

# Die Profiles der Java 2 Microedition

Thomas Bochtler  
Universität Ulm

[thomas.bochtler@informatik.uni-ulm.de](mailto:thomas.bochtler@informatik.uni-ulm.de)

Ausarbeitung im Rahmen des Proseminars Mobile Java

16.Juni 2004

## Inhaltsverzeichnis

- 1 Einleitung
- 2 CDC Profile
  - 2.1 Connected Device Configuration (CDC)
  - 2.2 Foundation Profile (FP)
  - 2.3 Personal Basis Profile (PBP)
  - 2.4 Personal Profile (PB)
- 3 CLDC Profile
  - 2.5 Connected Limited Device Configuration (CLDC)
  - 2.6 Mobile Information Device Profile (MIDP)
  - 2.7 Information Module Profile (IMP)
- 4 Optional Packages
- 5 Quellen

# 1 Einleitung

In den letzten Jahren hat man einen Boom im Bereich der mobilen Kommunikationsgeräte beobachten können. Dabei kamen verschiedene Technologien auf, um Programme für solche Geräte zu entwickeln. Eine dieser Technologien ist die Java 2 Micro Edition (J2ME). Der Aufbau der J2ME lässt sich anhand der Configurations und der Profiles erklären.

Configurations sind Spezifikationen der Virtual Machine und der Basis API (Application Programming Interface / Programmierschnittstelle). Die Configuration kann auf einer bestimmten Gerätegruppe verwendet werden. Derzeit gibt es zwei Configurations: Die Connected Limited Device Configuration (CLDC), die für Geräte mit 16 oder 32 bit Prozessoren und mindestens 160KB Gesamtspeicher gedacht ist, und die Connected Device Configuration, die auf Geräte mit 32 bit Prozessor und 2MB oder mehr Speicher zielt.

Durch diese Abgrenzung lassen sich Geräte mit bestimmter Hardwareausstattung einer Configuration zuordnen. Innerhalb dieser Gruppen können sich die Geräte in vielen anderen Punkten unterscheiden. So braucht ein Router beispielsweise keine graphische Benutzeroberfläche, ein Organizer hingegen schon.

Profiles sind eine Menge von APIs, welche die Configurations erweitern und in Kombination mit ihnen eine Anwendungsumgebung bilden. Mit der Auswahl eines Profiles kann noch genauer auf die Fähigkeiten der Hardware des Geräts eingegangen werden. Auf diese Weise wird vermieden, dass unbenutzte APIs unnötig Speicherplatz belegen.

Geräte besitzen teilweise sehr spezielle Fähigkeiten, die nicht alle über die Profiles angesprochen werden können. Um diese anzusprechen gibt es schließlich einige Optional Packages.

Im folgenden werden wir die Profiles die auf den beiden Configurations CDC und CLDC aufbauen näher betrachten.

## 2 CDC Profiles

### 2.1 Connected Device Configuration (CDC)

Die Connected Device Configuration (CDC) wurde entwickelt um Java in dem großen Bereich der netzwerkfähigen eingebetteten Geräte, als auch in dem Bereich der netzwerkfähigen Verbrauchergeräte einsetzen zu können. Es zielt auf Geräte mit etwas größeren trotzdem aber beschränkten Ressourcen. Darunter fallen Geräte mit 32-bit Prozessoren und 2MB oder mehr Speicher. Allerdings fehlt meistens eine vollständige Desktop-Benutzeroberfläche mit Fenstern und Mausbedienung. Stattdessen gibt es eine Menüführung beispielsweise mit Stift oder Softkeys.

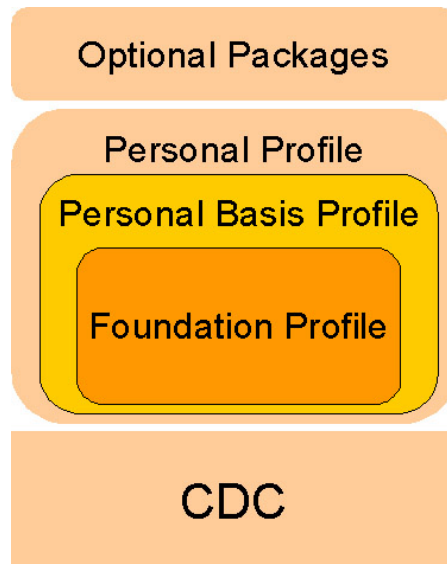
Die CDC besteht aus einer Virtual Machine und einer Menge von Standard APIs. Bei der Zusammenstellung der Basis API wurde darauf geachtet, dass ein hohes Maß an Kompatibilität mit der Java 2 Standard Edition (J2SE) bestehen bleibt. Deswegen leiten sich die APIs, die bereits im CDC verankert sind von denen der J2SE ab. Um Ressourcen zu sparen wurden allerdings manche APIs weggelassen und andere teilweise modifiziert. Außerdem wurden alle Klassen die als „depricated“ gelten weggelassen. Dies sind Klassen, für die neuere, bessere Klassen mit selber Funktionalität existieren. Insgesamt sind Klassen der folgenden Pakete berücksichtigt worden :

- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.lang.math
- java.net
- java.security
- java.security.cert
- java.text
- java.util
- java.util.jar
- java.util.zip

Hierbei ist darauf zu achten, dass aus den einzelnen Paketen bestimmte Klassen und Methoden weggelassen wurden und somit nicht die kompletten Pakete verfügbar sind. Beispielsweise enthält

das Paket `java.net` nur die Klassen, die etwas mit datagram oder dem Umgang mit URLs zu tun haben.

Alle Profiles die auf dem CDC aufbauen enthalten die obigen Pakete und erweitern die CDC zu einer vollständigen Anwendungsumgebung. Bisher gibt es insgesamt drei solcher Profiles: das Foundation Profile (FP), das Personal Basis Profile (PBP) und das Personal Profile (PP). Die Profiles selbst sind geschichtet, so dass alle Pakete des grundlegenden FP wiederum im PBP enthalten und die Pakete des PBP allesamt im PP zu finden sind. Optional Packages sind je nach Bedarf hinzuzufügen.



Schichtenmodell der CDC Profiles

Insgesamt muss allerdings angemerkt werden, dass bisher keine neueren Versionen der Profile erstellt wurden. Zwar hat der Java Community Process eine Aktualisierung der Profile des CDCs in Angriff genommen, konnte sich aber auf keine gemeinsame Linie einigen. Einer Vergrößerung der Klassenbibliotheken steht gegenüber, dass die CDC für Geräte, deren Ressourcen beschränkt sind gedacht ist. Auch gab es grundsätzliche Bedenken an der Architektur und ob diese für längere Zeit vertretbar ist.

Nichtsdestotrotz werden CDC Profile bereits in einigen Bereichen eingesetzt. Deswegen werden wir nun ein wenig genauer auf deren Aufbau eingehen.

## 2.2 Foundation Profile (FP)

Das Foundation Profile ist die unterste Schicht der Profiles und baut direkt auf der CDC auf. Es erweitert hauptsächlich die J2SE APIs die bereits im CDC enthalten waren. Dies sind im einzelnen :

- `java.io`
- `java.lang`
- `java.net`
- `java.security`
- `java.security.cert`
- `java.text`
- `java.util`
- `java.util.jar`
- `java.util.zip`

Außerdem wurden Klassen aus drei weiteren Paketen hinzugefügt :

- `java.security.acl`
- `java.security.interfaces`
- `java.security.spec`

Die Kombination CDC/FP enthält also eine mächtige Umgebung um allgemeine Anwendungen zu erstellen. Allerdings können damit keine interaktiven Anwendungen geschrieben werden. Das Abstract Window Toolkit (AWT) fehlt hier komplett, so dass es nicht möglich ist graphische Benutzeroberflächen zu erstellen. Dies ist durchaus gewollt, da Router oder Netzwerkdrucker keine besonders anspruchsvolle Displays haben.

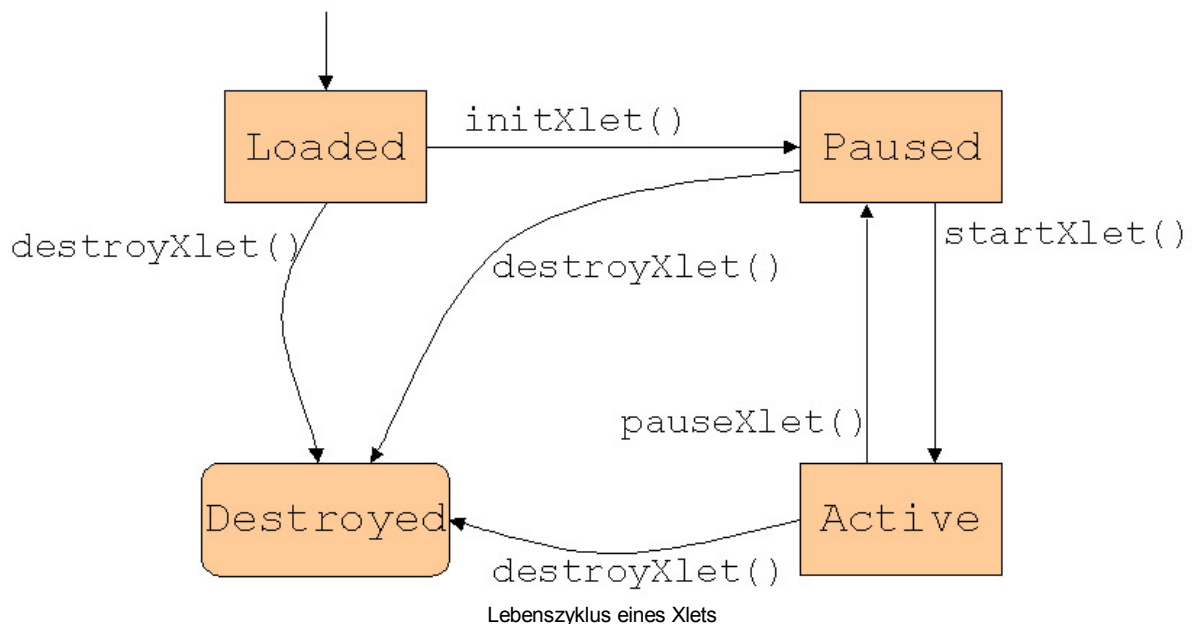
## 2.3 Personal Basis Profile (PBP)

