

Gerd Wütherich ♦ Nils Hartmann ♦ Bernd Kolb ♦ Matthias Lübken

Einführung in die OSGi Service Platform

Wer sind wir?

- » Gerd Wütherich – gerd@gerd-wuetherich.de
- » Nils Hartmann – nils@nilshartmann.net
- » Bernd Kolb – bernd.kolb@sap.com
- » Matthias Lübken – matthias.luebken@it-agile.de

Die OSGi Service Platform – Das Buch



- » Detaillierte Einführung in OSGi-Technologie
- » April 2008, dpunkt.verlag
- » ISBN 978-3-89864-457-0
- » Website: www.osgibook.org

Agenda

- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogramm-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

Agenda

- » **OSGi-Technologie im Überblick**
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogram-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

OSG – was?

- » Die OSGi Service Plattform...
 - » ... ist ein *dynamisches Modulsystem für Java*.
 - » ... ermöglicht die dynamische Integration und das Management von Softwarekomponenten (*Bundles*) und Diensten (*Services*).
- » Bundles und Services können zur Laufzeit in der Plattform *installiert, gestartet, gestoppt* und *deinstalliert* werden.
- » Besteht aus:
 - » OSGi Framework (Container für Bundles und Services)
 - » OSGi Standard Services (verschiedene, horizontale Services)

Woher kommt die OSGi Service Platform?

Die OSGi Alliance:

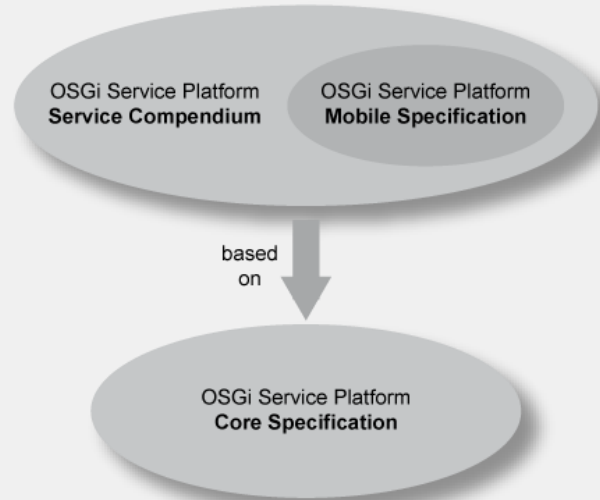
- » <http://www.osgi.org>
- » Zusammenschluß renommierter Unternehmen
- » Gegründet 1999
- » Derzeit mehr als 80 Mitglieder:
 - » Deutsche Telekom, Eclipse Foundation, IBM, Oracle, Prosyst, SAP, Siemens, Sun, SpringSource etc.
- » Spezifiziert und fördert die OSGi Service Platform

Wo wird die OSGi Service Platform eingesetzt?

Einige Beispiele:

- » Eclipse Platform:
 - » Eclipse SDKs (IDEs), RCP, eRCP, ...
- » IBM
 - » Websphere App Server (basiert auf OSGi)
 - » Lotus (basiert auf Eclipse-RCP, damit auch OSGi)
 - » Jazz (basiert auf Server-Side-Eclipse)
- » BEA/Oracle
- » SpringSource Application Platform / dm Server
- » Adobe
- » ...

Wo ist die OSGi Plattform spezifiziert?



- » Spezifikationen verfügbar unter <http://www.osgi.org>:
 - » Die Core Specification spezifiziert das OSGi Framework
 - » Das Service Compendium spezifiziert diverse Standard Services, z.B. einen Log Service oder einen Preferences Services.
 - » Die Mobile Specification definiert verschiedene Standard Services für mobile Applikationen.

Welche Vorteile bietet die OSGi Service Platform?

- » Modularisierung und Versionierung
- » Abhängigkeitsmanagement (zur Laufzeit)
- » Hot deployment
- » (Fern-) Management des laufenden Frameworks über sog. Management Agents
- » Serviceorientiertes Programmiermodell

SOA-Konfusion:

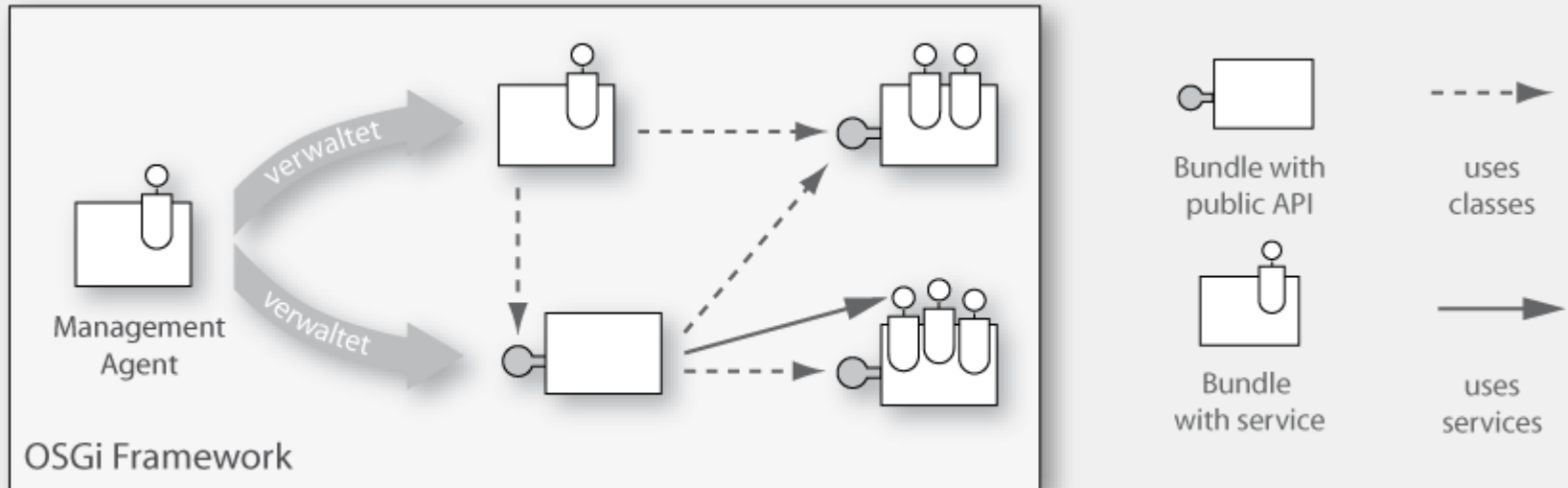
- » Web Services publizieren und nutzen Dienste über das Netz
- » OSGi Services publizieren und nutzen Dienste innerhalb einer virtuellen Maschine

Implementierungen der OSGi Service Plattform

- » Open Source Implementierungen:
 - » Eclipse Equinox (<http://www.eclipse.org/equinox/>)
 - » Apache Felix (<http://cwiki.apache.org/FELIX/index.html>)
 - » Knopflerfish (<http://www.knopflerfish.org/>)
 - » ProSyst mBedded Server Equinox Edition ([http://www.prosyst.com/products/osgi se equi ed.html](http://www.prosyst.com/products/osgi%20se%20equinox%20edition.html))

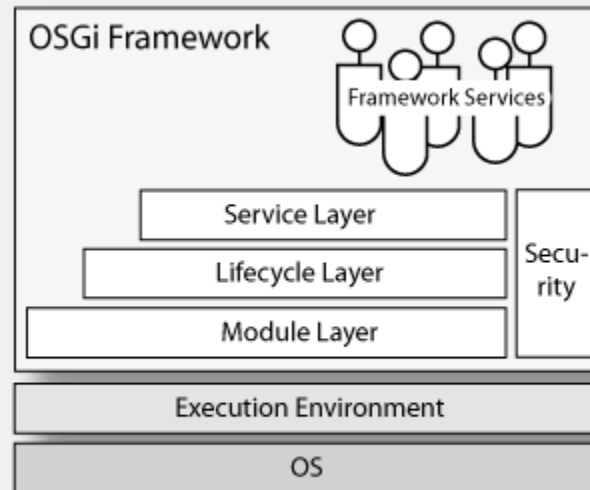
- » Kommerzielle Implementierungen:
 - » ProSyst (<http://www.prosyst.com/>)
 - » Knopflerfish Pro (<http://www.gatespacetelematics.com/>)
 - » ...

