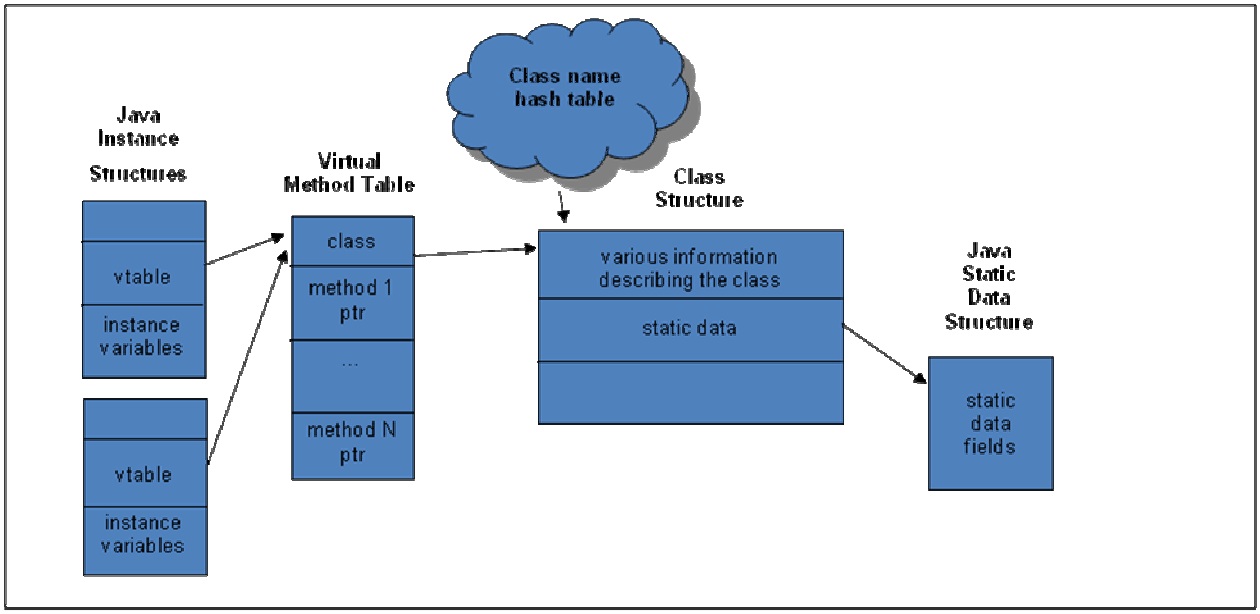


Figur 2: Shows the wrapper as the "glue" between the application and the device driver

By the implementation of a wrapper I am able to call the driver code without the need of changing it. This way there is no need to rewrite already existing drivers to the different hardware included in the CCP XS. The application class itself only needs to declare the wrapper functions and do an inbuilt system call to loadLibrary to load the wrapper.

3.3 Architecture

The architecture I use provides a swift way of accessing user data from a function. It places a reference to an objects instance variables on a virtual stack. This way I can create an object in my java class and use the functions initiated with its type. This is shown in the example in Figur 2: Shows the wrapper as the "glue" between the application and the device driver where myBacklightObj is the created object type. Then in the wrapper I only have to update the data value that is referenced by the object. This is how I transfer arguments to functions within the wrapper and it works in both directions.



Figur 3: Object reference chart

3.4 Discussions and Conclusions

The first step was to implement the **P**ERC **V**irtual **M**achine, PVM, on the CCPilot XS device. After some setup using the "getting started" guide from Aonix including "getting search path correct" I managed to get a simple "hello world" from the device using above mentioned PVM. There was a lot of work behind those simple words, having everything configured correct, but it was something that I would benefit from later on.

By using the slider in my demonstrator I show that the graphical library SWT is implemented and working as intended. Yet more important is that the backlight driver is called from the demo program. This shows not only that I can use and control the different hardware on the device but that I can pass and receive values by sending arguments to and from the device driver. In my case I have to send the new backlight value as soon as the slider is adjusted. To be able to test that I can receive values I implemented the functionality to update the slider whenever the increase/decrease buttons on the XS device are pressed. This is done by constantly polling the driver getting its value every 10ms, this is done in a separate thread. Whenever the value is changed the slider will update its position to correspond to the new value.