Das Personal Basis Profile liefert zusätzlich zu den bereits im FP enthaltenen Möglichkeiten die Unterstützung graphische Benutzeroberflächen zu erstellen, sowie das xlet application model umzusetzen und es erlaubt ferner mit Teilen des Remote Methode Invocation (RMI) die Inter-Xlet Communication (IXC). Das PBP enthält neben den Klassen in CDC/FP Teile der folgenden Pakete :

- java.awt
- java.awt.color
- java.awt.event
- java.awt.image
- java.beans
- java.rmi
- java.rmi.registry

Mit dem Abstract Window Toolkit (AWT, java.awt) ist es nun also mögliche graphische Benutzeroberflächen zu implementieren. Im PBP wurde allerdings nicht der komplette AWT übernommen, sondern ausschließlich die Klassen der leight-weight Komponenten. Somit enthält der AWT des PBP zum Beispiel keine Buttons, Listen oder Menüs. Eine Ausnahme sind die top-level Fenster, die übernommen wurden, obwohl sie heavy-weight sind. Der Grund dafür ist, dass top-level Fenstern Ereignisse des Betriebssystems mitgeteilt werden müssen. Es handelt sich um die Klassen java.awt.Window und java.awt.Frame. Doch auch diese beiden Komponenten wurden insofern beschränkt, dass nur eine Instanz dieser Klassen zu einer Zeit existieren darf.

Ein andere Name für des PBP Anwendungen ist Xlets. Xlets sind den bekannten Applets sehr ähnlich. Wie Applets werden Xlets von der Software kontrolliert, die sie aufgerufen hat. Xlets haben keine main-Methode und implementieren immer das Interface Xlet. Sie besitzen einen Lebenszyklus, der durch verschieden Methoden von einem Zustand in den nächsten gebracht werden kann. Mittels Inter-Xlet Communication können Xlets sogar untereinander kommunizieren.



Insgesamt ermöglicht die Kombination CDC/PBP interaktive Anwendungen zu schreiben, die einen durch Xlets wohl-definierten Anwendungslebenszyklus haben. Sie wird hauptsächlich bei Geräten gewählt, die interaktives Fernsehen ermöglichen.

## 2.4 Personal Profile (PP)

Das Personal Profile in Kombination mit CDC soll die Spezifikation PersonalJava ersetzen. PersonalJava war Suns erster Schritt Anwendungen für netzwerkfähige, mobile Geräte in Java zu ermöglichen. Anwendungen die für PersonalJava geschrieben wurden können nun auf einfache Weise in Anwendungen für CDC/PP umgeschrieben werden.

Des Weiteren ordnet sich das PP in das Schichtenmodell ein und erweitert das PBP in einigen Punkten. So unterstützt PP das applet application model und enthält eine umfangreichere Teilmenge des AWT. Das PP fügt den CDC/PBP Klassen der folgenden Pakete hinzu :

- `java.applet`
- `java.awt`
- `java.awt.datatransfer`

Um das application model zu unterstützen wurden nun auch heavy-weight Komponenten im AWT des PP aufgenommen. Dies ist notwendig, da sich die Klasse `java.awt.Applet` direkt aus der Klasse `java.awt.Panel` ableitet, und existierende Applets häufig graphische Standardkomponenten wie Buttons und Listen verwenden um ihre Oberfläche aufzubauen. Außerdem beinhaltet der AWT das event model, wiederum um das application model korrekt umsetzen zu können.

Somit ist die Kombination CDC/PP dem J2SE noch ähnlicher und enthält darüber hinaus das xlet application model sowie die Inter-Xlet Communication, welche schon im PBP hinzugefügt wurden.