Das OSGi Framework



- » Basiskomponente der OSGi Service Platform
- » Erlaubt die Installation und Verwaltung von Bundles und Services
- » Verwaltet Abhängigkeiten zwischen Bundles
- » Kann über Management Agents „von außen“ administriert werden

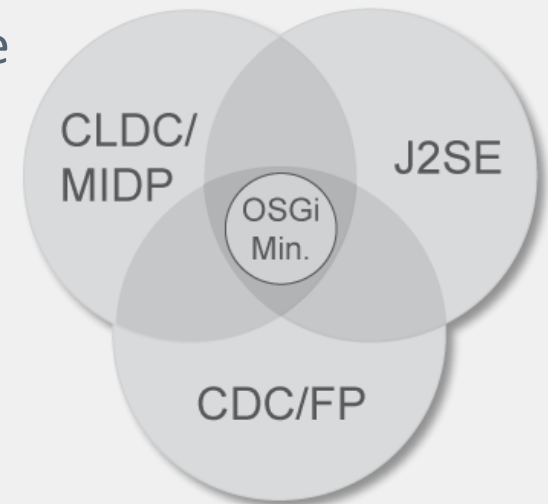
OSGi Framework: Schichten



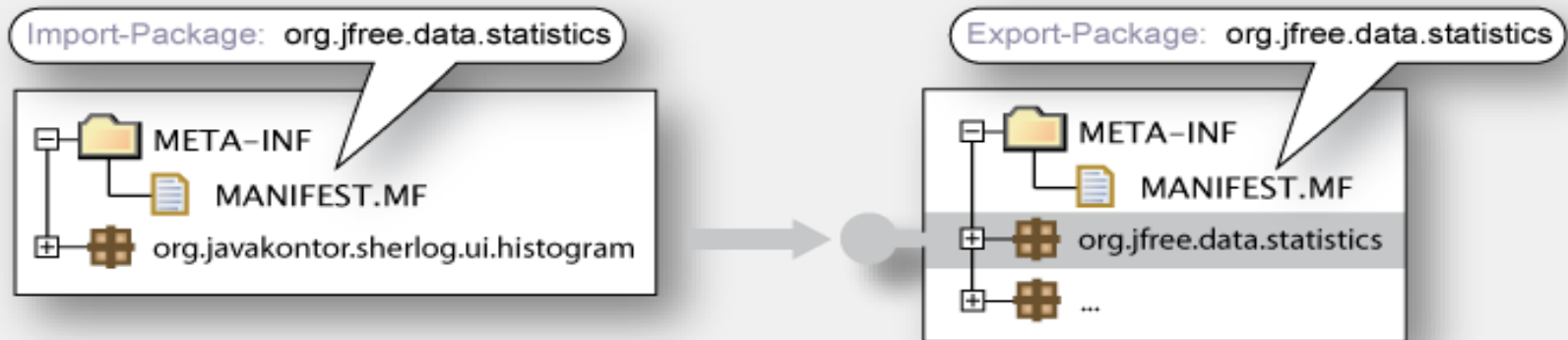
- » Das OSGi Framework ist in mehreren logischen Schichten definiert:
 - » Execution Environments
 - » Module Layer
 - » Lifecycle Layer
 - » Service Layer
 - » Security Layer (optional)

OSGi Framework: Execution Environments

- » Execution Environments ...
 - » ... abstrahieren von konkreten JREs
 - » ... definieren Klassen, Interfaces und Signaturen
- » OSGi spezifiziert zwei Execution Environments:
 - » OSGi/Minimum-1.1: Minimale Ausführungsumgebung für das OSGi Framework
 - » CDC-1.0/Foundation: JME Foundation Profile
- » Weitere Execution Environments:
 - » J2SE-1.5
 - » JavaSE-1.6
 - » ...

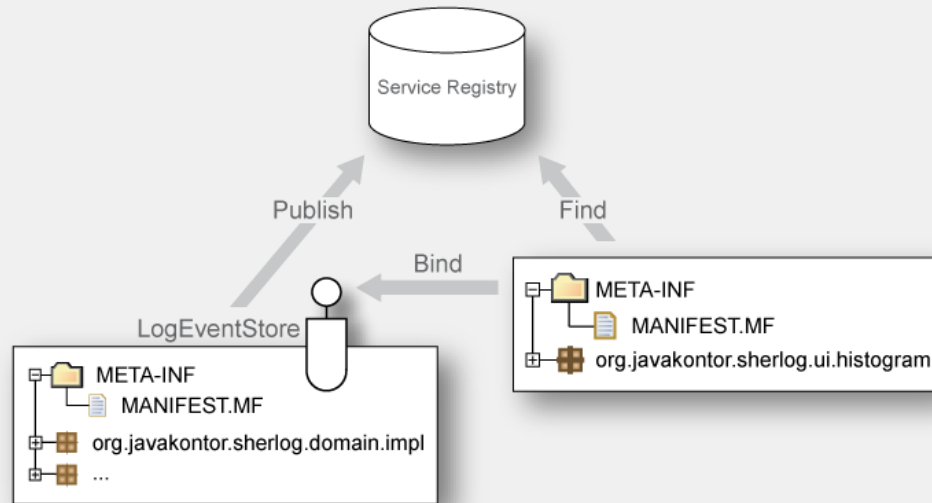


OSGi Framework: Module Layer



- » Der *Module Layer* definiert das Bundle als grundlegende Modularisierungseinheit.
- » Das Framework erlaubt die Definition von
 - » Sichtbarkeiten von Modul-Bestandteilen (public-API vs. private-API)
 - » Abhängigkeiten zwischen Modulen, sowie
 - » Versionen von Modulen

OSGi Framework: Service Layer



» Der Service Layer ...

- » ...definiert, wie Services zur Laufzeit über die Service-Registry veröffentlichen und wieder entfernen werden können.
- » ...definiert, wie Bundles über die Service-Registry Services finden und verwenden können.

OSGi Framework: Security Layer

- » Der Security Layer ...
 - » ... ist ein optionaler Layer
 - » ... basiert auf der Java 2 Security Architektur
 - » Permissions
 - » Bundle signing
 - » Dynamisch

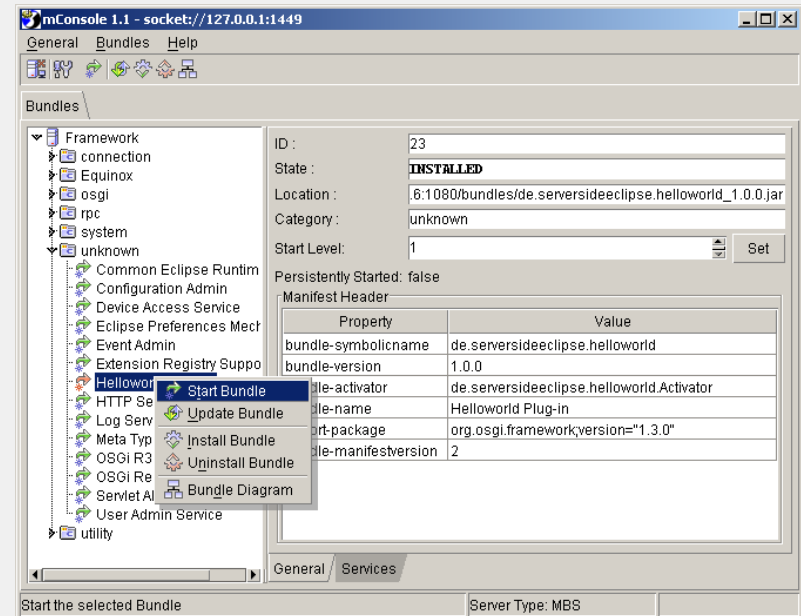
OSGi Framework: Management Agents I

- » Management Agents ermöglichen die Administration eines OSGi Frameworks.
- » Breites Spektrum verfügbar:
 - » Kommandobasierte Konsole (Equinox Konsole)
 - » Grafisch-interaktive Anwendungen (Knopflerfish Desktop, Prosyst mConsole)
 - » Web-basierte Oberfläche (Knopflerfish Web-Konsole)
- » Nicht standardisiert, aber Zugriff auf das Framework über definierte Schnittstellen.