4 RELATED WORK

As I mentioned before there is a lot of different project which tries to implement Java in any format on an embedded device. If I only look within the company (CC Systems) there have been two larger studies regarding this issue in the last few years. 4 years ago (2004) there was a Master's Thesis work, "*Java in Distributed Control Systems*", by Johan Hedlund that researched this field of study and concluded that there were no recent JVM capable of handling the hardware capacity of the XS device at that time. An in-house study has been made during 2006 researching the possibility to adapt java on the XS environment without success. Now, in 2008, when I have done my study I have realized that there is a great demand for J2SE capability on a lot of embedded systems/devices. It is not until the last few years JVM's have been available specifically for targeted platforms with J2SE specifications.

5 FUTURE DEVELOPMENT

The purpose of this thesis and the task that CC systems gave me were to research the possibilities to implement Java (J2SE) on one of their embedded computers. I developed a demo application showing that this is possible using the Perc JVM.

In my demo application I only implemented one device driver. To fully use java on the XS all the other hardware interfaces also needs to be integrated. This is however simply done by following the example I have set out when implementing the backlight device. A new wrapper or extending the existing to include all device drivers will suffice. There is different gain from writing a new wrapper for each interface versus extending the one existing. With one wrapper you can reuse code and you will only have to maintain and update one file when needed. A new wrapper for every new device driver will give you easier overview of the code and the means to only use the wrapper files needed for that specific device.

Being able to use the Swing GUI in the future will acquire a device with more power when it comes to memory. Today there is no Virtual Machine supporting Swing for the Windows CE version. Embedded devices will have to limit themselves to the SWT libraries when it comes to graphics. This is however not a major issue because the embedded devices usually comes with smaller screens and the SWT libraries are powerful enough, providing you with some useful tools when it comes to graphical user interfaces.

6 CONCLUSIONS

Java is a continuously growing programming language constantly in change with new libraries, new possibilities and increased functionality. The ongoing development from companies and open source projects make it an up to date modern language. Java has been a language mostly concentrated on the personal computers. It is only in the last few years that it has changed towards the initial goal for Java, a language especially designed for embedded devices. The support and develompent for J2SE on the mobile ebedded devices has become a bit outdated in comparison to the tools and JVM's designed and created for personal computers. Though, the market today is becoming more and more focused on the mobile devices.

By integrating the Java platform on the CCP XS machine I have shown that it is possible to run J2SE on an embedded device as long as certain criterias are fulfilled. The memory consumption of the systme was not as problematic as first anticipated. The systems peak memory usage lies well within the boundaries of the target platform. Furthermore I have found that there is no support for all existing platforms and there is still work to do regarding implementation of graphical libraries like Swing etc. As long as the hardware meets the specifics and the systems run Windows CE there should be no complications integrating J2SE. With the use of wrapper code the integration is simplified and a lot of code (drivers) on the system can be reused.

The more complex embedded systmes such as controller systems in the industy and advanced vehicles systems will benefit from a broader set of tools when it comes to programming. With better tools it is easier to implement the device to its environment. A better graphical interface can be developed to increase the usability and interaction with users. This is because of the extended graphical libraries swing etc. brings.

REFERENCES

Literature

Eric Giguère, John Wiley & Sons, Inc, 2000, *Java 2 Micro Edition*

Daniel A.Norton, Addison Wesley, May 1992, *Writing Windows Device Drivers*

Mary Campione, Kathy Walrath, Addison Wesley, 2000, *The Java Tutorial Second Edition*

Paper

A NewMonics White Paper, 2007, *Differentiated Network Services and The Network Element Platform Lifecycle*

Aonix White Paper, 2008, *J2SE vs J2ME for Embedded Systems*

Johan Hedlund, 2004, *Java in Distributed Control Systems*

Internet

NSIcom, 2008-03-07 http://nsicom.com

JudoScript.com, 2008-03-07 http://www.judoscript.com/articles/jamaica.html

Mysaifu JVM, 2008-03-07 http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html