## 3 CLDC Profiles

### 3.1 Connected Limited Device (CLDC)

Die Connected Limited Device Configuration (CLDC) wurde für Geräte mit sehr beschränkten Ressourcen entwickelt. Diese zeichnen sich dadurch aus, dass ihr Rechengeschwindigkeit, ihre Speicherkapazitäten, als auch die graphischen Darstellungsmöglichkeiten stark begrenzt sind. So ist die Displayfläche stark eingeschränkt, der Arbeitsspeicher zwischen 128 und 512KB, und die Prozessorleistung beträgt 16 oder 32bit. Typischerweise handelt es sich dabei um ein Mobiltelefon oder einen Personal Digital Assistant (PDA). Die Anwendung des CDCs auf andere Geräte ist aber nicht ausgeschlossen. Weiter Anwendungsgebiete wären elektronisches Spielzeug und Haushaltsgeräte. Insgesamt wird das CLDC weit häufiger ausgewählt als das CDC.

Die CLDC enthält die drei Pakete `java.io`, `java.lang` und `java.util`. Diese bestehen aus einer Untermenge der APIs aus J2SE. Der Inhalt dieser Pakete ist daher nichts neues. Allerdings muss man sich auf einen verminderten Umfang einstellen, so fehlt CLDC 1.0 zum Beispiel die Unterstützung von Gleitkommazahlen. CLDC 1.1 behebt diese Manko, setzt allerdings den minimalen Arbeitsspeicher von 128KB auf 192KB.

Für die CLDC gibt es momentan ein Profil auf das sich die meiste Aufmerksamkeit richtet: das Mobile Information Device Profile. Außerdem gibt es das Information Module Profile (IMP). Darüber hinaus arbeitet der Java Community Process mittlerweile auch an dem sogenannten Personal Digital Assistant Profile (PDAP). Dieses wird wohl größtenteils aus dem bereits existierenden Optional Package für PDAs hervorgehen.

### 3.1 Mobile Information Device Profile

MIDP ist das erste CLDC basierte Profile. Da es sowohl ein neues application model definierte, als auch neue Klassen für die Gestaltung von Benutzeroberflächen und der persistenten Speicherung von Daten, hat das MIDP sehr viel Aufmerksamkeit erregt. Jedoch enthielt die erste Version noch einige Verbesserungsmöglichkeiten. Deshalb folgte auf das MIDP 1.0 (JSR 37) bald das MIDP 2.0 (JSR 118). Im weiteren betrachten wir nun zunächst einmal das MIDP 1.0, um danach auf die Verbesserungen einzugehen, die das MIDP 2.0 mit sich brachte.

Zunächst einmal wurde die Minimalen Hardwarevoraussetzungen eines Geräts beschrieben, das der Klasse der mobilen Informationsgeräte (Mobile Information Device / MID) angehört.

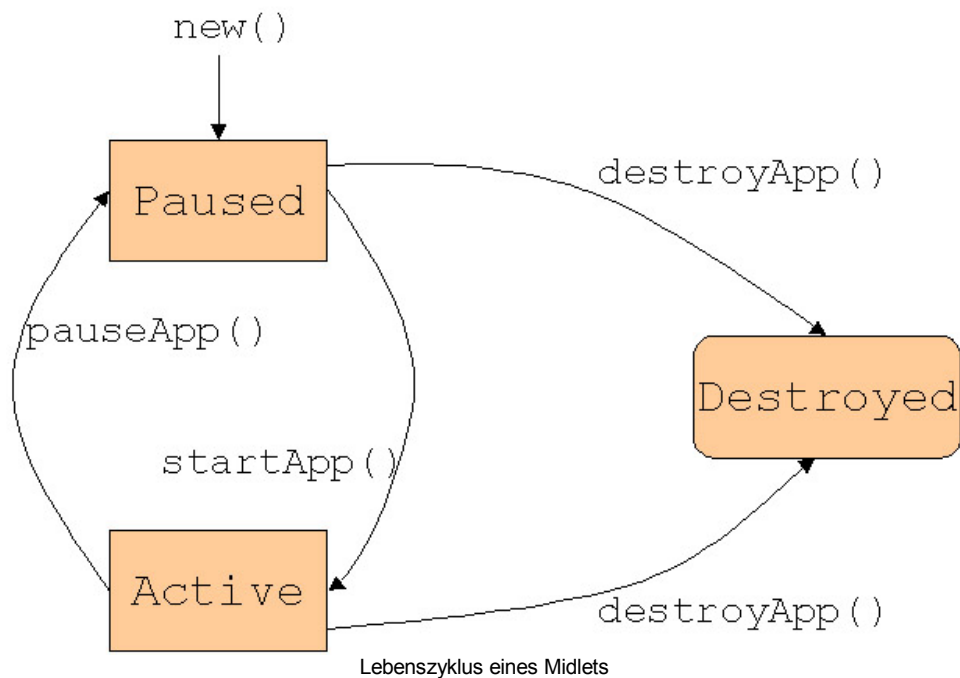
Das Display sollte mindestens eine Größe von 96x54 Pixel haben, die Pixel ungefähr ein Längen zu Breiten Verhältnis von 1:1 und die Bildtiefe mindestens 1-bit betreffen. Es sollte entweder eine einhändige Tastatur, eine zweihändige Tastatur oder einen Touchscreen als Eingabemöglichkeit haben. An Minimum Speicher sollten 128KB für die MIDP Komponenten, 8KB für die dauerhaften Daten der Anwendungen und 32KB flüchtiger Speicher für die Java Laufzeit zur Verfügung stehen. Letztendlich sollte das Gerät eine Netzwerkverbindung haben. Dieses sollte eine bidirektionale, drahtlose, möglicherweise unterbrochene und mit limitierter Bandbreite arbeitende Verbindung sein. Hieraus wird ersichtlich, dass das MIDP hauptsächlich auf Handys zielt. Es kann aber auch bei low-level PDAs oder Pagern eingesetzt werden. Wie schon erwähnt definierte das MIDP einige neue Klassen. Hier eine Liste der Pakete, die das MIDP enthält.

