J2ME Polish

Automatische Optimierung auf Endgeräte und Design per CSS

J2ME Entwickler stehen vor einem Dilemma: entweder sie nutzen nur den kleinsten gemeinsamen Nenner der J2ME Geräte – sprich den MIDP/1.0 Standard – oder aber sie unterstützen nur einige Handys, für die sie ihre Anwendung mit hohen Aufwand optimieren. Das Open Source Tool J2ME Polish verspricht einen Ausweg aus dieser Misere und mehr.

Die J2ME Politur „J2ME Polish“ besteht aus einem Ant-basierten Build-Prozess und einer darauf abgestimmten optionalen J2ME GUI, die mittels CSS designed werden kann. Der Entwickler kann sich dank des Build-Prozesses nur auf den MIDP/1.0 oder MIDP/2.0-Standard beschränken. Die Optimierung auf die einzelnen Geräte und die Einbindung der GUI erfolgt automatisch durch das Preprocessing der Quellcodes im Build-Prozess und einer erweiterbaren Device-Datenbank.

Steckbrief

J2ME Polish

Editionen: GPL (für Open Source Produkte), Early Bird (Einführungs-Edition inkl. Updates bis Version 1.0, 99,- €), Single License (199,-€ gültig für eine J2ME Anwendung, Runtime License (199,-€ gültig für beliebig viele Anwendungen, aber insgesamt max. 100 Anwendungs-Installationen), Enterprise License (2990,-€ gültig für beliebig viele J2ME Anwendungen). Zusätzlich muss für jeden Entwickler ein „Developer Seat“ für 99,-€ erworben werden.

Betriebssysteme: Windows, Linux, OS X

Voraussetzungen: Java Wireless Toolkit, Ant

Build-Prozess: Preprocessing, Device-Optimierung, Geräte-Datenbank, Compile, Preverify, Obfuscation, Jar und Jad

GUI: Design per CSS, gezielte Einbindung von Resourcen für bestimmte Geräte oder Geräte-Gruppen, 100% kompatibel zum MIDP-Standard

www.j2mepolish.org

Ein Beispiel

In unserem Beispiel verwenden wir eine einfache Auswahlliste, wie sie als Hauptmenü in etlichen Spielen vorkommt (Listing 1).

Ant Einstellungen

Das kompilieren, preverifying und packaging

Installation

J2ME Polish enthält einen grafischen Installer, der alle notwendigen Einstellungen vornimmt. Am besten wird J2ME Polish in einen neuen Ordner im Workspace der Lieblings-IDE installiert. Die enthaltene Beispielsanwendung kann sofort nach der Installation mit dem Kommando „ant“ auf der Kommandozeile getestet werden.

Listing 1: MenuMidlet.java

```
public class MenuMidlet extends MIDlet {

List menuScreen;

public MenuMidlet() {
 super();
 System.out.println("starting MenuMidlet");
 this.menuScreen = new List("J2ME Polish",
List.IMPLICIT);
 this.menuScreen.append("Start game", null);
 this.menuScreen.append("Load game", null);
 this.menuScreen.append("Highscore", null);
 this.menuScreen.append("Quit", null);
 this.menuScreen.setCommandListener(this);
 System.out.println("intialisation done.");
}

protected void startApp() throws
MIDletStateChangeException {
 System.out.println("setting display.");
 Display.getDisplay(this).setCurrent(
this.menuScreen );
}
[...]
```