Aonix, 2008-03-07 http://www.aonix.com/perc.html

Kaffe.org, 2008-03-07 http://www.kaffe.org

Wikipedia, 2008-05-12 http://www.wikipedia.org/

Sun Developer Network, 2008-05-12 http://java.sun.com/

Appendix A: JVM study

A study of today's Java Virtual Machine

Open Source

Mysaifu

http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html

Java Version:
J2SE Java 1.5 (No final release yet, got some bugs)

Supported platforms:
StrongARM, XScale

OS:
Pocket PC 2003, Pocket PC 2003 SE, Windows Mobile 5.0/6.0, WinCE 5.0

Tools:
No tools available

Latest Release:
29 feb 2008 (0.3.6)

KaffeCE

http://www.kaffe.org
http://www.rainer-keuchel.de/wince/kaffe.html

Java Version:
J2SE Java 1.3.2 (Clean room implementation, AWT but no Swing support)

Supported platforms:
StrongARM, XScale

OS:
Windows Mobile 5.0, WinCE 5.0

Tools:
No tools available

Latest Release:
KaffeCE had its latest release 26 apr 2002
(Kaffe for Linux had its latest release 26 feb 2008)

Commercial products

CEE-J (SKELMIR)
http://www.skelmir.com/products/ceej.html
Footprint: Feature set selection at compile-time.
Customizable to be as small as 100k
ROM: 360kb – 760kb(with AWT)

Java Version:
J2SE (Clean room, awt support, few libraries)

Supported platforms:
PPC, x86, MIPS 32, MIPS 64, ST20, ST40, ARM, StrongARM, XScale, SH2, SH3 & SH4, Equator BSP-15, Equator MAP-CA, TI OMAP & DM642, DSPs & VLIWs, SPARC

OS:
Windows 98/NT/2000/XP, GNU Linux, pSOS, WindRiver VxWorks, Apple MacOS/X, Windows CE, PocketPC, Solaris, Chorus, ATI Nucleus RTOS.

Latest Release:

J9 IBM
http://www128.ibm.com/developerworks/websphere/zones/wireless/weme_eval_runtimes.html?S_CMP=rnav

Java Version: (J2ME)
CDC 1.0/1.1 Personal Profile/Foundation Profile
CLDC 1.1. MIDP 2.0

Supported platforms:
ARM, MIPS, XScale, SH4, PowerPC, x86.

OS:
Windows CE 5.0, Windows Mobile 5.0, Linux, Mobilinux, MontaVista Linux, QNX 6.2.1, Palm OS Garnet

Latest Release:

Jamaica JVM
http://www.judoscript.com/articles/jamaica.html

Java Version:
J2SE, Bytecode programming, judoscript, low level programing

Jbed/Jeode (Esmertec, J2ME)
http://www.esmertec.com/solutions/mobile_multimedia/jbed_advanced/configurations_profiles.shtml

Java Version:
CDC 1.0 (JSR 36)
Personal Profile 1.0 (JSR 62)
Personal Basis Profile 1.0 (JSR 129)
Foundation Profile 1.0 (JSR 46)
RMI Optional Package 1.0 (JSR 66)
JDBC Optional Package 1.0 (JSR 169)

Supported platforms:
ARM-core, MIPS-core, SH3, SH4, PowerPC, x86, others on request

OS:
Windows CE, Windows Mobile, Linux and Qtopia, Unix, VxWorks, others on request

Latest Release:

Perc
http://www.aonix.com/perc.html

Java Version:
J2SE Java 1.5 (AWT for linux, SWT/PEG+ for WinCE)

Supported platforms:
Power PC, Intel x86, Coldfire, MIPS, ARM, XScale.

OS:
VxWorks, VxSim, LunxOS, LynxOS-178, Linux, Windows NT, WinCE, OSE, OSE Softkernel, QNX, Nucleus, ETS, PikeOS.