- `java.util`
- `java.lang`
- `javax.microedition.rms`
- `javax.microedition.midlet`
- `javax.microedition.io`
- `javax.microedition.lcdui`

`java.util` und `java.lang` ist eine Untermenge der entsprechend gleichnamigen Pakete des J2SE mit der bekannten Einschränkungen. Auf die anderen werden wir nun einzeln kurz eingehen.

#### `javax.microedition.midlet`

Dieses Paket beschreibt MIDP Anwendungen und die Interaktionen zwischen der Anwendung und der Umgebung, in der die Anwendung abläuft. Solche Anwendungen werden MIDlets genannt und sind den weit mehr bekannten Applets sehr ähnlich. MIDlets befinden sich immer in einem der drei Zustände `Paused`, `Active` oder `Destroyed`. Eine vom Geräte abhängige Management Software reagiert auf Ereignisse und führt die MIDlets durch die Zustände. Zum Beispiel könnten MIDlets bei einem eingehenden Anruf vom Zustand `Active` in den `Paused` Zustand überführt werden. Dies geschieht dann über einen Aufruf wie zum Beispiel `pauseApp()`. Der Entwickler legt über die Methoden `startApp()`, `pauseApp()` und `destroyApp()` fest, wie sich das MIDlet in einer solchen Situation zu verhalten hat. Außerdem kann ein MIDlet der Laufzeitumgebung einen gewünschten Zustandswechsel über die Methoden `notifyDestroyed()`, `notifyPaused()` oder `resumeRequest()` mitteilen.



javax.microedition.io

Dieses Paket ist bereits in der Spezifikation des CLDCs berücksichtigt und enthält das Generic Connection Framework (GCF), welches die Netzwerkmöglichkeiten, die bisher in java.net zu finden waren ersetzt. Zusätzlich zum Standard GCF enthält das Paket auch noch eine Klasse `HttpConnection`, mit der Verbindungen über das http möglich sind. Zentrales Element dieses Pakets ist die `Connector` Klasse. Übergibt man dabei der Methode `open()` eine URL, so erhält man als Rückgabewert je nach URL unterschiedliche Ein-/Ausgabeklasse. Das besondere daran ist, dass mittels der URL auch der Protokolltyp festgelegt wird (zum Beispiel `http://....`, `file://....`).