übernimmt J2ME Polish. Dafür benötigen wir eine Datei namens build.xml, mit welcher der build-Prozess gesteuert wird (Listing 2). In dieser Datei muss zunächst das Ant-Property „wtk.home“ gesetzt werden, damit J2ME Polish das Wireless Toolkit finden kann. Danach wird der J2ME Polish-Task definiert. Der Task selbst unterteilt sich in die drei Bereiche „info“, „deviceRequirements“ und „build“.
Im „info“-Bereich werden allgemeine Angaben gemacht, wie beispielsweise die Beschreibung der Anwendung.
Interessanter ist der „deviceRequirements“-Bereich. Hier werden die Geräte ausgewählt, die man unterstützen möchte. Dies kann sehr speziell erfolgen wie durch Angabe des Geräts oder des Herstellers, oder aber auch indirekt durch benötigte Fähigkeiten der Geräte, wie beispielsweise die Unterstützung einer API, das Vorhandensein einer Kamera oder eine Mindest-Farbanzahl. Benötigte Fähigkeiten lassen sich durch and, or und xor sehr fein granuliert einstellen. Die Geräte-Auswahl kann auch aufgeteilt werden, damit sich Testläufe nur auf ein Gerät beschränken und richtige Builds dann alle gewünschten Geräte umfassen.
Der „build“-Bereich beschreibt Einzelheiten des eigentlichen Build-Vorgangs. Hier müssen insbesondere die MIDlet-Klassen angegeben werden. Eine interessante Einstellung ist im Beispiel das Attribut `fullscreen="menu"`, mit dem die Anwendung automatisch FullScreen-Klassen nutzt, sofern das jeweilige Gerät dies unterstützt. „menu“ besagt weiterhin, dass die Anwendung Commands nutzt, ein „yes“ würde nur reine „FullScreen“-Klassen nutzen, ohne dass Commands unterstützt werden.

Listing 2: build.xml

```
<project name="example" default="j2mepolish">

<property name="wtk.home" value="C:\wtk2.1">

<taskdef name="j2mepolish"
 classname="de.enough.polish.ant.PolishTask"
 classpath="import/enough-j2mepolish-
build.jar:import/jdom.jar:import/proguard.jar"/
>

<target name="j2mepolish">
<j2mepolish>
 <info
   name="SimpleMenu"
   version="1.0.0"
   description="A test project"
   jarName="${polish.vendor}-${polish.name}-
example.jar"
   jarUrl="${polish.jarName}"
 </info>
 <deviceRequirements>
  <requirement name="JavaPackage"
               value="nokia-ui" />
 </deviceRequirements>
 <build
   fullscreen="menu"
   usePolishGui="true"
  <midlet class="MenuMidlet" name="Example"/>
 </build>
</j2mepolish>
</target>
</project>
```

Design Einstellungen

Um J2ME Polish nutzen zu können, muss nur noch das Design festgelegt werden. Dafür erstellt man im Projekt die Datei „resources/polish.css“ (Listing 3). In dieser Datei wird mittels CSS das Design festgelegt. Um geräte-spezifische Anpassungen vorzunehmen, können für Hersteller, Geräte und Geräte-Gruppen die Grundeinstellungen erweitert oder überschrieben werden. Dafür legt man die Datei „polish.css“ einfach in den entsprechenden Unterverzeichnissen ab. Einstellungen für den Hersteller Siemens kommen so in „resources/Siemens/polish.css“, Einstellungen für alle MIDP/2.0 Geräte in „resources/midp2/polish.css“ und Einstellungen für das Nokia 6600 Handy schliesslich in „resources/Nokia/6600/polish.css“. Dasselbe Prinzip wird für alle weiteren Resourcen wie Bilder oder Filme genutzt. So können beispielsweise im Ordner „resources“ die allgemeinen Icon-Bilder abgelegt werden, während in „resources/BitsPerPixel.8+“ die gleichnamigen farbigen Icons für alle Handys mit einer Farbtiefe von mindestens 8 Bits pro Pixel residieren.

J2ME Polish unterstützt das CSS Box Modell, d.h. jedes Item hat einen Abstand (margin) zu anderen Items

und einen Innenabstand (padding) zum eigentlichen Inhalt des Items. Hintergründe und Ränder können sehr weitgehend angepasst werden, so können beispielsweise auch animierte Hintergründe eingebunden werden. Um gezielt Einstellungen vorzunehmen, können Selektoren kombiniert werden: mit „form p“ werden zum Beispiel alle Textelemente in Formularen selektiert. Um die Listen-Elemente unseres Beispiels zu selektieren wird der Selektor „listitem“ genutzt. Listen-Elemente unterstützen ebenso wie Elemente einer ChoiceGroup Bilder, die mittels der CSS-Attribute „icon-image“ und „icon-image-align“ gesteuert werden können:

```
listitem {
 [...]
 icon-image: url( icon%INDEX%.png );
 icon-image-align: left;
}
```