Latest Release:
2008

Appendix B

Userguide, Environment Setup, Example deployment Of PERC Ultra on CCP XS

(Java Platform Adaptation for On-Board Computers)

CONTENT

1 FILES

1.1 PERC

1.1.1 perc-windows-5.0.0699.exe — **Setup file for windows host**

1.1.2 perc-plugins-3.1.0.zip — ***Eclipse plugin***

1.2 Documentation

1.2.1 start.pdf — **PERC Getting started guide**

1.2.2 manual.pdf — ***PERC Manual***

1.2.3 TN-RD-100-JV-PERC-V42.pdf — **Technical note for optimization**

1.2.4 PERC evaluation downloads and evaluation key.msg — **Email from Aonix (evaluation information)**

1.3 Demonstrator

1.3.1 Demon.java — **Source code of the demonstrator**

1.3.2 swt.jar — **SWT library from eclipse.org/swt (WinCE ver)**

1.3.3 swt-win32-3347.dll — **Additional swt driver**

1.3.4 XSDriverWrapper.dll — **Wrapper for the BacklightDrv.dll**

1.3.5 ReadMe.txt — **Command list on how to make a java archive file**

1.3.6 headers.txt — **Compile information used when creating the java archive file**

2 INSTALLATION

2.1 Windows

Required OS, Windows 2000 (SP4 or later) or Windows XP (SP2 or later).
Run the *perc-windows-5.0.0699.exe* file and the setup progress will initiate. The default installation directory is C:\perc5.0; however, during installation you have opportunity to change the directory path. The classpath and other environmental variables will automatically be set by the installation process. For additional information please refer to the *start.pdf* file

2.2 Eclipse

First make sure that eclipse (ver 3.0.x or later) is installed accordingly (eclipse.org). Then unpack the *perc-plugins-3.1.0.zip* file into your eclipse/plugin folder.
Eclipse will automatically detect the new plugin when it is launched.

See *manual.pdf* appendix E for remote debugging with Eclipse, appendix F for usage of the eclipse plugin.

2.3 CCP XS

Please see *manual.pdf* appendix G for information on how to set up your environment on the CCPilot XS (Windows CE).

Additional notes:
You run a target file on the XS by entering the following command on the prompt;
Change to desired directory e.g.
\>cd PERC\Demon\

run the execution command e.g.
\PERC\Demon> "\PERC\bin\pvm.exe" –c
\PERC\lib\perclib.jar;perctool.jar;gcp.jar;\PERC\swtWinCE\swt.jar;\PERC\Demon\Demon.jar Demon

3 EXAMPLE – STEP BY STEP

This is a summarized tutorial for creating and deploying an example Java program using both the SWT Graphical User Interface as well as the Perc Native Interface. I have included some sample files, listed in paragraph 1 which will make it easier to follow this tutorial.

3.1 JAVA class

Make sure that your system is setup with eclipse (eclipse.org) to run with perc pvm (more info in paragraph 2.2). Create a new java application project and the test file *Demon.java* and write your code.

You need to declare your native functions e.g.

```
public native void getBacklight();
```

And you need to make a system call to load the native library *XSDriverWrapper.cpp*

```
System.loadLibrary( "XSDriverWrapper" );
```

If eclipse is set up accordingly you should have the "JRE System Library [PERC]" included into your project by default. To add the swt gui functionality you need to add swt.jar. Right click the project folder, under Build Path choose Add External Archives and find your downloaded swt.jar (eclipse.org/swt).

3.2 Wrapper driver (dll)

Create a new project in MVS 5.0 or similarly. Use the installed template "MFC Smart Device DLL" under Visual C++/Smart Device. During the setup wizard, change to the preinstalled CCPXS platform SDK (Pocket PC 2003) to be able to deploy your wrapper on the CCPilot XS.

In the command prompt, go to your project (java) directory. Use jikes to compile to the .class file and then use perch on that file to generate a skeleton for the wrapper. Copy the skeleton to the mvs project and replace the default files there. Fill out the skeleton with the calls you would like to make to your desired driver (see the drivers documentation) Save, compile and your XSDriverWrapper.dll is ready.

In addition you can use eclipse to develop the dll (C++) as well but I have not done any "how to" on that.

3.3 Deploying on the CCP XS

The easiest way to deploy and run on the XS is to create an archive file where the different libraries are linked in.

Enter the following on the windows promt to create a jar file which has the swt library as well as the XSDriverWrapper linked in to it.