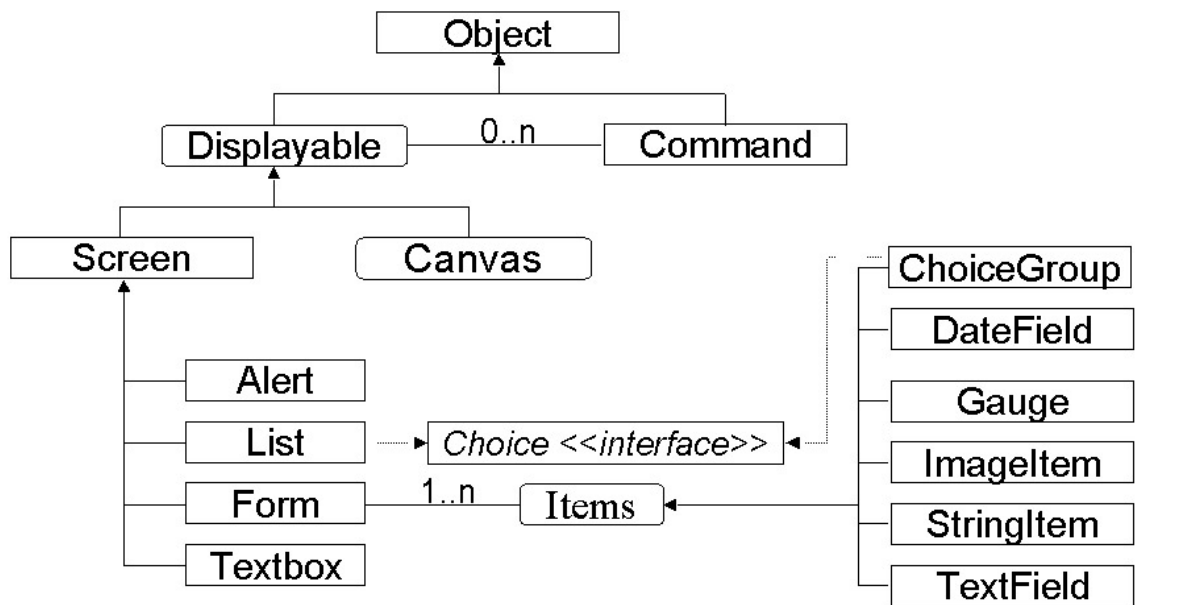
javax.microedition.rms

Das Record Management System (RMS) Paket macht es möglich Daten dauerhaft in Form von Records abzuspeichern und sie später wieder aus dem Speicher zu lesen. Dies kann bei einem Spiel dazu dienen Highscores abzuspeichern. Das Paket stellt dazu die Klasse `RecordStore` bereit. Darüber hinaus werden einige Interfaces um den Umgang mit den Records zur Verfügung gestellt. Dabei handelt es sich um die Interfaces `RecordComparator` (um Records miteinander zu vergleichen), `RecordEnumerator` (zum Traversieren aller Records), `RecordListener` (Empfangen von Ereignisse, falls sich der Inhalt verändert) und `RecordFilter` (zum herausfinden von passenden Einträgen).

javax.mircoedition.lcdui

Die beschränkten Hardwaremöglichkeiten eines Kleingeräts ließen es nicht zu, den AWT aus J2SE einfach zu übernehmen. Deswegen wurde dieses Paket ebenfalls komplett neu zur Programmierung von graphischen Benutzeroberflächen entwickelt. Das Lowest Common Denominator User Interface (LCDUI) enthält eine Menge graphischer Elemente zur Darstellung von Listen (Klasse `List`), Textfeldern (Klasse `TextField`) oder Schalter (Klasse `ChoiceGroup`). Außerdem bietet es die Container Klasse `Form`, in dem solche Elemente (Items) platziert werden können und die Klasse `Graphics`, zum pixelorientiertem Zeichnen von Grafiken.

Weiter existiert ein einfaches Ereignismodell. Die Listener können aber nur dem Display hinzugefügt werden.



Keys : **Class** **Abstract Class** ——— Extends ·····

Klassenstruktur des Pakets javax.microedition.lcdui

Zusammengefasst enthält das MIDP 1.0 APIs für http Verbindungen, die dauerhafte Speicherung von Daten und die Erstellung von Benutzeroberflächen. Dazu definiert mittels MIDlets den Lebenszyklus einer Anwendung für MIDs.

Das MIDP 2.0 erweitert das MIDP 1.0 in vielen Punkten. Es geht dabei vor allem auf die rasanten Verbesserungen der Hardware von MIDs ein und nutzt diese. Dazu stellt es neue Mindestanforderungen an den Speicher der Geräte. Für das MIDP 2.0 müssen mindestens 256KB für die MIDP 2.0 Komponenten, 128KB flüchtiger Speicher für die Java Laufzeit und 8KB dauerhafter Speicher für die Daten der Anwendung zur Verfügung stehen. Ferner sollte das Gerät dazu fähig sein Töne abzuspielen.

Das MIDP 2.0 erleichtert auch die Arbeit des Entwicklers wie zum Beispiel in der Gestaltung von Oberflächen oder der Spieleprogrammierung. Neu sind außerdem die verbesserten Netzwerkfähigkeiten sowie verschlüsselte Verbindungen. Dazu wurde verstärkt das Thema Sicherheit berücksichtigt.

Zusätzlich zu den bereits im MIDP 1.0 enthaltenen Paketen stellt das MIDP 2.0 folgende Pakete zur Verfügung :

- `javax.microedition.media`
- `javax.microedition.media.control`
- `javax.microedition.lcdui.game`
- `javax.microedition.pki`

Diese Pakete brachten folgende Neuerungen mit sich :

### **Multimedia**

`javax.microedition.media` und `javax.microedition.media.control` sind Untermengen der Mobile Media API (MMAPI, JSR 135). Diese Untermenge enthält ausschließlich Klassen mit deren Hilfe Töne, Sequenzen und WAV Dateien abgespielt werden können. Die zentralen Klassen heißen hier `Player` und `Manager`.