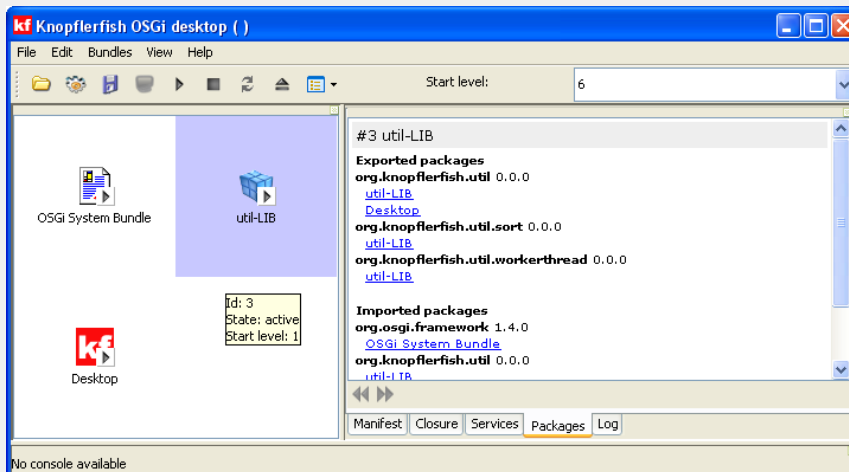
OSGi Framework: Management Agents II

```
C:\WINDOWS\system32\cmd.exe - java -jar plugins/org.eclipse.osgi_3.3.0.v20070530.jar -console
C:\equinox>java -jar plugins/org.eclipse.osgi_3.3.0.v20070530.jar -console
osgi> ss
Framework is launched.
id      State      Bundle
0       ACTIVE    org.eclipse.osgi_3.3.0.v20070530
osgi> install file:/c:/bundles/de.serversideeclipse.helloworld_1.0.0.jar
Bundle id is 1
osgi> ss
Framework is launched.
id      State      Bundle
0       ACTIVE    org.eclipse.osgi_3.3.0.v20070530
1       INSTALLED de.serversideeclipse.helloworld_1.0.0
osgi> start 1
```