>jdtc -cp .\swt.jar Demon.java

>jar cmf headers.txt Demon.jar Demon.class Demon$1.class Demon$2.class Demon$3.class Demon$4.class swt.jar swt-win32-3347.dll XSDriverWrapper.dll

Copy the Demon.jar and XSDriverWrapper.dll to the CCP XS. If your wrapper does any calls to the BacklightDrv.dll you need to make sure that like the XSDriverWrapper.dll, it is also installed under the /Windows directory on the CCP XS. The Demon.jar could be copied to any directory (\PERC\Demon in my example). Then execute the file with the following command.

>"\PERC\bin\pvm.exe" -c \PERC\lib\perclib.jar;perctool.jar;gcp.jar;\PERC\swtWinCE\swt.jar;\PERC\Demon\Demon.jar Demon

OBS! run on winCE (swt.jar swt-win32-3347.dll are compiled for winCE)

Appendix C

Demon.java

```
/**
 * Demonstrator of the Java J2SE functionality(graphical and native)
 * on the XS(with WinCE)using PERC JVM. As a part of a Master Thesis
 * work by Daniel Gille at CC Systems 2008.
 */

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.layout.*;

/**
 * @author Daniel Gille
 *
 */
public class Demon implements Runnable {

		/*
		 * Backlight function
		 */
		int backlight, resolution, bLight;
		static Slider backlightSlider;
		static GridData slider;

		Demon()
		{
				resolution = 0;
				backlight = 0;
		}

		/*
		 * Thread constantly polling the backlight value
		 * updating the backlight slider when changed.
		 * @see java.lang.Runnable#run()
		 */
		public void run()
		{
				while(true){
						this.getBacklight();

						if (this.backlight != bLight)
						{
								bLight = this.backlight;

								Display.getDefault().asyncExec(new Runnable() {
										public void run()
										{

		backlightSlider.setSelection(bLight);
										}
								});
						}

						try {
								Thread.sleep(10);
						}
						catch (InterruptedException e) {
								e.printStackTrace();
						}
				}
		} // End run
```

```
/*
 * Native functions implemented in XSDriverWrapper
 */
public native void getBacklightResolution();
public native void getBacklight();
public native void setBacklight();

/**
 * @param args
 */
public static void main(String[] args) {

        // Variables
        Group infoGroup, deviceGroup, backlight, gps, can;

        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setText("CC Pilot XS Demonstrator");
        RowLayout rowLayout = new RowLayout();
        rowLayout.type = SWT.VERTICAL;
        shell.setLayout(rowLayout);

        infoGroup = new Group(shell, SWT.NONE);
        infoGroup.setText("Info");
        infoGroup.setBackground(new Color(display, 255,255,240));
        infoGroup.setLayout(rowLayout);
        infoGroup.setLayoutData(new RowData(780, 45));
//      layoutComposite = new Composite(layoutGroup, SWT.BORDER);
//      layoutComposite.setLayoutData(new GridData(GridData.FILL_BOTH));

        /*
         * Info label
         */
        Label label1 = new Label(infoGroup, SWT.NONE);
        label1.setText ("Demonstrator of Java (J2SE) functionality on " +
                        "the XS Pilot with WinCE OS using PERC JVM. " +
                        "Implementation of the SWT GUI as well as \nusing JNI for " +
                                "device functionality is demonstrated. " +
        "As a part of a Master Thesis work by Daniel Gille at CC Systems 2008. \n\n");
        label1.setBackground(new Color(display, 255,255,240));

        deviceGroup = new Group(shell, SWT.NONE);
        deviceGroup.setText("CC Pilot XS devices control");

        RowLayout deviceLayout = new RowLayout();
        deviceLayout.marginTop = 10;
        deviceGroup.setLayout(deviceLayout);
        deviceGroup.setLayoutData(new RowData(780, 450));

        /*
         * Backlight device
         */
        backlight = new Group(deviceGroup, SWT.NONE);
        backlight.setText("Backlight");
        GridLayout gridLayout = new GridLayout(4, false);
        gridLayout.marginTop = 8;
        backlight.setLayout(gridLayout);

        Composite blComp = new Composite(backlight, SWT.BORDER);
        blComp.setLayout(new GridLayout(2, false));
        //blComp.setLayoutData(new GridData(GridData.FILL_BOTH));

        Label label2 = new Label(blComp, SWT.NULL);
        label2.setText ("Low");
        Label label3 = new Label(blComp, SWT.NULL);
        label3.setText ("High");
        GridData gridData = new GridData();
        gridData.horizontalAlignment = GridData.END;
```

```
label3.setLayoutData(gridData);

