Java Entwicklung für Embedded Devices Best & Worst Practices!

George Mesesan
Microdoc GmbH

Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Natürlich können wir dieses neue log4j Bundle auch auf dem Device verwenden. Ist doch alles Java."

- Java Micro Edition (ME) ≠ Java Standard Edition
- Die Kompatibilität zu Java ME muss geprüft werden:
  - Minimun Execution Environment wurde im MANIFEST konfiguriert.
  - Der Sourcecode ist verfügbar und lässt sich mit einem Java ME Compiler fehlerfrei übersetzen.
- Ist das Bundle nicht kompatibel so kann eine Portierung vorgenommen werden.
- Ein nicht-kompatibles Bundle sollte auf keinen Fall eingesetzt werden nach dem Motto: *"…aber es läuft ja eh!".* Die böse Überraschung tritt meist erst im Feld auf.

Java-Entwicklung für Embedded Devices

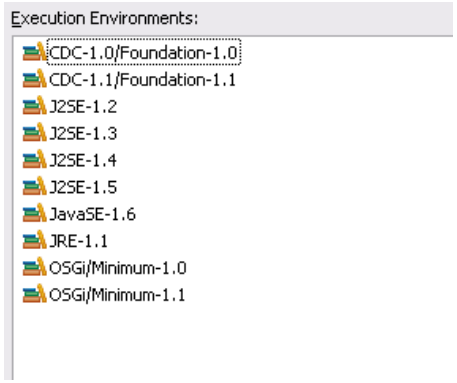
Best & Worst Practices

"Wir wissen daß Java ME nicht gleich Java SE ist. Deshalb haben wir auch den Source Level von unserem Java 5 Compiler auf 1.3 gesetzt."

- Dies reicht leider nicht aus.
- Das Problem ist dass der Code gegen die Java 5 Class Library gelinkt wird:

```
java.lang.NoSuchMethodError: java.lang.StringBuffer: method
append(Ljava/lang/StringBuffer;)Ljava/lang/StringBuffer;
not found
```

- Es muss entweder ein reiner Java ME Compiler verwendet oder ein JRE Execution Environment konfiguriert werden.

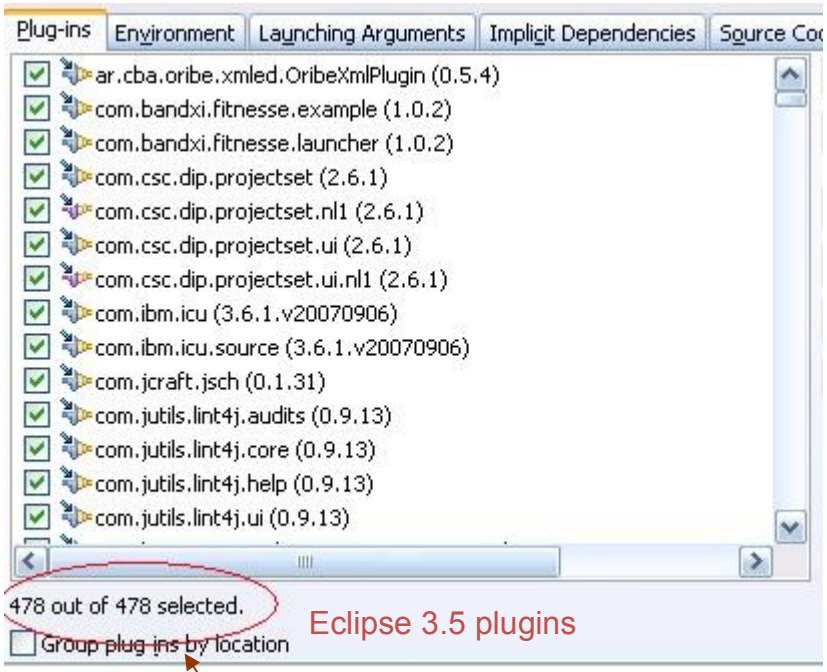


Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Target Platform? Noch nie gehört! "

- Unter Target Platform versteht man alle Bundles die man für seine Anwendung benötigt (Framework Bundles, …)
- Als Standard wird in Eclipse die Target Platform der IDE gesetzt.
- Um einen konsistenten Stand der verwendeten Bundles zur Entwicklung und Laufzeit zu haben muss die Target Platform konfiguriert werden.



auf meinem device ?

Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Wir haben die Target Platform richtig konfiguriert und können also auf unserem Entwicklungsrechner sehr gut testen. Auf dem Device zu testen machen wir meist eine Woche vor der Lieferung."

- Kontinuierliches Testen am Device ist unverzichtbar um frühzeitig Probleme zu erkennen:
  - Performace-Engpässe
    - OSGi Startup Zeiten wär hier ein Problem das immer wieder auftritt.
  - Speicher-Probleme
    - Windows Mobile stellt einer Anwendung nur 32MB zur Verfügung. DLLs müssen manchmal schon sehr früh im Startprozess der OSGi Anwendung geladen werden.
  - Filesystem & Pfade können Probleme verursachen
  - Bugs von 3rd Party Code: eSWT Fullscreen auf Windows Mobile 2003
  - u.v.m

"Wir brauchen keine Simulatoren. Wir testen immer gleich am Device."

- Simulatoren beschleuningen den Turnaround-Zyklus "code-deploy-test-debug" und tragen deshalb wesentlich zur Qualität bei.
- In OSGi sind Simulatoren relativ einfach zu implementieren.
- Bundles mit Hardware-Zugriff können einfach durch Bundles mit simuliertem Zugriff ersetzt werden.
- Simulatoren ersetzen jedoch nicht das Testen am Device.

Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Testen ist nicht (mehr) so wichtig. Wenn wir einen Bug entdecken, machen wir einfach ein Remote Update. Mit OSGi geht das ja ganz einfach."

- Ein Remote Update ist ein komplexer Prozess mit technischen und ökonomischen Anforderungen:
  - Neue Updates können das Device in einen Zustand bringen wo kein weiteres Update mehr möglich ist.
  - Ein Upate vieler Geräte die über GSM/GPRS ihr Update beziehen kann zum Kostenfaktor werden. Die Kosten können sogar explodieren wenn Roaming Kosten hinzukommen.
- Je weniger Bugfix-Updates also notwendig sind, umso besser.

"Bundle-Versionierung brauchen wir nicht. Wir machen einfach ein Update der ganzen Anwendung."

- In Szenarien wo, teilweise hohe, Verbindungskosten (GSM) anfallen, ist es ratsam das Update so klein wie möglich zu halten.
- Bundle-Versionierung ermöglicht es nur Bundles auszutauschen, welche sich auch wirklich geändert haben.

Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Zum testen unseres JNI Codes starten wir einfach die Anwendung hoch und testen damit."

- Generell sollte Anwendungs- und Frameworkcode nicht gemischt werden → POJOs
- POJOs zu verwenden ist insbesondere bei JNI Teilen wichtig
  - Spart Zeit beim Testen
  - Spart Zeit bei der Suche nach Fehlern
  - Erhöht die Testbarkeit

Java-Entwicklung für Embedded Devices

Best & Worst Practices

"Wir testen am Device immer manuell. Automatisiertes Testen am Device ist entweder nicht möglich oder nur mit großen Aufwand implementierbar."

- Mittlerweile gibt es Testing Frameworks dies es ermöglichen relativ einfach Testcases (JUnit, …) auf dem Device auszuführen.
- eFitNesse: http://www.microdoc.com/efitnesse