Die URL verweist dabei im Beispiel auf die Bilder-Dateien „icon0.png“, „icon1.png“ usw. Die Nummerierung wird dabei mittels des Stichworts „%INDEX%“ vorgenommen. Die Position des Bildes relativ zum Text kann mit dem Attribute „icon-image-align“ bestimmt werden, zur Auswahl stehen hier „left“, „right“, „top“ und „bottom“. Um die Listen-Elemente noch weiter aufzupeppen, soll zusätzlich hinter dem gerade ausgewählten Element ein kleines Bildchen angezeigt werden. Dazu wird für das Attribut „after“ im „focused“- Stil genutzt:

```
focused {
 [...]
 after: url( heart.png );
}
```

Zwischen-Ergebnis

Wenn wir nun den Build-Prozess mittels „ant“ auf der Kommandozeile oder mittels Rechtsklick auf build.xml in der Lieblings-IDE und „Run Ant“ ausgeführt haben, finden wir im „dist“-Ordner des Projekts die erstellte Anwendung (Abb. 1).

Listing 3: polish.css

```
colors {
  pink:  rgb(248,39,186);
  darkpink:  rgb(185,26,138);
}

title {
 padding: 2; margin-bottom: 5; font-
face: proportional; font-size: large;
font-style: bold; font-color: white;
 background-color: darkpink;
 border: none;
 layout: center | expand;
}

focused {
 padding: 3; padding-left: 10; padding-
 right: 5; padding-horizontal: 10;
 background {
  type: round-rect; arc: 8; color: pink;
 }
 border {
  type: round-rect; arc: 8; color:
  yellow; width: 2;
 }
 font {
 style: bold; color: black; size: small;
 }
 layout: expand | left;
 after: url(heart.png);
}

list {
 padding: 5; padding-left: 15; padding-
 right: 15;
 background {
  type: pulsating; start-color: white;
  end-color: pink; steps: 30; repeat:
  false; back-and-forth: false;
 }
 layout: expand | center | vertical-
center;
}

listitem {
 margin: 2; padding: 3;padding-left: 10;
 padding-right: 5; padding-horizontal:
 10;
 background: none;
 font-color: white; font-style: bold;
 font-size: small;
 layout: left;
 icon-image: url( icon%INDEX%.png );
 icon-image-align: left;
}
```

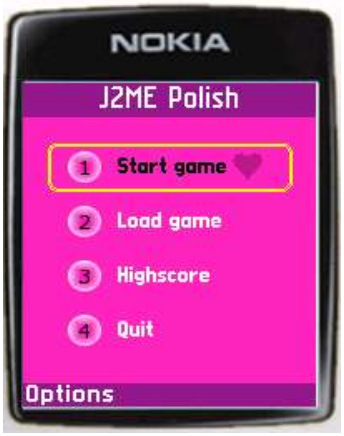

Abb. 1: Die Anwendung, wie sie von J2ME Polish für ein Series 60 Gerät optimiert wird. Was man nicht sieht: der Hintergrund ist animiert.

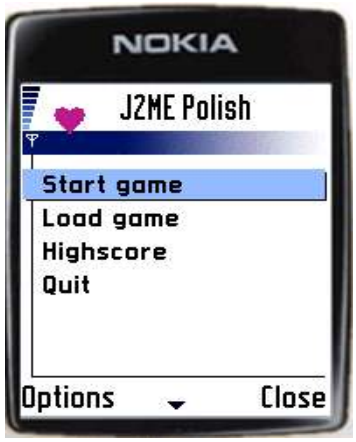

Abb. 2: Dieselbe Anwendung ohne J2ME Polish GUI: funktionales aber unveränderbares Design.

Dieselbe Anwendung kann natürlich auch ohne die erweiterte GUI von J2ME Polish kompiliert werden, beispielsweise indem das „usePolishGui“-Attribut in der „build.xml“-Datei auf „false“ gesetzt wird (Abb.2).