Eclipse Equinox Console

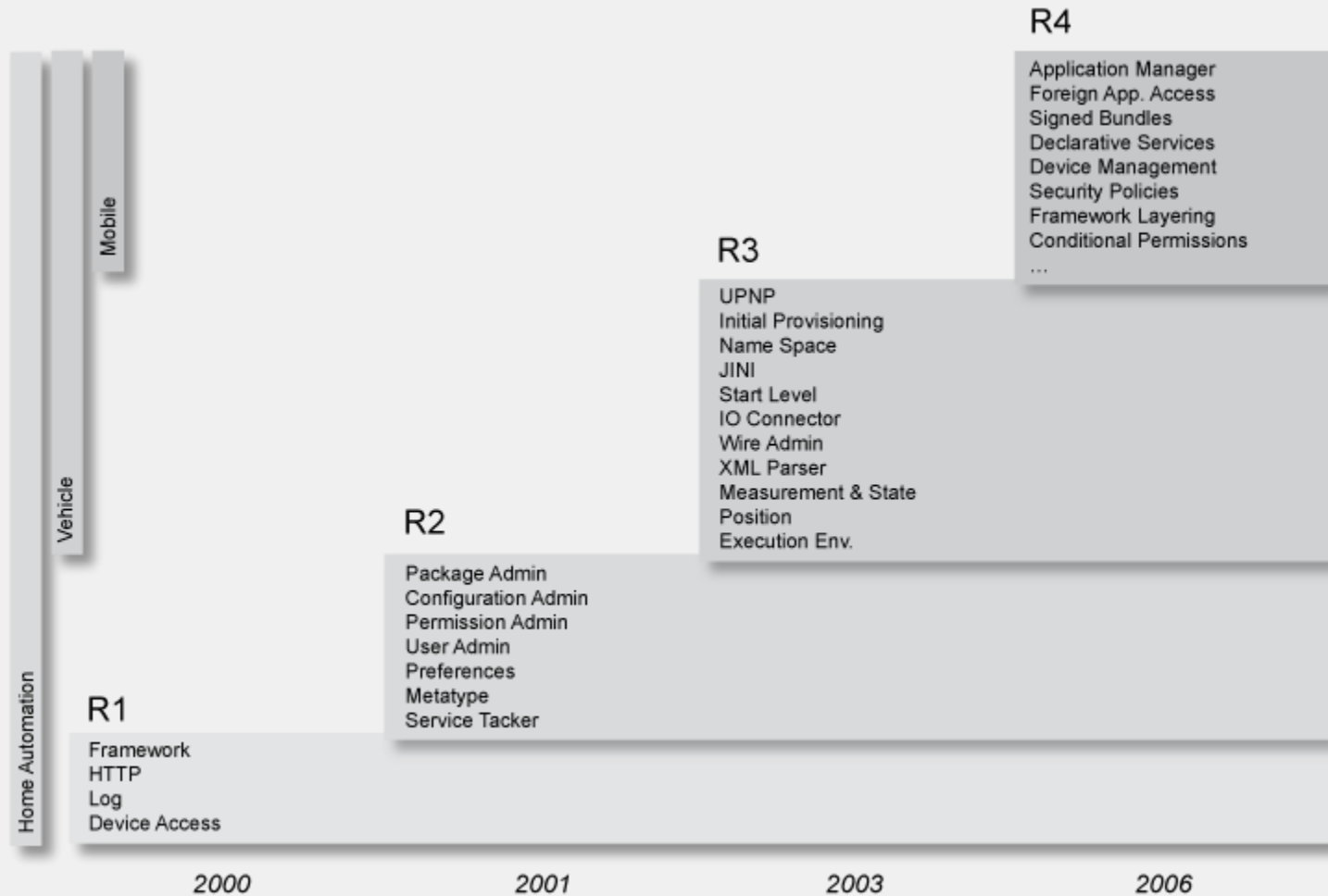


Prosyst mConsole

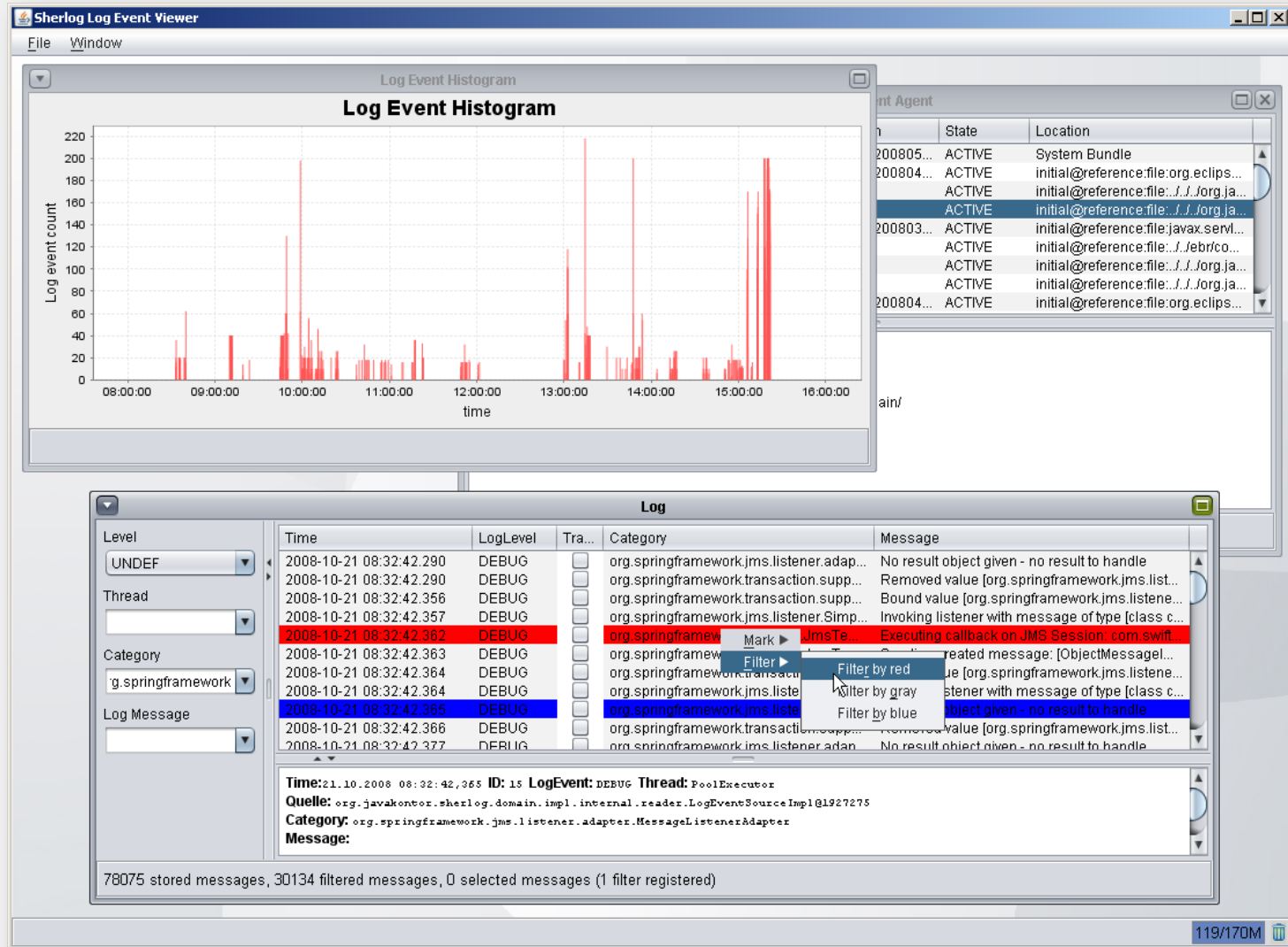


Knopflerfish Desktop

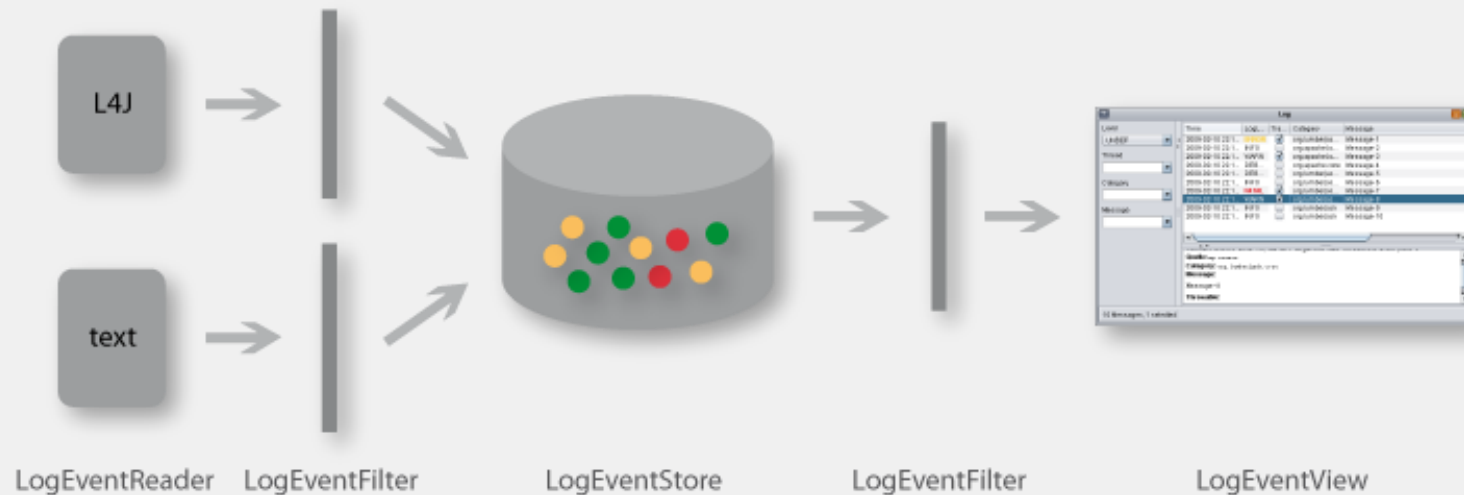
OSGi Standard Services



OSGi-Demo: Sherlog - Log-File-Analyser

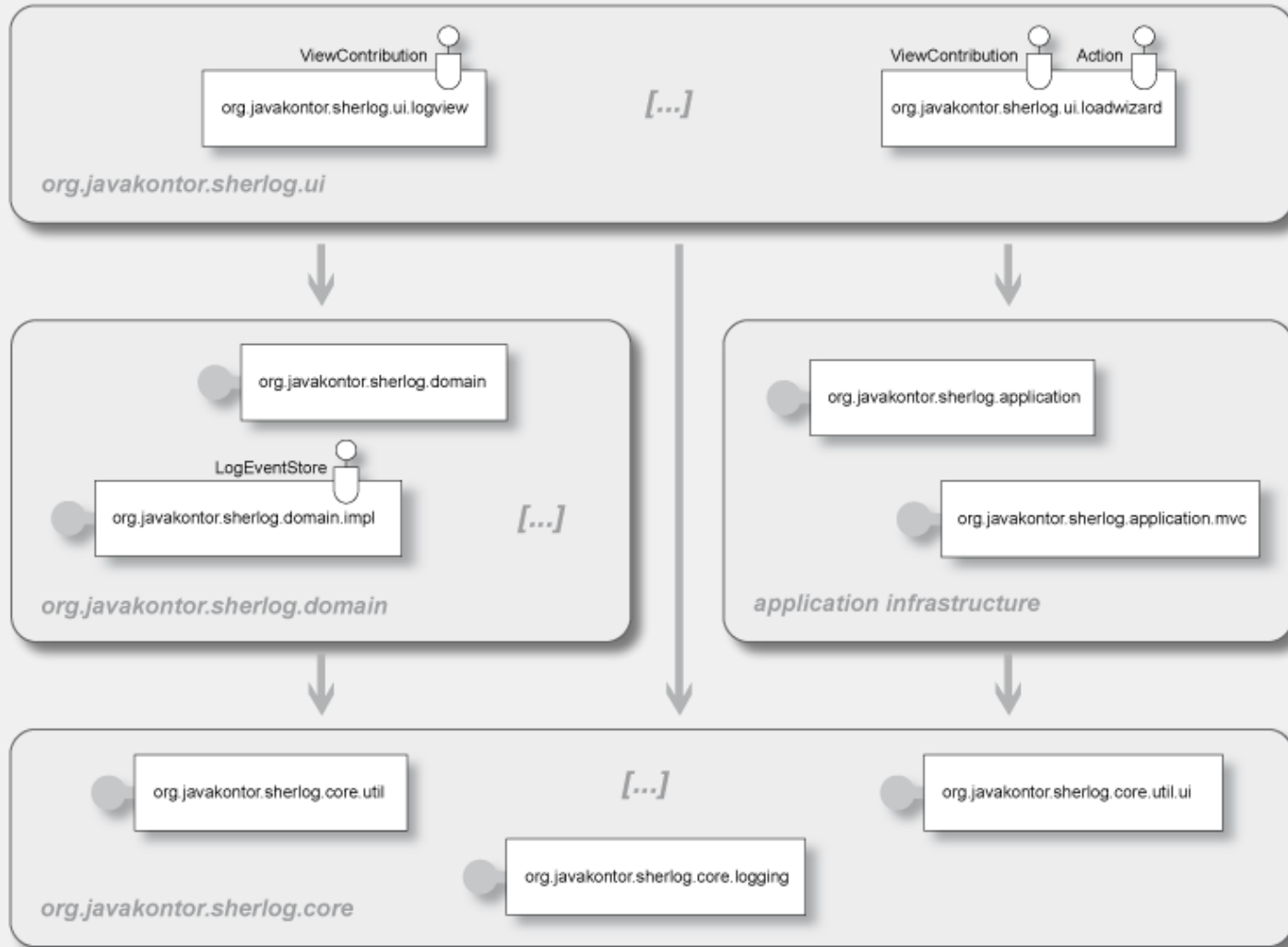


Sherlog-Domänenmodell

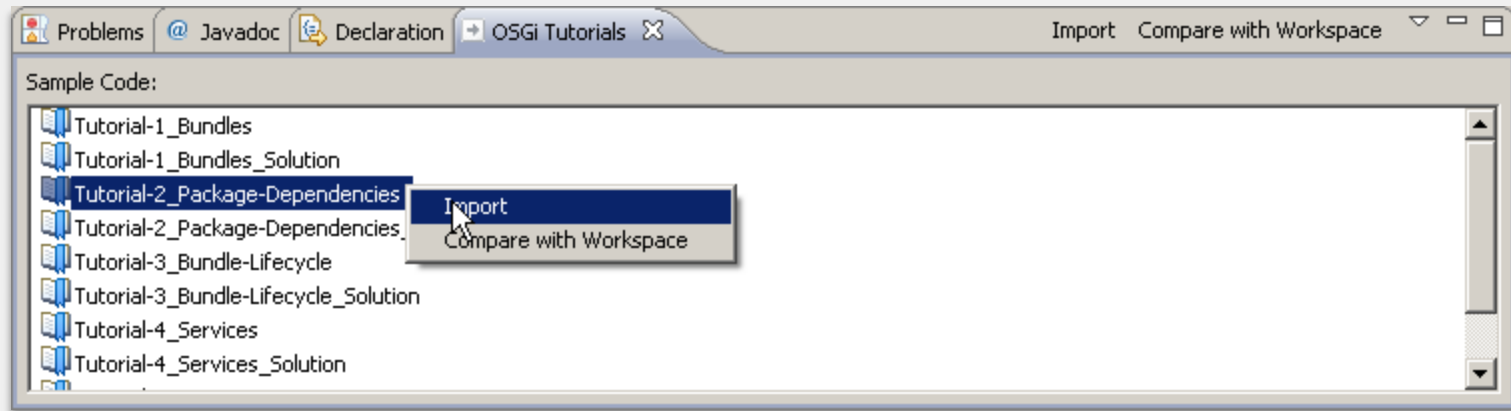


LogEvent	Kapselt einen Log-Eintrag
LogEventStore	Enthält alle geladenen Log Events
LogEventReader	Liest Log Events aus einer Quelle (bspw. Log-Datei lokal oder remote, L4J-SocketAppender)
LogEventFilter	Filtert Log Events nach bestimmten Eigenschaften
LogEventView	Visualisiert die Log Events aus dem Log Event Store

Sherlog-Systemarchitektur



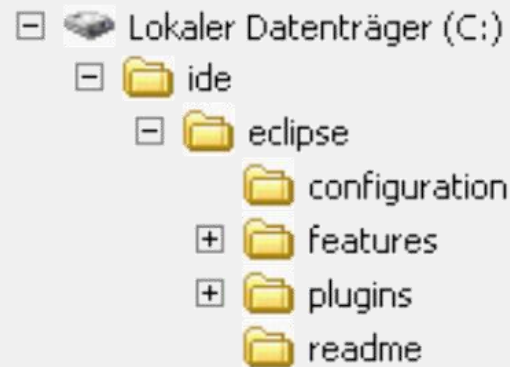
Tutorial 0: Installation



Aufgabe:

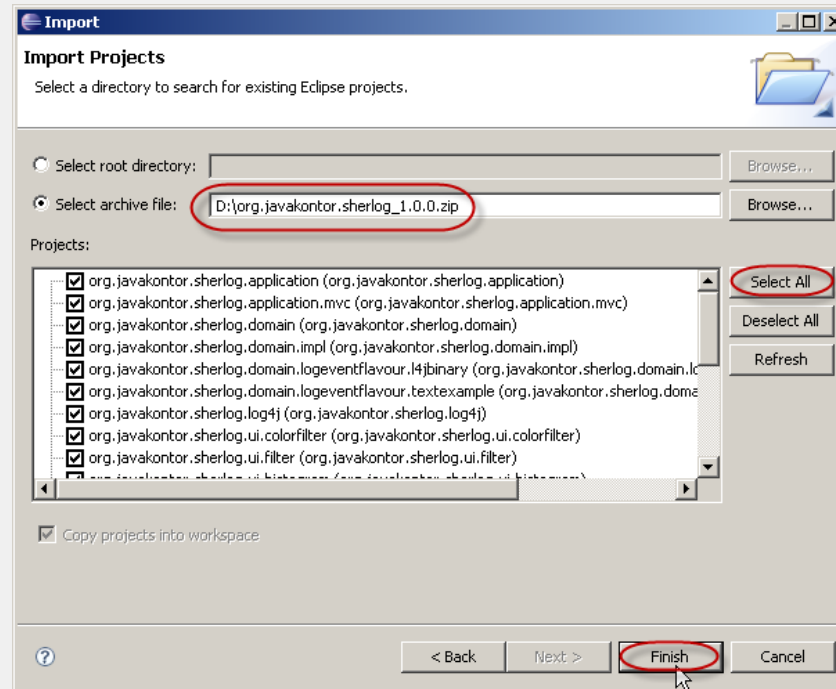
- » Installieren Sie die Eclipse IDE (classic)
- » Importieren Sie die Sherlog-Projekte
- » Richten Sie den Sample Manager mit den Beispielen ein

0.1: Die Eclipse IDE installieren



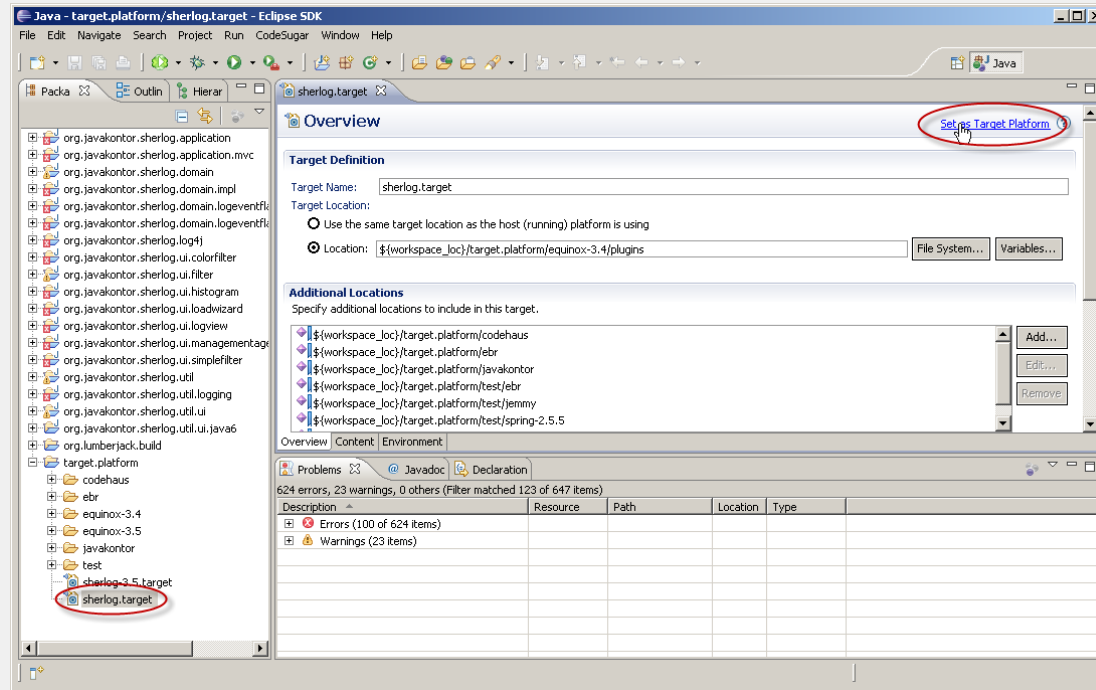
- » Wir nutzen Eclipse SDK Classic, Version 3.4.2
 - » z.B. eclipse-SDK-3.4.2-win.zip
- » Installation durch Entpacken:
 - » z.B. c:\ide\
 - » Eclipse starten und neuen Workspace anlegen

0.2: Die Sherlog-Projekte importieren



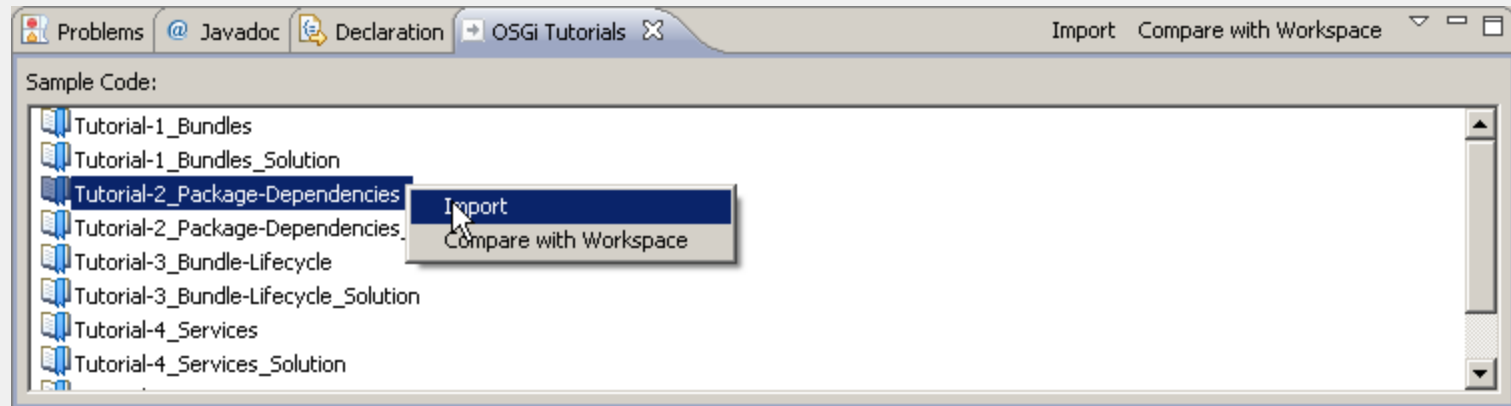
» Importieren Sie die Sherlog-Projekte aus der Datei
“org.javakontor.sherlog_1.0.0.zip”

0.3 Die PDE Target Platform einrichten



- » Öffnen Sie die Datei „target.platform/sherlog.target“
- » Im Target Editor können Sie die Konfiguration als Target Platform setzen

0.4 Beispiele installieren

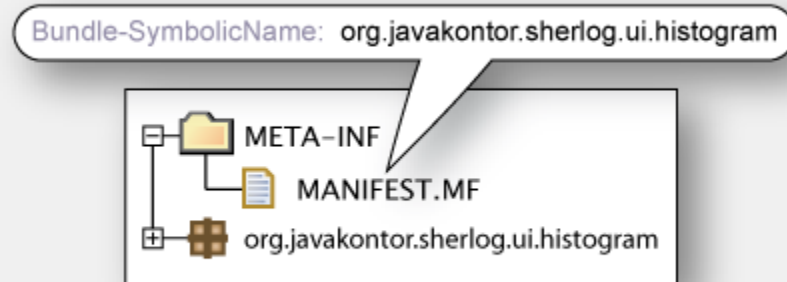


- » Alle Beispiele stehen innerhalb des Sample Managers bereit (Datei „osgi_examples.zip“)
- » Installation über „Help ⇒ Software updates ⇒ Add Site... ⇒ Archive...“
- » Aufruf über “Window ⇒ Show view ⇒ Other... ⇒ Other ⇒ OSGi Tutorials”

Agenda

- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » **Bundles**
 - » *Tutorial: Ein Histogramm-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

Bundles

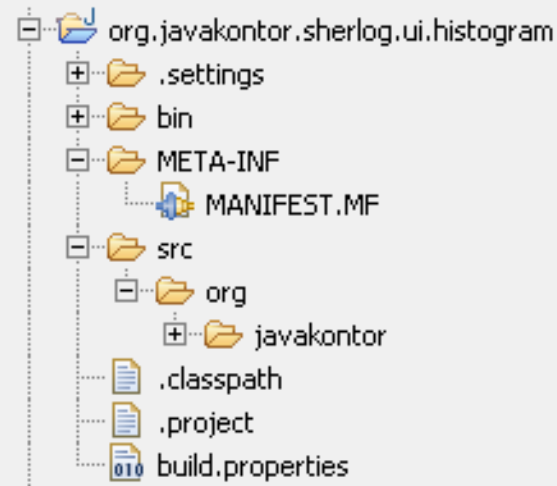


- » ... sind die Modularisierungseinheiten innerhalb des OSGi Frameworks
- » ... enthalten zusammengehörige Klassen und Ressourcen
- » ... können unabhängig im OSGi Framework deployed werden
- » ... sind JAR-Dateien
- » ... enthalten ein Bundle Manifest, das das Bundle beschreibt

Eclipse Plug-in Projekte

Eclipse Plug-in Projekte...

