1 APT User Guide

1.1 Hussein Shafie,

Jean-Yves Belmonte, Pixware Immeuble Capricorne 23 rue Colbert 78180 Montigny Le Bretonneux France http://www.pixware.frwww.pixware.fr mailto:hussein@pixware.frhussein@pixware.fr Phone: +33 (0)1 30 60 07 00 Fax: +33 (0)1 30 96 05 23 ——————————————————————-
July 13, 2005

1.2 Introduction

APT (Almost Plain Text) is a simple markup language (like HTML) than can be used to write simple article-like documents (like HTML). Unlike HTML, APT uses as few tags as possible to express the structure of the document. Instead, APT uses paragraph indentation.

The benefits of using APT are:

- APT documents are not tedious to type using text editors.
- When writing an APT document using a text editor, what you type is readable (i.e. not obfuscated by markup).
- APT documents can be embedded in source code (C, C++, Tcl) comments (a la javadoc).
- APT documents can be converted to many formats (currently LaTeX, PS, PDF, HTML, SGML or XML/DocBook, RTF) using aptconvert (lightweight, 100 Java, OpenSource tool).
- APT is content oriented and strictly structured. That's why APT documents are perennial. For example, it is possible to convert them to DocBook with almost no loss of structure.

The drawbacks of using APT are:

- You are limited to simple (simplistic?) article-like documents.
- The style of the documents generated by aptconvert is simple and neutral. It cannot be parametrized to enforce an entreprise specific look.
- Using a text editor to author structured documents is clearly not the wave of the future (which our current project *xmledit* is :-).

2 Java ME

von Marcel Schneider 15. 8. 2010

Das Projekt *Laugh* verwendet zum Kompilieren und Testen der Programme für JavaME verschiedene Plattformen und Emulatoren, vor allem das *Sun WTK 2.5.2* und den *Microemulator*, deren Verwendung hier beschrieben wird.

2.1 Microemulator

Der Microemulator ist ein für Java SE geschriebenes Open-Source Projekt, das das Kompilieren und Ausführen von Handyprogrammen unter Java SE ermöglicht. Er wird hauptsächlich zum Testen der Handyprogramme verwendet. Außerdem hilft er, wenn es beim Kompilieren der *Laugh-Library* für Java SE Kompatibilitätsprobleme gibt.

Vor- und Nachteile

Vorteile

- **Einfach.** Der Microemu macht normalerweise nirgends Probleme.
- **Schnell.** Der Microemu startet sehr viel schneller als der WTK-Emu.
- **Nur ein Projekt nötig.** Der Microemu kann einfach aus einem Java-SE Projekt in Netbeans heraus verwendet werden.

Nachteile

- **Unrealistisch.** Der Microemu lässt vieles zu, was auf Handys nicht funktioniert, Programme laufen sehr viel schneller wie auf Handys.
- **Kompilieren für Handys nicht möglich** – das geht nur mit dem WTK.
- **Keine Netbeans-Unterstützung.** Da Netbeans nur ein SE-Projekt sieht, funktionieren die Hinweise, was in Java ME alles *nicht* möglich ist, im Editor.
- **Nicht alle APIs.** Der Microemu bringt keine der besonderen JSR-Erweiterungen, die die Handys haben, mit. Diese können aber durch weitere SE-Librarys ergänzt werden.

Verwendung

Im Prinzip kann man die Datei `microemulator.jar` als Java-Programm starten und dann eine Java ME Jar-Datei laden.

Zum Testen gibt es aber ein komfortableres Verfahren: Wenn der Microemulator als Library ins Projekt eingebunden ist, kann eine `main`-Methode geschrieben werden, die den Microemulator mit einer bestimmten Midlet-Klasse startet. Eine solche Klasse aknn zum Beispiel so aussehen:

```
1        public static void main(String[] args) {
2            String[] name = new String[1];
3            name[0] = "laugh.midlet.btpong.BtPong";
4            org.microemu.app.Main.main(name);
5        }
```

Klassen mit solchen Methoden befinden sich für alle MIDlets des laugh-Projekts im Paket `laugh.midletse`.

Die Klassen können in Netbeans einfach über Rechtsklick->Ausführen gestartet werden. (Bei den Programmen, die Bluetooth verwenden, kann es Probleme geben, siehe nächster Abschnitt)

Bluetooth mit Bluecove

Um im Microemulator auch Programme, die Bluetooth verwenden, testen zu können, wird die Library *Bluecove* verwendet – ein weiteres Open-Source-Projekt.

Die Verwendung kann Probleme bereiten, wenn nicht alles richtig eingerichtet ist...