Ein Alternativ-Design

Ein grosser Vorteil von J2ME Polish ist es, dass verschiedene Designs leicht ausprobiert werden können: Einfach ein alternatives Design in einen neuen Ordner erstellen und dann das „resDir“-Attribut des <build>-Elements in der build.xml auf den neuen Ordner setzen.

Wir können beispielsweise ein Hintergrundbild nutzen und die Icons in einer Tabelle anordnen (Abb. 3). Dafür wird in der Datei polish.css der Stil „List“ folgendermaßen geändert:

```
list {
 [...]
 background-image: url( bg.png );
 columns: 2;
}
```

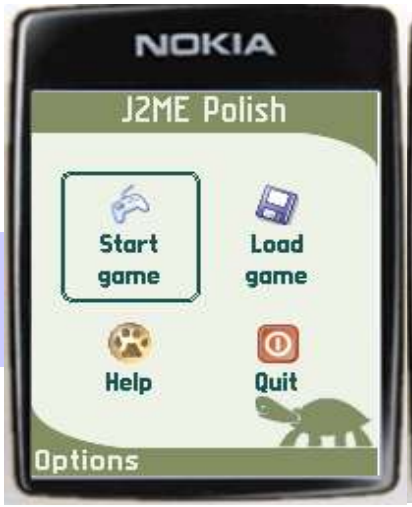

Abb. 3: Die Beispiels-Anwendung mit einem tabellarischen Design

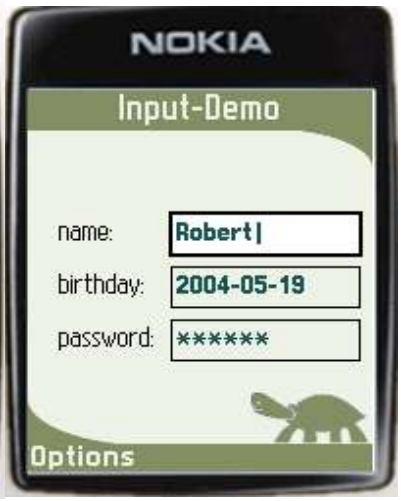

Abb. 4: Tabellen eignen sich auch sehr gut für Eingaben.

Tabellen eignen sich auch sehr gut, um Eingaben zu strukturieren. Dank der pixelgenauen Design-Möglichkeiten kann dabei der Text des Labels und der eigentliche Eingabe-Text auf einer Linie gesetzt werden (Abb. 4). Die Breite der Spalten kann pixelgenau mit dem Attribut „columns-width“ angegeben werden, sofern dies nötig ist.

Statische Stile

Das oben aufgeführte Beispiel nutzt dynamische CSS Stile, also Stile, die erst zur Laufzeit zugewiesen werden. Flexibler und auch schneller sind dagegen statische Stile, die bereits zur Compile-Zeit eingebunden werden. Dazu wird die `//#style` Preprocessing Direktive genutzt:

```
//#style mainScreen
this.menuScreen = new List("J2ME Polish", List.IMPLICIT);
//#style mainCommand
this.menuScreen.append("Start game", null);
```
In der polish.css-Datei müssen dann die so benannten Stile anstelle von „list“ und „listitem“ benutzt werden:

```
.mainScreen {
 [...]
}
.mainCommand {
 [...]
}
```
Dabei beginnen die Selektoren von statischen Stilen in der CSS-Datei immer mit einem Punkt. Bei Verweisen auf solche Stile kann der Punkt auch weggelassen werden.

Logging

Logging unter J2ME ist schwierig. Zum einen verbrauchen diese Logging-Meldungen Speicherplatz und Rechenzeit und zum anderen gibt es unter J2ME keine Logging-Frameworks wie log4j. J2ME Polish enthält daher ein Logging-Framework mit folgenden Features:

- Der Logging-Level (debug, info, warn, error etc.) kann für einzelne Klassen oder Packages eingestellt werden. Auch eigene Logging-Level wie zum Beispiel „benchmark“ können genutzt werden.
- Die Logging-Meldungen können auch auf richtigen Geräten eingesehen werden.
- Logging-Meldungen, die nicht aktiv sind, werden überhaupt nicht mitkompiliert.
- Das Logging kann komplett deaktiviert werden.
- Zum loggen werden einfach System.out.println()-Aufrufe genutzt.