- » ... werden vom Plug-in Development Environment (PDE) bereit gestellt
- » ... können für reguläre Bundles und für Eclipse Plug-ins verwendet werden
- » ... bieten OSGi-Unterstützung
 - » “New Plug-in Project” Wizard
 - » Bundle Manifest Editor
 - » OSGi run/launch configuration
 - » Target platform
 - » Export wizard
 - » ...
- » ... können direkt in Eclipse Equinox installiert werden



Das Bundle Manifest

- » ... ist Teil des Bundles in der Datei **META-INF/MANIFEST.MF**
- » ... enthält Informationen, die das Bundle beschreiben, z.B.
 - » den eindeutigen Namen und die Version
 - » die öffentliche Schnittstelle/API
 - » Package-Abhängigkeiten

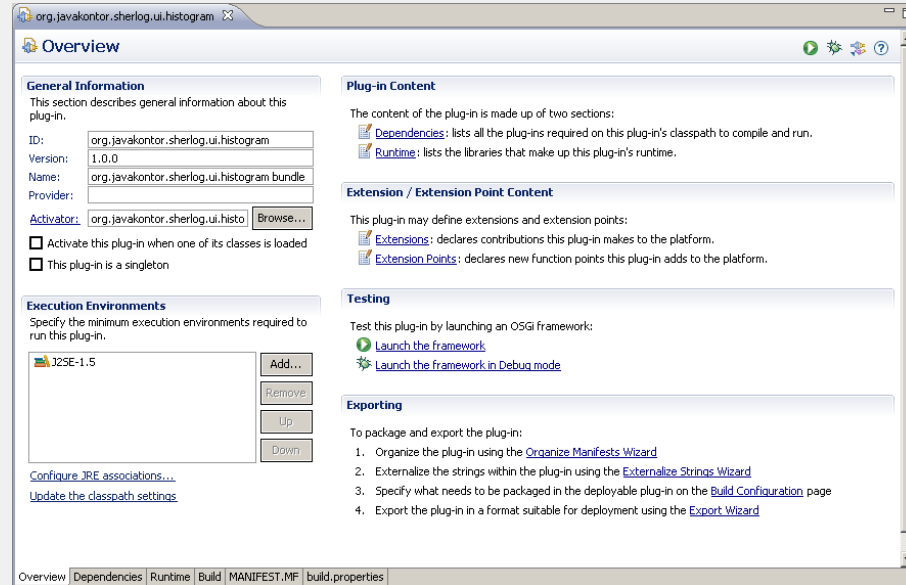
```
Manifest-Version: 1.0  
Bundle-ManifestVersion: 2  
Bundle-Name: Histogram Bundle  
Bundle-SymbolicName: org.javakontor.sherlog.ui.histogram  
Bundle-Version: 1.0.0  
Bundle-Activator: org.javakontor.sherlog.ui.histogram.Activator  
Import-Package: org.osgi.framework;version="1.4.0"  
Bundle-ClassPath: .
```

Wichtige Manifest Header

Manifest Header	Bedeutung
Bundle-SymbolicName *	Eindeutiger Name innerhalb des OSGi-Frameworks; Konvention: umgekehrter Domain-Name
Bundle-Version	“major.minor.micro.qualifier”, z.B. “1.2.3.test”; Default: “0.0.0”
Bundle-Classpath	Alle Jars/Verzeichnisse innerhalb des Bundle-Jars, die den Klassenpfad bilden. Default “.”
Bundle-Activator	Der <i>BundleActivator</i> wird bei der (De)Aktivierung des Bundles ausgeführt
Import-Package	Eine Liste aller benötigten Pakete
Export-Package	Eine Liste aller Pakete, die API bilden, und für andere Bundles sichtbar sind

* erforderlich

Der Eclipse Plug-in Manifest Editor



- » Bestandteil des Plug-in Development Environment (PDE)
- » Ermöglicht die interaktive Bearbeitung des Bundle-Manifestes
- » Syntax-Prüfung und Code-Completion für Manifest-Datei

Bundle Aktivierung (1)

- » Jedes Bundle kann im Bundle-Manifest einen Bundle-Aktivator definieren

```
Bundle-Activator: org.javakontor. ... .Activator
```

- » Der Bundle-Aktivator muss das Interface *BundleActivator* implementieren

```
package org.osgi.framework;  
  
public interface BundleActivator {  
    public void start(BundleContext context) throws Exception;  
    public void stop(BundleContext context) throws Exception;  
}
```

Bundle Aktivierung (2)

- » Das OSGi Framework erzeugt genau eine Instanz pro Bundle
- » Wenn das Bundle gestartet wird, wird die *start()*-Methode aufgerufen
- » Wenn das Bundle gestoppt wird, wird die *stop()*-Method aufgerufen

Best practice:

- » Da *start()* und *stop()* auf dem Framework-Thread ausgeführt werden, sollte für langdauernde Aktionen ein eigener Thread gestartet werden.

Der interne Bundle Klassenpfad

- » ... wird mit dem Manifest Header “Bundle-Classpath” spezifiziert
- » Die Klassenpfad-Einträge werden durch Komma separiert:
 - » “.” steht für das Root-Verzeichnis des Bundles (der Default-Klassenpfad)
 - » weitere Einträge können Verzeichnisse oder JAR-Dateien innerhalb des Bundles sein

```
Bundle-Classpath: ., lib/slf4j-api-1.4.3.jar
```

- » Alle Einträge sind Bundle-intern und relativ zum Root-Verzeichnis des Bundles

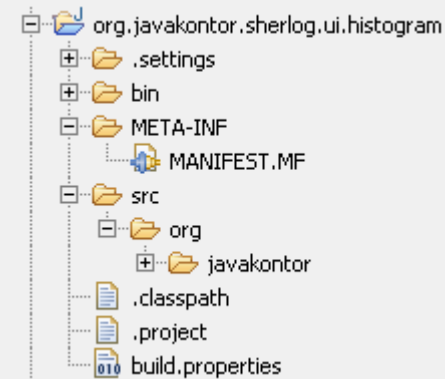
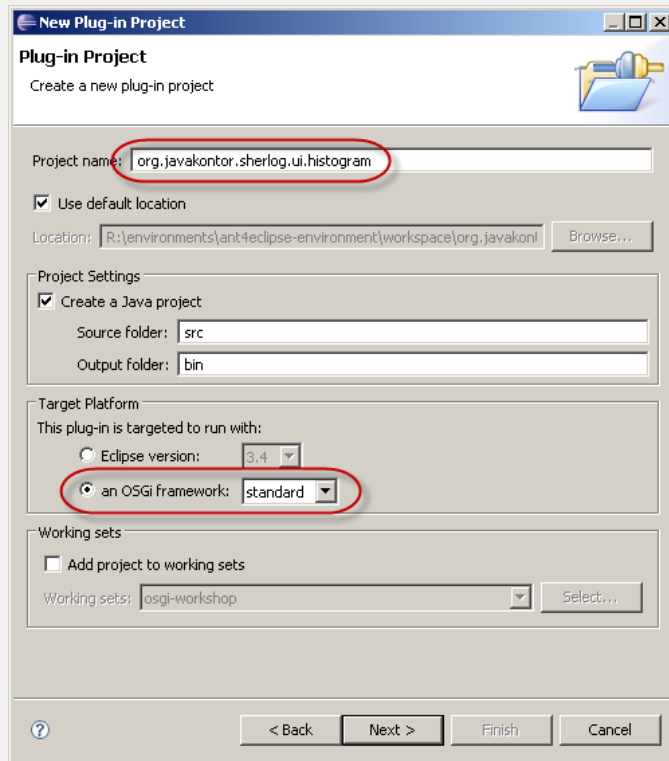
Tutorial 1: Ein „Histogram“-Bundle