### **Game API**

Das Paket `javax.microedition.lcdui.game` vereinfacht die Entwicklung von Spielen. Der Inhalt eines `Screen` kann jetzt in verschiedenen Schichten (`Layer`) aufgebaut werden, wie Hintergrund, Spielfigur, Hindernisse, Nebel oder ähnliches. Mit einem `LayerManager` kann schließlich das Zusammenspiel der verschiedenen `Layer` (`TitledLayer` oder `Sprites`) geregelt werden. Mit der Klasse `GameCanvas` kann nun auch Offscreen-Rendering (Bilder im Hintergrund zeichnen und darstellen wenn sie fertig sind) betrieben werden.

### **Sicherheit**

Das Paket `javax.microedition.pki` enthält die Klassen `Certificate` und `CertificateException`. Mit Hilfe des Pakets kann Programmcode signiert werden. Erkennt das Gerät die Unterschrift so wird das MIDlet als `trusted`, ist dies nicht der Fall, als `untrusted` klassifiziert. `Untrusted MIDlets` wird es verwehrt Verbindungen ohne Zustimmung des Benutzers herzustellen.

### **Größer Verbindungsfähigkeit**

Die Zertifikate werden unter anderem bei sicheren Verbindungen über SSL und https benutzt. Solche Verbindungen waren im MIDP 1.0 jedoch nur über `Optional Packages` verfügbar. Im MIDP 2.0 sind die Klassen `HttpsConnection`, `CommConnection`, `SecureConnection`, `ServerSocketConnection`, `SocketConnection` und `UDPDatagramConnection` im Paket `javax.microedition.io` bereitgestellt und bieten zusammen mit dem GCF eine Schnittstelle zur Verwendung der entsprechenden Standard Verbindungstypen.

### **Oberfläche**

Mit die größten Veränderungen betreffen jedoch ein Paket, das bisher nicht erwähnt wurde : `javax.microedition.lcdui`. So wurde der Container Klasse `Form` ein neues Layout verpasst. `Items` werden nun nicht mehr nur vertikal (untereinander) angeordnet, sondern auch horizontal (nebeneinander), vergleichbar mit dem `FlowLayout` aus J2SE. Je nach Sprachkonvention werden die Elemente in der ersten Reihe der `Form` von links nach rechts oder von rechts nach links eingefügt. Ist eine Reihe voll, so wird eine neue unterhalb der existierende Reihe hinzugefügt. Ob ein Item in einer neuen Reihe platziert werden soll, obwohl

die vorherige noch nicht voll ist, kann mit verschiedenen, vom den Items abhängigen Einstellungen bestimmt werden

Neu hinzugekommen ist dass Items nun eine minimale und eine bevorzugte Größe haben. Ferner wurden die Klassen `Spacer` und `CustomItem`, sowie das Interface `ItemCommandListener`. Bei `Spacer` handelt es sich um ein leeres Item, das zur Feinabstimmung des Layouts einer Form verwendet werden kann. `CustomItem` erlaubt die Implementierung von neuen, eigenen Items. Denkbar sind zum Beispiel Toggle-Buttons. Mit Hilfe des Interface `ItemCommandListener` ist es nun auch möglich Listener den einzelnen Items hinzuzufügen. Der Klasse `ChoiceGroup` wurde außerdem noch der Typ `POPUP` hinzugefügt. Eine `ChoiceGroup` des Typs `POPUP` entspricht der `ComboBox` aus J2SE.

### **RGB Bilder**

Letztendlich wurde eine Methode eingebaut, mit deren Hilfe RGB Bilder dargestellt und manipuliert werden können. Ein Bild wird als Integer Arrays repräsentiert. Pixel haben je 8bit für Rot, Grün und Blau sowie auch für den Alpha-Kanal (Transparenz).

### **Over-the-Air Provisioning**

Das MIDP 2.0 definiert außerdem wie MIDlets auf ein Gerät geladen und installiert werden können. Diesen Vorgang nennt man Provisioning. Ein Benutzer kann beispielsweise eine neue Anwendung im Internet finden. Eine Anwendung könnte sich auch selbst regelmäßig aktualisieren wollen. Zwar ist dies über USB Schnittstellen oder eine serielle Verbindung möglich, wesentlich komfortabler und einfacher wäre es aber über eine drahtlose Verbindung. Die Over-the-Air Provisioning (OTA) Spezifikation macht dies möglich.

### **Sonstige Neuerungen**

Eine „Push-Registry“ erlaubt es MIDlets zu starten, wenn externe Ereignisse auftreten. (z.B. Anruf). Außerdem können `RecordStores` nun zwischen MIDlets geteilt werden.