Bluecove mit Bluetooth

Normalerweise versucht Bluecove auf verschiedenen Wegen mit einen realen Bluetooth-Gerät am PC in Verbindung zu treten und damit echten Bluetooth-Funkverkehr durchzuführen.

Dies ist eine sehr nützliche Funktion zum Testen eines Programms auf einem Handy, weil man 1. kein zweites Handy braucht und 2. man wenigstens einen Kommunikationspartner vollständig debuggen kann.

Probleme gibts natürlich wenn man am PC keinen Bluetooth-Adapter hat bzw. Bluecove diesen nicht verwenden kann. Dies führt zur 1. häufigen Fehlermeldung beim starten eines Programms im Microemulator:

```
javax.bluetooth.BluetoothStateException: Bluetooth Device is not available
        at com.intel.bluetooth.BluetoothStackBlueZ.nativeGetDeviceID(Nativ
Method)
        at com.intel.bluetooth.BluetoothStackBlueZ.initialize(BluetoothSta
        at com.intel.bluetooth.BlueCoveImpl.setBluetoothStack(BlueCoveImpl
        at com.intel.bluetooth.BlueCoveImpl.detectStack(BlueCoveImpl.java:
        at com.intel.bluetooth.BlueCoveImpl.access$500(BlueCoveImpl.java:6
        at com.intel.bluetooth.BlueCoveImpl$1.run(BlueCoveImpl.java:1020)
        at java.security.AccessController.doPrivileged(Native Method)
        at com.intel.bluetooth.BlueCoveImpl.detectStackPrivileged(BlueCove
        at com.intel.bluetooth.BlueCoveImpl.getBluetoothStack(BlueCoveImpl
        at javax.bluetooth.LocalDevice.getLocalDeviceInstance(LocalDevice.
        at javax.bluetooth.LocalDevice.getLocalDevice(LocalDevice.java:95)
Loading MIDlet class laugh/libme/settings/FehlerSpeicher of version 49
1.5
          org.microemu.app.classloader.ChangeCallsClassVisitor.visit(Chang
        at laugh.libme.settings.specs.getBluetoothMacAdress(specs.java:37)
        at laugh.midlet.multiwurm.multiWurmCanvas.<init>(multiWurmCanvas.j
        at laugh.midlet.multiwurm.multiWurm.init(multiWurm.java:17)
        at laugh.libme.tools.laughMidlet.startApp(laughMidlet.java:36)
```

So oder so ähnlich. Abhilfe schafft der Emulator...

Der Bluecove-Emulator

Bluecove bietet auch die Möglichkeit, Bluetooth lokal zu emulieren. Dazu muss eine *Java-Systemproperty* gesetzt werden. In Netbeans ist das möglich, in dem man unter Projekteinstellungen -> Ausführen bei `VM-Optionen` folgende Option hinzufügt:

```
1    -Dbluecove.stack=emulator
```

Diese Option sollte meistens schon gesetzt sein.

Das allein hilft allerdings noch nicht . Beim Starten erhält man denn die 2. häufig Fehlermeldung:

```
javax.bluetooth.BluetoothStateException: Connection refused to host:
localhost
        at com.intel.bluetooth.EmulatorHelper.createNewLocalDevice(Emulato
        at com.intel.bluetooth.BluetoothEmulator.initialize(BluetoothEmula
        at com.intel.bluetooth.BlueCoveImpl.setBluetoothStack(BlueCoveImpl
        at com.intel.bluetooth.BlueCoveImpl.detectStack(BlueCoveImpl.java:
        at com.intel.bluetooth.BlueCoveImpl.access$500(BlueCoveImpl.java:6
        at com.intel.bluetooth.BlueCoveImpl$1.run(BlueCoveImpl.java:1020)
        at java.security.AccessController.doPrivileged(Native Method)
        at com.intel.bluetooth.BlueCoveImpl.detectStackPrivileged(BlueCove
        at com.intel.bluetooth.BlueCoveImpl.getBluetoothStack(BlueCoveImpl
        at javax.bluetooth.LocalDevice.getLocalDeviceInstance(LocalDevice.
```

Das Problem ist ganz ganz einfach, das der Bluecove-Emulator einen Server benötigt, den man ganz einfach durch ausführen der Klasse `laugh.server.bluecoveemu.ServerMain` starten kann. Dann sollte alles funktionieren.

2.2 WTK 2.5.2

WTK 2.5.2 vs. Mobile SDK 3

Das *Wireless Toolkit* von Sun ist die offizielle Entwicklungsplattform für Java ME. Inzwischen wurde es *Mobile SDK 3* abgelöst, von uns wurde aber trotzdem das ältere WTK 2.5.2 verwendet, weil das SDK 3 nur für Windows verfügbar ist.