Aufgabe:

- » Legen Sie ein neues Bundle `org.javakontor.sherlog.ui.histogram` an
- » Bei Start des Bundles soll ein JFrame mit einem Label „Log Event Histogram“ erzeugt werden
- » Beim Stopp des Bundles soll das JFrame wieder geschlossen werden

1.1 Anlegen des Projektes



- » Anlegen eines neuen Plug-in-Projektes in Eclipse:
 - » File -> New -> Project ... -> Plug-in Project

1.2 Implementierung der Aktivator-Klasse

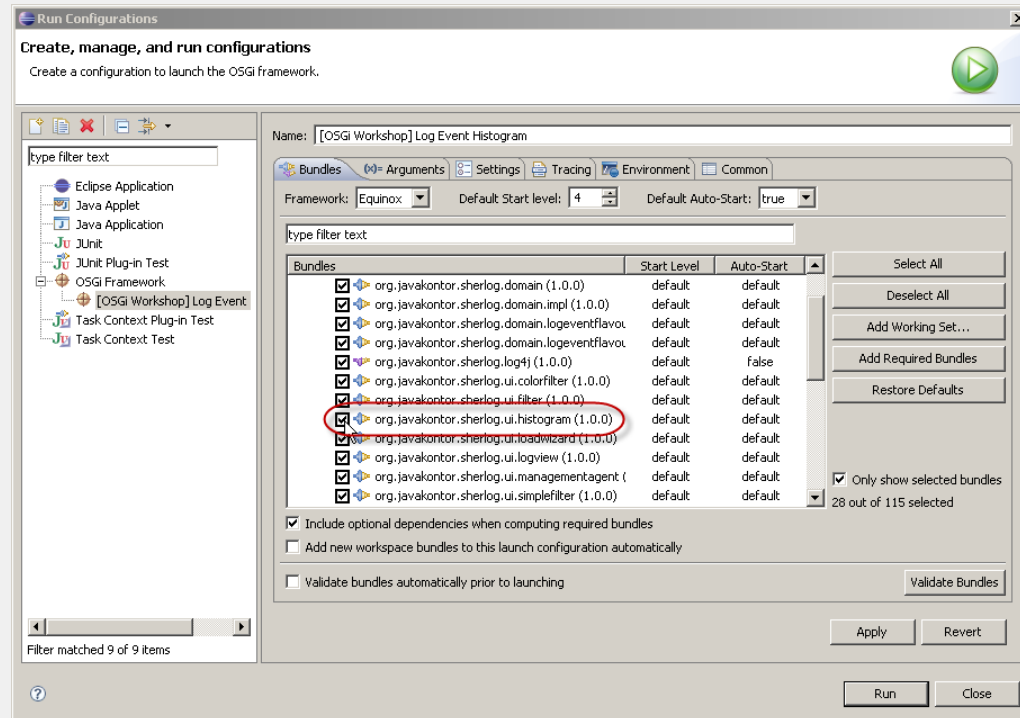
```
public class Activator implements BundleActivator {

    private JFrame _frame;

    public void start(BundleContext context) throws Exception {
        _frame = new JFrame("Log Event Histogram");
        JLabel label = new JLabel("Log Event Histogram", SwingConstants.CENTER);
        label.setPreferredSize(new Dimension(200, 100));
        _frame.add(label);
        _frame.pack();
        _frame.setVisible(true);
    }

    public void stop(BundleContext context) throws Exception {
        _frame.setVisible(false);
        _frame.dispose();
        _frame = null;
    }
}
```

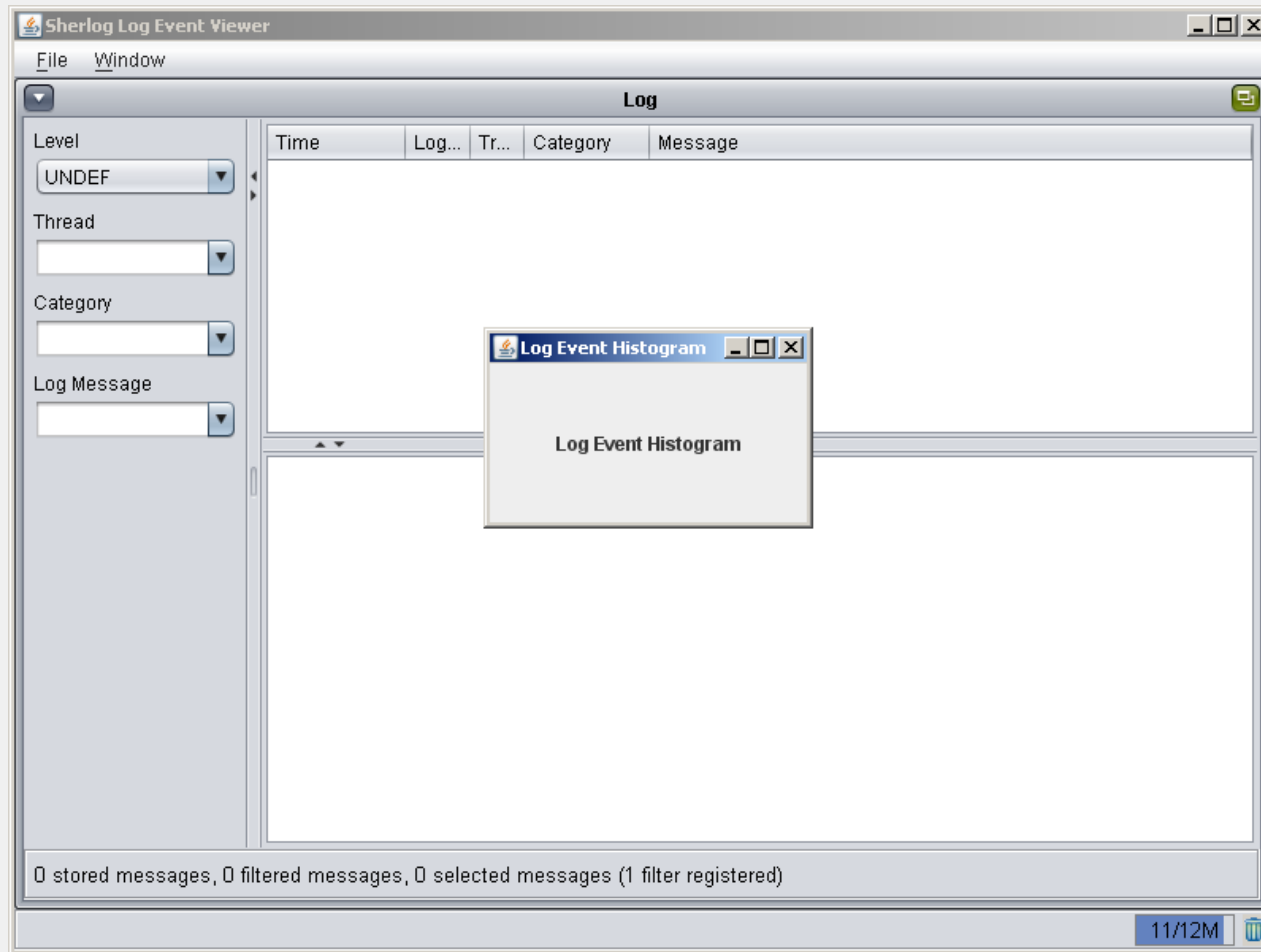
1.3 Starten der OSGi Plattform



Anpassen der Launch-Konfiguration:

» Run -> Run Configurations...

1.4 Ausführen den Beispiels



Agenda

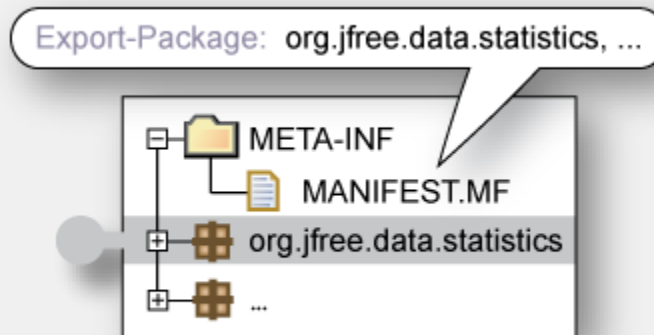
- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogram-Bundle*
- » **Package-Abhängigkeiten zwischen Bundles**
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

Package-Abhängigkeiten



- » Package-Abhängigkeiten müssen explizit angegeben werden:
 - » Packages müssen **exportiert** werden, um sichtbar für andere Bundles zu sein.
 - » Packages müssen **importiert** werden, um im Bundle genutzt werden zu können.
- » Das OSGi Framework ist verantwortlich für das Auflösen der importierten und exportierten Packages.