/*
 * Create Demon object containing lightlevel
 * and resolution data.
 * Get init values from BacklightDrv.dll driver
 * using XSDriverWrapper
 */
final Demon myBacklightObj = new Demon();
myBacklightObj.getBacklightResolution();
myBacklightObj.getBacklight();

/*
 * Slider
 */
backlightSlider = new Slider(blComp, SWT.NULL);
slider = new GridData();
slider.horizontalAlignment = GridData.FILL;
slider.horizontalSpan = 2;
backlightSlider.setLayoutData(slider);
backlightSlider.setMinimum(0);
backlightSlider.setMaximum(myBacklightObj.resolution);
backlightSlider.setSelection(myBacklightObj.backlight);

backlightSlider.addListener(SWT.Selection, new Listener() {
                        public void handleEvent(Event event)
                        {
myBacklightObj.backlight = backlightSlider.getSelection();
                                                myBacklightObj.setBacklight();
                        }
});


/*
 * Listen to changes of Object.backlight and updates slider
 */
new Thread(myBacklightObj).start();


/*
 * GPS device
 */
gps = new Group(deviceGroup, SWT.NONE);
gps.setText("GPS");
gps.setLayout(gridLayout);

Composite gpsComp = new Composite(gps, SWT.BORDER);
gpsComp.setLayout(new GridLayout(1, true));
gpsComp.setLayoutData(new GridData(GridData.FILL_BOTH));

Label label4 = new Label(gpsComp, SWT.NULL);
label4.setText ("Get GPS position");

Button gpsButton = new Button (gpsComp, SWT.PUSH);
gpsButton.setText ("&Get");

final Text text = new Text(gpsComp, SWT.SHADOW_IN);
 text.setBounds(0, 20, 150, 25);

gpsButton.addListener(SWT.Selection, new Listener() {
                        public void handleEvent(Event event) {
                                                if (event.button == 0) {
                                                                        text.setText("x:14.2 y:72.5");
                                                } else {
                                                                        text.setText("x:0 y:0");
                                                }
                        }
});
```

```
                        /*
                         * CAN device
                         */
                        can = new Group(deviceGroup, SWT.NONE);
                        can.setText("CAN");
                        can.setLayout(gridLayout);

                        Composite canComp = new Composite(can, SWT.BORDER);
                        canComp.setLayout(new GridLayout(1, true));
                        canComp.setLayoutData(new GridData(GridData.FILL_BOTH));

                        Label label5 = new Label(canComp, SWT.NULL);
                        label5.setText ("Test CAN Bus");

                        Button canButton = new Button (canComp, SWT.PUSH);
                        canButton.setText ("Test");

                        final Text text2 = new Text(canComp, SWT.SHADOW_IN);
                        text2.setBounds(0, 20, 150, 25);

                        canButton.addListener(SWT.Selection, new Listener() {
                                public void handleEvent(Event event) {
                                        if (event.button == 0) {
                                                text2.setText("Testing...");
                                        } else {
                                                text2.setText("Failure");
                                        }
                                }
                        });

                        shell.pack ();
                        shell.setSize(800, 570);
                        shell.open ();

                        /*
                         * loop updating the UI (SWT)
                         */
                        while (!shell.isDisposed ()) {
                                if (!display.readAndDispatch ())
                                        display.sleep ();
                        }
                        display.dispose ();

        } // End main

        /*
         * Load native PNI wrapper
         * Gives access to getBacklightResolution, getBacklight and setBacklight
         */
        static
        {
                        System.loadLibrary( "XSDriverWrapper" );
        }

} // End Demon class
```

XSDriverWrapper.def

```
; XSDriverWrapper.def : Declares the module parameters for the DLL.

LIBRARY      "XSDriverWrapper"

EXPORTS
   ; Explicit exports can go here
                        Demon_init
```