Die Netbeans-JavaME-Unterstützung basiert auf dem WTK bzw. Mobile SDK. Beim Mobile SDK ist normalerweise eine auf Netbeans basierende IDE enthalten, das normale Netbeans-Plugin verwendet allerdings im Hintergrund auch das SDK 3 (Windows) bzw. WTK 2.5.2 (andere Plattformen).

Das Laugh-Projekt benötigt unbedingt das WTK 2.5.2, wenn man unsere Projekte verwenden will. Unter Windows muss deshalb das WTK 2.5.2 nachinstalliert werden, sonst kann die Version des Netbeans-Plugins verwendet werden.

Im folgenden gehe ich immer vom WTK 2.5.2 aus.

Vor- und Nachteile

Vorteile

- **Echter Emulator.** Das WTK stellt einen echten Emulator bereit, der sich auch halbwegs realistisch verhält (zumindest meistens)
- **Zusatzapis.** Das WTK unterstützt viele JSR-Erweiterungen für die ME-Plattform. Allerdings werden einige erst im neueren SDK unterstützt, z. B. JSR256 für Sensoren.
- **Netbeans-Unterstützung.** Wenn man den Netbeans-ME-Projekttyp verwendet, erhält man viele Hilfestellungen.
- **Richtig kompilieren.** Nur mit dem WTK kann man Dateien erzeugen, die auch auf einem Handy laufen

Nachteile

- **Langsam.** Der WTK-Emu ist extrem langsam, sowohl im Starten als auch im ausführen, langsamer als viele reale Handys.
- **Separates Projekt.** Man kann nicht ohne weiteres ME- und SE-Klassen mischen.

Verwendung

Zum Programmieren

Laugh-Library

Zum echte Programme erstellen ist es recht angenehm das Netbeans-Plugin zu verwenden, allerdings ist das mit unserem Projekt nicht so einfach. Am einfachsten ist es, die Quelltextdateien das eigentlichen Programms in ein neues ME-Projekt zu kopieren und die laugh-Library `library.jar` fertig kompiliert einzubinden.

Sourcen in meheren Projekten verwenden

Alternativ ist es möglich, ein zweites ME-Projekt zu verwenden, das die selben Quelltexte wie das Hauptprojekt verwendet. Ein solches Projekt existiert im biggame-Projekt unter dem Namen `me`. Ich empfehle aber, dieses Projekt im Moment **nicht** zu verwenden, da vermutlich aufgrund eines Bugs in Netbeans im Netbeans-Editor Fehler angezeigt werden, wo gar keine sind, auch wenn das Projekt wieder geschlossen wird.

Es ist allerdings möglich dieses Projekt zu Verwenden, wenn man sich an den vielen roten Ausrufezeichen nicht stört. Evtl. kann darauf in Zukunft ein System zum kompilieren der ME-Klassen aus dem SE-Projekt heraus aufgebaut werden.

Eine Variante, die fehlerfrei funktioniert, ist es, ein neues ME-Projekt zu erstellen und dann die betreffenden Quelldateien als Symlinks einzubinden. Dies ist jedoch nur unter Linux

möglich (evtl. auch unter Win Vista aufwärts; neuere NTFS-Versionen unterstützen ebenfalls Symlinks)

Zum Kompilieren

Zum Kompilieren für Handys muss das WTK verwendet werden. Im Laugh-Projekt wurde dazu ein eigenes Programm verwendet, das die nötigen Schritte ausführt, evtl. wird es in Zukunft für biggame eine andere Lösung geben.

Zum laugh-Buildsystem hier ein Auszug aus der laugh-Doku.

laugh-Buildtools

Um aus einem Projekt heraus die verschiedenen Handyprogramme und Libraries für Java ME und -SE zu kompilieren sind die normalen Systeme zum Kompilieren von Java-Programmen nur begrenzt geeignet. Deshalb erschien es am sinnvollsten, ein eigenes Java-Programm zu schreiben, das die verschiedenen Kompilierungsschritte ausführt plattformunabhängig alle JAR-Archive für unser Projekt erzeugen kann.

Arbeitsweise

Das Programm build.java liest zunächst zwei Konfigurationsdateien, und beginnt dann, die MIDlets zu kompilieren. Dazu wird an den in der Konfiguration angegeben Orten nach MANIFEST.MF-Dateien gesucht. Wenn solche Dateien gefunden werden, werden aus diesen Dateien Informationen über die Hauptklassen entnommen und diese durch Aufruf von javac kompiliert. Wenn alle Klassen kompiliert sind, wird preverify aufgerufen, um die Klassen für die Verwendung in Java ME vorzubereiten. Dann werden die JAR-Dateien für die MIDlets mit den Informationen aus den MANIFEST.MF-Dateien und den Konfigurationsdateien durch Aufruf von jar gepackt, außerdem werden in den Quelltext-Ordnern zusätzliche Resourcen, z. B. Bilder, gesucht und mit eingeschlossen. Die Library für Java ME wird zusammen mit dem MIDlets kompiliert.