Das Logging wird in der build.xml mit dem Element <debug> gesteuert:

```
<debug enable="true" showLogOnError="true" verbose="false" level="error">
    <filter pattern="com.company.package.*" level="info" />
    <filter pattern="com.company.package.MyClass" level="debug" />
</debug>
```
Dabei steuert das „enable“-Attribut global, ob das Logging überhaupt aktiviert werden soll. Mit dem „showLogOnError“-Attribut wird gesteuert, ob das Log automatisch angezeigt werden soll, wenn ein Fehler gelogged wird. Ein Fehler wird zum Beispiel durch folgenden Code gelogged:

```
try {
  callComplexMethod();
  //#debug info
  System.out.println( "complex method succeeded." );
} catch (MyException e) {
  //#debug error
  System.out.println( "complex method failed" + e ); // wenn showLogOnError aktiviert ist,
}                                                    // wird an dieser Stelle das Log eingeblendet
```

Wird das „verbose“-Attribut auf „true“ gesetzt, so wird vor jeder Logging-Meldung die aktuelle Systemzeit, der Dateiname und Programmzeile der Meldung ausgegeben. Damit können auftauchende Probleme sehr schnell und einfach lokalisiert werden, denn auf realen Geräten kann man den Stacktrace einer Ausnahme leider nicht auswerten. Ausserdem werden im „verbose“-Modus genaue Beschreibungen bei Fehlern des J2ME Polish Frameworks ausgegeben. Das „level“-Attribut steuert schließlich den notwendigen Logging-Level, sobald nichts spezielleres mit den <filter>-Elementen eingestellt wurde. Dabei gibt es folgende

Hierarchie:

debug < info < warn < error < fatal < Benutzer-definiert

Wenn also für eine Klasse der Logging-Level „info“ aktiv ist, werden auch alle Meldungen mit einer höheren Gewichtung ausgegeben, also Meldungen mit den Leveln „warn“, „error“, etc.

Im Programmcode werden Logging-Meldungen mit der //#debug Direktiven eingeleitet, die als Parameter den Log-Level angibt (s. obiges Beispiel). Wenn hinter der //#debug Direktiven kein Level angegeben wird, so wird der „debug“-Level angenommen.

Manuelle Optimierungen

Optimierungen für bestimmte Handys betreffen in erster Linie die Benutzeroberfläche und werden von der J2ME Polish GUI automatisch vorgenommen. Es gibt aber auch andere Situationen, in denen manuelle Optimierungen sinnvoll sind. Mit den Preprocessing-Möglichkeiten und der Geräte-Datenbank von J2ME Polish sind solche Optimierungen schnell gemacht, ohne das die Portabilität der Anwendung eingeschränkt wird. Dazu können insbesondere die Direktiven „//#ifdef“, „//#if“, und „//#=“ genutzt werden, die im folgenden kurz vorgestellt werden:

Um Daten sicher zu übertragen, müssen sie entweder verschlüsselt oder über eine gesicherte Verbindung übertragen werden. Nicht alle J2ME Handys unterstützen jedoch den HTTPS-Standard, daher muss dieser Fall im Code unterschieden werden. J2ME Polish bietet dafür die Variable `polish.JavaProtocol`. Wenn das Gerät den HTTPS Standard unterstützt, wird automatisch das Symbol `polish.JavaProtocol.https` definiert und kann im Code abgefragt werden:

```
//#ifdef polish.JavaProtocol.https
   HttpsConnection c = (HttpsConnection)Connector.open( url );
   [...]
//#else
   // create MD5-Hash or encrypt data
   [...]
//#endif
```