XSDriverWrapper.h

```
/* Total generated by PERCH 3.0 at Thu May 11 14:41:40 MDT 2000 */
#ifdef __cplusplus
extern "C"
{
#endif

#ifndef __XSDriverWrapper_H__
#define __XSDriverWrapper_H__

#include <perc.h>


/* Begin File Scope Edits after this line */
/* End File Scope Edits before this line */

struct Backlight_object {
                        /**/JInt backlight;
                        /**/JInt resolution;
};

extern void Demon_init(void);
#endif

#ifdef __cplusplus
}
#endif

/*
// XSDriverWrapper.h : main header file for the XSDriverWrapper DLL
//

#pragma once

#ifndef __AFXWIN_H__
                        #error "include 'stdafx.h' before including this file for PCH"
#endif

#include "resource.h"

// CXSDriverWrapperApp
// See XSDriverWrapper.cpp for the implementation of this class
//

class CXSDriverWrapperApp : public CWinApp
{
public:
                        CXSDriverWrapperApp();

// Overrides
public:
                        virtual BOOL InitInstance();

                        DECLARE_MESSAGE_MAP()
};
*/
```

XSDriverWrapper.cpp

```
/*
 * XSDriverWrapper.cpp
 * by Daniel Gille 2008
 */
#include "stdafx.h"
#include "XSDriverWrapper.h"
#include "Backlight.h"

/* Begin File Scope Edits after this line */
/* End File Scope Edits before this line */

/* Native method for Demon.getBacklightResolution()V*/
void Demon_getBacklightResolution(void) {
enum pStack { pTHIS};
/* Begin edits after this line for getBacklightResolution()V*/
/*
 *
 *
 */
                struct Backlight_object *thisptr;
                unsigned long bLightR, bytesReturned, resolution;

                thisptr = (struct Backlight_object *) GetLocalPtr(pTHIS);
                /*
                 * get resolution value from BacklightDrv.dll
                 */
                HANDLE m_hDevice;

                                if ((m_hDevice =
                                CreateFile(
                                                TEXT("BKL1:"),
                                                GENERIC_READ | GENERIC_WRITE,
                                                FILE_SHARE_READ | FILE_SHARE_WRITE,
                                                NULL,
                                                OPEN_EXISTING,
                                                0,
                                                NULL )) != INVALID_HANDLE_VALUE )
                                {
//              unsigned long bytesReturned, resolution;

                if ( DeviceIoControl(
                                                m_hDevice,
                                                IOCTL_BACKLIGHT_GET_RESOLUTION,
                                                NULL,
                                                0,
                                                &resolution,
                                                sizeof(resolution),
                                                &bytesReturned,
                                                NULL ))
                {
                                printf( "Resolution in steps is %lu\n", resolution );
                }
                else
                {
                                printf(
                                "!!ERROR, %lu when calling \"DeviceIoControl\"\n",
                                GetLastError() );
                }
}
else
{
                printf(
                                "!!ERROR, %lu when calling \"CreateFile\"\n",
                                GetLastError() );
}
```

```
                bLightR = resolution;

                // bLightR = GetHeapNonPtr(thisptr, ->resolution, struct Backlight_object *);

                SetHeapNonPtr(thisptr, ->resolution, struct Backlight_object *, (JInt) bLightR);
                return;
/* End edits before this line for getBacklightResolution()V*/
}


/* Native method for Demon.getBacklight()V*/
void Demon_getBacklight(void) {
enum pStack { pTHIS};
/* Begin edits after this line for getBacklight()V*/
/*
 *
 *
 */
                struct Backlight_object *thisptr;
                unsigned long bLight, bytesReturned, level;

                thisptr = (struct Backlight_object *) GetLocalPtr(pTHIS);
                /*
                 * get backlight value from BacklightDrv.dll
                 */
                HANDLE m_hDevice;

                if ((m_hDevice =
                                CreateFile(
                                                TEXT("BKL1:"),
                                                GENERIC_READ | GENERIC_WRITE,
                                                FILE_SHARE_READ | FILE_SHARE_WRITE,
                                                NULL,
                                                OPEN_EXISTING,
                                                0,
                                                NULL )) != INVALID_HANDLE_VALUE )
                {

//              unsigned long bytesReturned, level;

                if ( DeviceIoControl(
                                                m_hDevice,
                                                IOCTL_BACKLIGHT_GET,