In einem weiteren Schritt werden noch alle MIDlets zusammen in ein JAR-Archiv als Midlet-Suite verpackt. In die MANIFEST.MF-Datei dieses Archivs werden die Informationen aus `src/laugh/midlet/MANIFEST.MF` mit eingeschlossen. Im letzten Schritt werden alle auch für Java SE verwendbaren Klassen noch einmal für für Java SE kompiliert und als Library in eine JAR-Datei gepackt.

Verwendung

Das Programm build muss im Projektordner ausgeführt werden, dies kann zum Beispiel aus der Netbeans-IDE heraus geschehen. Die erste Konfigurationsdatei wird als Ressource aus den Quelltext-Ordnern gelesen, sie befindet sich im Unterordner src/laugh/buildtools/build.conf. Die zweite Datei befindet sich außerhalb des Projektordners im selben Verzeichnis wie dieser und heißt ebenfalls build.conf. Sie muss nicht unbedingt existieren, in ihr können aber Einstellungen lokal angepasst werden, ohne dass sie durch dass Versionskontrollsystem beeinflusst werden. Beide Dateien enthalten Einstellungen zeilenweise nach dem Muster`<eigenschaft=wert`, wobei mit // beginnende Zeilen ignoriert werden, und können folgende Einträge enthalten:

- packages: alle Unterpakete von laugh, die Klassen für die MIDlets enthalten.
- packages-jar: die Unterpakete von laugh, die in alle MIDlet-JAR-Dateien eingeschlossen werden sollen.

- midlet-dir: der Unterordner von laugh, in dem nach MIDlet-Ordnern mit MANIFEST.MF-Dateien gesucht wird.
- resource-regex: ein regulärer Ausdruck, der festlegt, welche weiteren Dateien in die MIDlet-JARs eingeschlossen werden sollen
- libse-dummy: eine Klasse, die alle Programmteile, die nachher in der Library für Java SE vorhanden sein sollen, verwendet. Die Klasse selbst muss sonst keinen weiteren Nutzen haben.

Zusätzlich zu diesen allgemeinen Einstellungen gibt es noch drei weitere, die an das lokale System angepasst werden müssen:

- wtkhome: Pfad zu dem Ordner, der das für die Kompilierung der Java ME Teile notwendige WTK 2.5.2 enthält.
- javac: der Pfad zum Programm javac, dem Java-Compiler. Wenn das JDK richtig installiert ist, genügt javac.
- jar: der Pfad zum Programm jar. Wenn das JDK richtig installiert ist, genügt jar.

Damit ein MIDlet automatisch mit kompiliert wird, müssen sich nur die Klassen in einem Unterpaket des in midlet-dir angegeben Ordners befinden (normalerweise src/laugh/midlet/), zusammen mit einer Datei namens MANIFEST.MF, die mindestens die Zeile der Form `MIDlet-1: [Programmname],[Optional: Pfad zum Icon],`Voller Klassenname des MIDlet enthält, in diesem Ordner befinden. Weitere Zeilen, die für das Funktionieren des Programms nötig sein können, können ebenfalls eingefügt werden. Es wird dann eine JAR-Datei mit dem Namen des MIDlet-Ordners im Projektordner erzeugt, außerdem außerdem wird das MIDlet in die Datei Laugh.jar zusammen mit allen anderen MIDlets eingeschlossen.

Um weitere Dateien, z. B. Bilder, in die JAR-Datei einzufügen, müssen sich diese Dateien im MIDlet-Ordner befinden und ihr Name muss zu resource-regex passen (normalerweise genügt es, wenn jar im Namen auftaucht). Damit Klassen in die Java ME-Library (library.jar) eingeschlossen werden, müssen sie von der Klasse laugh.midlet.library.libdummy verwendet werden. Damit Klassen in die Java SE-Library (libse.jar) eingeschlossen werden, müssen sie von der in der Konfigurationsdatei angegeben Klasse verwendet werden.

Beispiele

Der normale Inhalt der Datei build.conf:

```
1  packages=lib,libme,midlet
2  packages-jar=lib,libme
3  midlet-dir=midlet
4  resource-regex=.*jar.*
5  libse-dummy=laugh/libse/libdummy.java
6  wtkhome=/home/marcel/Java /.netbeans-6.8/mobility8/WTK2.5.2
7  jar=jar
8  javac=javac
```