Ebenso können die unterstützten APIs eines Geräts mit der Variablen `polish.JavaPackage` abgefragt werden. Wird die Multimedia-API vom Gerät unterstützt, so wird das Symbol `polish.JavaPackage.mmapi` oder auch kurz `polish.api.mmapi` definiert. Im folgenden Beispiel wird zusätzlich abgefragt, ob das Gerät den 3GPP Video-Standard unterstützt:

```
//#if polish.api.mmapi && polish.VideoFormat.3gpp
   // now play Video-File:
   [...]
//#else
   // show simple splash screen:
   [...]
//#endif
```

Variablen-Inhalte können auch verglichen werden, zum Beispiel gibt die Variable „polish.BitsPerPixel“ an, wie gross die Farbtiefe ist:

```
//#if polish.BitsPerPixel >= 12
   // this device uses at least 12 bits per pixel.
//#endif
```

Wenn Anpassungen speziell für einen Hersteller zu machen sind, kann dies mit der Variablen „polish.Vendor“ geschehen:

```
//#if polish.Vendor == Nokia
   // this is a Nokia device
//#endif
```

Auch können Variablen selbst definiert und genutzt werden, so kann beispielsweise eine URL definiert und im Programmcode genutzt werden. Dazu wird in der build.xml einfach eine Variables-Definition hinzugefügt:

```
<variables>
   <variable name="update-url" value="http://www.enough.de/update" />
</variables>
```

Im Programm kann diese URL dann mit der „//#=“-Direktiven genutzt werden:

```
//#ifdef update-url:defined
   //#= public static final String UPDATE_URL ="${ update-url }";
//#else
   public static final String UPDATE_URL ="http://default.com/update";
//#endif
```

Die Geräte-Datenbank von J2ME Polish enthält eine Vielzahl von Variablen für jedes Handy, die zusammen mit den Preprocessing-Direktiven so genutzt werden können, dass man leicht maßgeschneiderte Anwendungen erstellen kann, ohne die Portierungsmöglichkeiten einschränken zu müssen.

Fazit

J2ME Polish bietet sowohl den Entwicklern als auch den Designern von mobilen Anwendungen viele Vorteile:

- Umfassende Design-Möglichkeiten: Es können Bilder hinzugefügt, Schriften verändert, Hintergründe und Farben gesetzt werden, ohne dass der Programmcode geändert werden muss.
- Automatische Optimierungen: Optimierungen der Benutzeroberfläche an verschiedene Geräte sind extrem aufwendig. Die GUI von J2ME Polish adaptiert sich vollautomatisch an die unterschiedlichen Geräte und es ist sehr einfach spezielle Einstellungen für einzelne Geräte und Gerätegruppen vorzunehmen.
- Standard-kompatibel: Entwickler müssen keine neue API lernen, sondern nutzen den MIDP/1.0 oder MIDP/2.0 Standard für die Applikationen. Wenn ältere Geräte die erweiterte GUI nicht unterstützen, so wird für diese eben die native GUI genutzt.
- CSS Design: Designer können mit ihrem erlernten Know-how unabhängig vom Entwickler arbeiten und die Entwickler brauchen sich um das Design der Anwendung nicht mehr zu kümmern.

- Manuelle Optimierungen: Dank der Geräte-Datenbank von J2ME Polish können auch in der Applikation Anpassungen an die Geräte-Eigenschaften sehr einfach durchgeführt werden.

Natürlich kann J2ME Polish fast beliebig erweitert werden. So können eigene Hintergründe, Ränder oder auch eigene Items hinzugefügt und genutzt werden. Diese Erweiterungen sprengen jedoch den Rahmen dieses Artikels.

J2ME Polish wird unter der Open Source Lizenz GPL und einigen kommerziellen Lizenzen vertrieben. Das Tool kann unter http://www.j2mepolish.org runtergeladen werden.

Alternativen

kAWT: Personal Java und J2ME GUI, GPL und kommerziell, kawt.de

Synclast UI API: J2ME GUI, GPL und kommerziell, www.synclast.com/ui_api.jsp

JTGL: Java und J2ME GUI, LGPL und kommerziell, www.jtgl.org

antenna: build tool für J2ME, LGPL, antenna.sourceforge.net