                                                NULL,
                                                0,
                                                &level,
                                                sizeof(level),
                                                &bytesReturned,
                                                NULL ))
                {
//                              printf( "Level in steps is %lu\n", level );
                }
                else
                {
                                printf(
                                                "!!ERROR, %lu when calling \"DeviceIoControl\"\n",
                                                GetLastError() );
                }
                }
                else
                {
                printf(
                                "!!ERROR, %lu when calling \"CreateFile\"\n",
                                GetLastError() );
                }

                bLight = level;
```

```
                    // bLight = GetHeapNonPtr(thisptr, ->backlight, struct Backlight_object *);

                    SetHeapNonPtr(thisptr, ->backlight, struct Backlight_object *, (JInt) bLight);
                    return;

/* End edits before this line for getBacklight()V*/
}


/* Native method for Demon.setBacklight()V*/
void Demon_setBacklight(void) {
enum pStack { pTHIS};
/* Begin edits after this line for setBacklight()V*/
/*
 *
 *
 */
                    struct Backlight_object *thisptr;
                    unsigned long bLight, bytesReturned, level;

                    thisptr = (struct Backlight_object *) GetLocalPtr(pTHIS);
                    /*
                     * set backlight value through BacklightDrv.dll
                     */

                     bLight = GetHeapNonPtr(thisptr, ->backlight, struct Backlight_object *);

                    level = bLight;

                    HANDLE m_hDevice;

                    if ((m_hDevice =
                                        CreateFile(
                                                            TEXT("BKL1:"),
                                                            GENERIC_READ | GENERIC_WRITE,
                                                            FILE_SHARE_READ | FILE_SHARE_WRITE,
                                                            NULL,
                                                            OPEN_EXISTING,
                                                            0,
                                                            NULL )) != INVALID_HANDLE_VALUE )
{
//                  unsigned long bytesReturned, level = 10;

                    if ( !DeviceIoControl(
                                                            m_hDevice,
                                                            IOCTL_BACKLIGHT_SET,
                                                            &level,
                                                            sizeof(level),
                                                            NULL,
                                                            0,
                                                            &bytesReturned,
                                                            NULL ))
                    {
                                        printf(
                                                            "!!ERROR, %lu when calling \"DeviceIoControl\"\n",
                                                            GetLastError() );
                    }
}
else
{
                    printf(
                                        "!!ERROR, %lu when calling \"CreateFile\"\n",
                                        GetLastError() );
}

//                  bLight = level;

//  bLight = GetHeapNonPtr(thisptr, ->backlight, struct Backlight_object *);
```

```
                        //SetHeapNonPtr(thisptr, ->backlight, struct Backlight_object *, (JInt) bLight);
                        return;

/* End edits before this line for getBacklight()V*/
}


void Demon_init(void) {
registerNativeMethod("Demon", "getBacklightResolution()V", Demon_getBacklightResolution);
registerNativeMethod("Demon", "getBacklight()V", Demon_getBacklight);
registerNativeMethod("Demon", "setBacklight()V", Demon_setBacklight);

}
#if 0 /* Lost code follows this line (Do not delete this line) */
#endif /* Lost code precedes this line (Do not delete this line) */



/*
// XSDriverWrapper.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "XSDriverWrapper.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif



// CXSDriverWrapperApp

BEGIN_MESSAGE_MAP(CXSDriverWrapperApp, CWinApp)
END_MESSAGE_MAP()


// CXSDriverWrapperApp construction

CXSDriverWrapperApp::CXSDriverWrapperApp()
{
                        // TODO: add construction code here,
                        // Place all significant initialization in InitInstance
}


// The one and only CXSDriverWrapperApp object

CXSDriverWrapperApp theApp;


// CXSDriverWrapperApp initialization

BOOL CXSDriverWrapperApp::InitInstance()
{
                        CWinApp::InitInstance();

                        return TRUE;
}
*/
```