## **3.2 Information Module Profile (IMP)**

Beim IMP handelt es sich um eine Untermenge des MIDP 1.0. Es ist, wie der Name bereits sagt für Informationsmodule gedacht. Darunter versteht man unter anderem Module in Alarmsystemen, Notrufzellen oder Parkuhren. Solche Module können beispielsweise ein Signal senden, falls ein Getränkeautomat leer ist und wieder aufgefüllt werden muss. Das IM wird also bei Modulen eingesetzt, die in den Bereich Machine-to-Machine Kommunikation eingeordnet werden können.

Ein Information Module (IM) sollte den selben minimalen Hardware Voraussetzungen genügen, die auch schon im MIDP 1.0 festgelegt wurde. Der einzige Unterschied besteht darin, dass ein IM kein Display besitzt. Falls doch sollte dieses Display die Anwendung einer MIDP Benutzeroberfläche nicht unterstützen. Ist das der Fall darf so das IMP nicht eingesetzt werden.

Bezogen auf Datendienste verfügen diese Module über ähnliche Fähigkeiten wie Mobiltelefone, haben aber keine Bedienoberfläche im gewohnten Sinne und unterstützen keine Sprachtelefonie.

Das IMP stellt API Pakete zur Verfügung, die die grundlegende Funktionalität für Maschine-Maschine Anwendungen beinhalten. Darunter fallen Netzwerkverbindungen, dauerhafte Speicherung und die Definition eines Anwendungslebenszyklus für IM. Eine Anwendung für ein IM wird deswegen auf `IMlet` genannt. Ein `IMlet` leitet sich direkt aus der Klasse `javax.microedition.midlet.MIDlet` ab.

Eine genau Auflistung der enthaltenen Pakete und Klassen war leider nicht zu finden.

## **4 Optional Packages**

Wir haben bereits gesehen, dass mit Hilfe von Configurations und Profiles ziemlich genau auf die Möglichkeiten eines Geräts eingegangen werden kann. Manche Geräte nutzen aber sehr spezielle Technologien. Daher können sich Geräte, welche die selbe Kombination Configuration/Profile verwenden trotzdem noch in diesen Funktionen unterscheiden. Zum Beispiel gibt es Handys, die Bluetooth unterstützen, andere tun dies nicht.

Optional Packages bilden eine Schnittstelle, um auf solche Funktionen eines Gerätes zuzugreifen. Sie sind nicht in den Profiles enthalten, um einen unnötigen Speicherverbrauch zu vermeiden. Zu den bekanntesten gehören :

### **PDA Optional Package**

Es bieten einerseits Programmierschnittstellen für den Umgang mit Dateien und andererseits Grundfunktionen für die Implementierung von Applikationen wie Terminkalender und Adressbuch.

### **Bluetooth Optional Package**

Die Java APIs für Bluetooth erlauben es, eine im Endgerät vorhandenes Bluetooth-Infrastruktur über Java anzusprechen.

### **Wireless Messaging API**

Unterstützt das Versenden und Empfangen von Mitteilungen wie Kurznachrichten (SMS) und ab Version 2.0 multimediale Mitteilungen (MMS).

### **Mobile Media API**

Ist im Bereich der Medienverarbeitung angesiedelt. Mit seiner Hilfe können Audio- und Videodateien wiedergegeben und aufgezeichnet werden.

### **Webservices API**

Vereinfacht die Nutzung von entfernten Diensten über das Simple Object Access Protocol (SOAP).

### **Location API**

Ermöglicht es, den aktuellen Ort und die Bewegungsgeschwindigkeit des Endgeräts zu bestimmen.

## **5 Quellen**

- [1] Sun Seiten zu den entsprechenden Themen:  
<http://java.sun.com/j2me/>  
u.a.:
  - What's new in MIDP 2.0:  
<http://developers.sun.com/techttopics/mobility/midp/articles/midp20/>
  - Difference between Personal Basis Profile und Personal Profile  
[http://developers.sun.com/techttopics/mobility/personal/articles/pbp\\_pp/index.html](http://developers.sun.com/techttopics/mobility/personal/articles/pbp_pp/index.html)
  - Whitepapers, Datasheets, Overviews, ...
- [2] der Brontofundus: Neu in MIDP 2.0  
[http://bf.monis.ch/prog/java/midp/\\_neu\\_in\\_midp20.html](http://bf.monis.ch/prog/java/midp/_neu_in_midp20.html)
- [3] Ausarbeitung über die JavaPhone und JavaTV API von Georg Waldispühl  
<http://www.ifi.unizh.ch/~riedl/lectures/phonetv.pdf>
- [4] Kapitel 2 aus dem Buch Java 2 Mircoedition von Klaus-Dieter Schmatz  
[www.dpunkt.de/leseproben/3-89864-271-2/Kapitel\\_2.pdf](http://www.dpunkt.de/leseproben/3-89864-271-2/Kapitel_2.pdf)