Packages exportieren



- » Nur die öffentliche API ist für andere Bundles sichtbar
- » Die öffentliche API muss im Bundle Manifest explizit gekennzeichnet werden
- » Manifest-Header “Export-Package”: kommaseparierte Auflistung aller exportierten Packages

```
Export-Package: org.jfree.data.statistics, ...
```

Packages importieren

- » Zunächst kann ein Bundle nur Klassen seines “Bundle-Classpath” sehen
- » Um die öffentliche API anderer Bundles zu nutzen, müssen diese Abhängigkeiten explizit angegeben werden
- » Zwei Möglichkeiten, Abhängigkeiten anzugeben:
 - » Importieren von Packages (Import-Package)
 - » Abhängigkeit auf ganze Bundles (Require-Bundle)

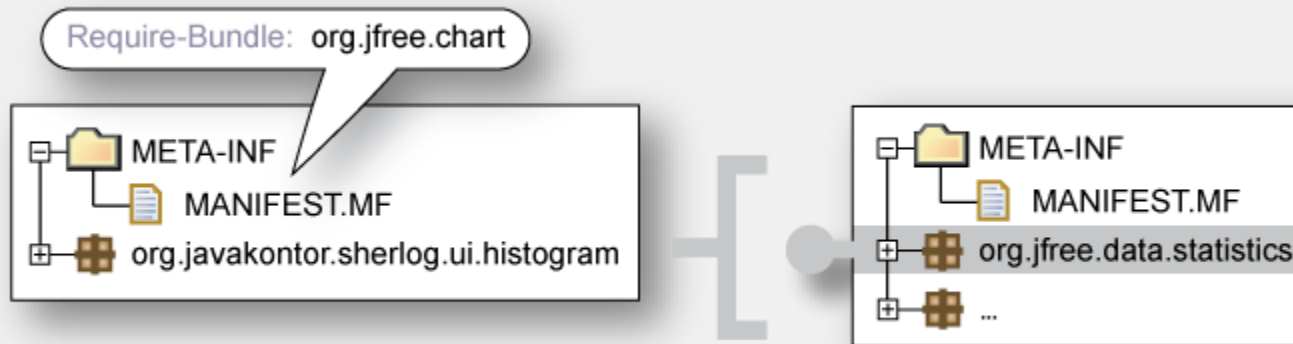
Importieren von Packages mit Import-Package



- » Der Manifest-Header “Import-Package” gibt Package-Abhängigkeiten an
 - » Importierte Packages werden durch Komma getrennt aufgelistet

```
Import-Package: org.jfree.data.statistics
```


Packages importieren mit Require-Bundle



- » Referenziert ein bestimmtes Bundle über dessen symbolischen Namen
- » Bundles werden durch Komma getrennt aufgeführt

```
Require-Bundle: org.free.chart
```

- » *Alle* exportierten Packages der benötigten Bundles werden vom Bundle importiert
- » Die Packages aller re-exportierten Bundles der benötigten Bundles werden ebenfalls automatisch importiert !

Import-Package oder Require-Bundle

- » Import-Package ursprünglicher Ansatz, Abhängigkeiten auszudrücken
- » Require-Bundle hat Nachteile:
 - » Abhängigkeiten auf ganze Bundles, nicht auf Packages
 - » Import nicht benötigter Packages
 - » “Split-Packages”: zwei Bundles exportieren dasselbe Package...

Empfehlung:
Verwenden Sie Import-Package statt Require-Bundle!

Versionen

- » Bundles und Packages lassen sich versionieren
- » Versionen:
 - » major.minor.micro.qualifier, z.B. 1.0.3.20081025NB
 - » Default: 0.0.0
- » Beispiel: Versionierung von Bundles

```
Bundle-SymbolicName: org.free.chart  
Bundle-Version: 1.0.12
```

- » Beispiel: Versionierung von Packages

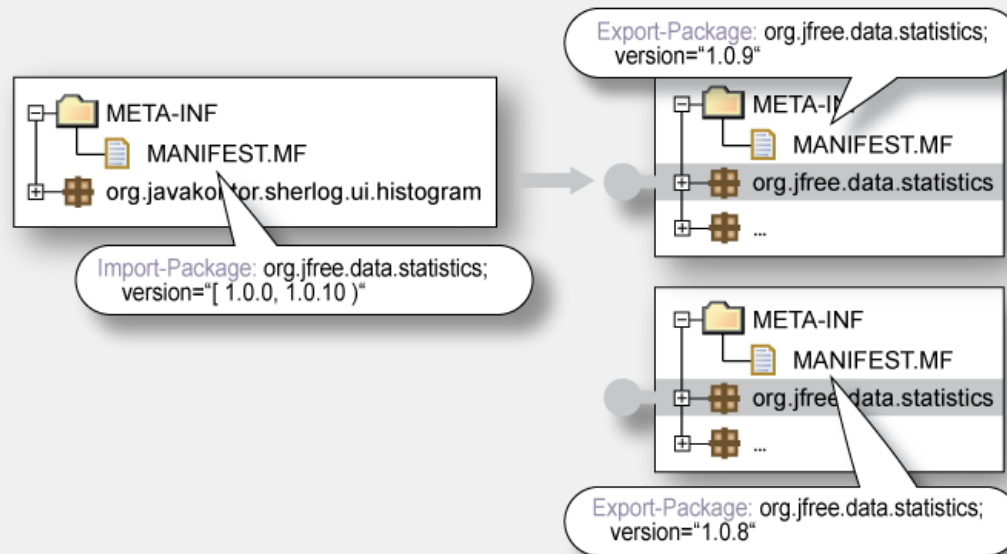
```
Export-Package: org.jfree.data.statistics;  
    version="1.0.12"
```

Versionsbereiche

- » Abhängigkeiten können mit Versionsbereichen genauer beschrieben werden
- » Angabe von Minimalwert und Maximalwert:
 - » (1.1, 1.2] entspricht: $1.1.0 < x \leq 1.2.0$
 - » (): „größer-als“ bzw. „kleiner-als“
 - » []: „größer-oder-gleich-als“ bzw. „kleiner-oder-gleich-als“
 - » [1,2] entspricht $1.0.0 \leq x \leq 2.0.0$
 - » 1.2 entspricht $1.2 < x$
 - » Default: $[0.0.0, \infty)$
- » Beispiel: Version ≥ 1.0 und < 1.5 ist:

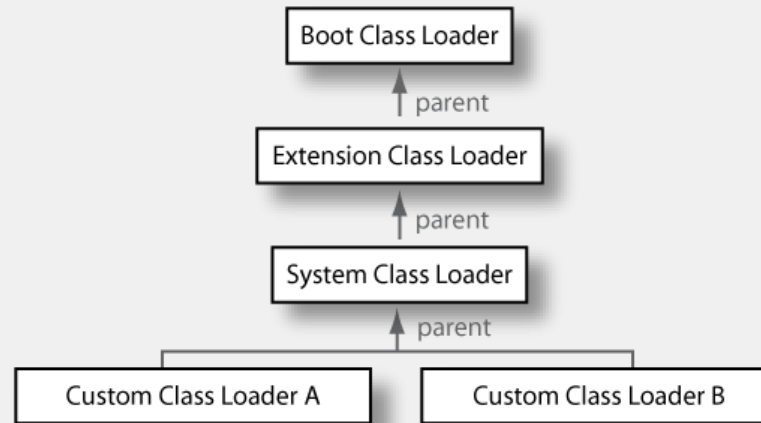
```
Import-Package: org.jfree.data.statistics;  
version="[1.0.0, 1.5.0)"
```

Importieren versionierter Packages



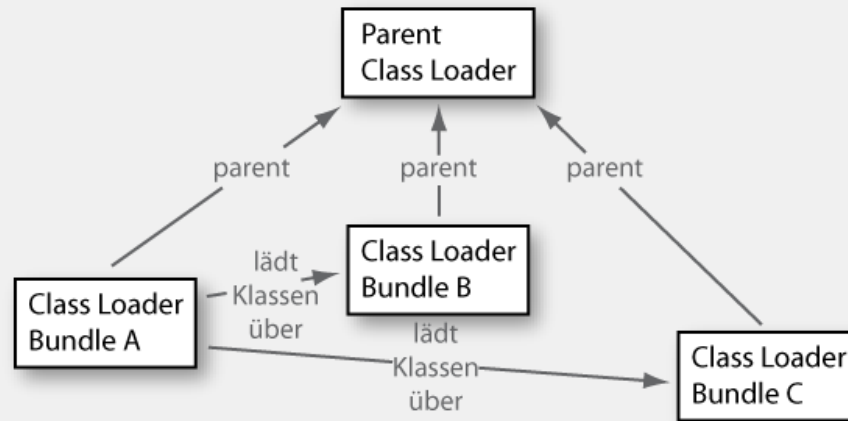
- » Passen mehrere Packages/Bundles zu einen Versionsbereich, wird das Package/Bundle mit höchster Versionsnummer zugewiesen

Exkurs: Class Loading in Java



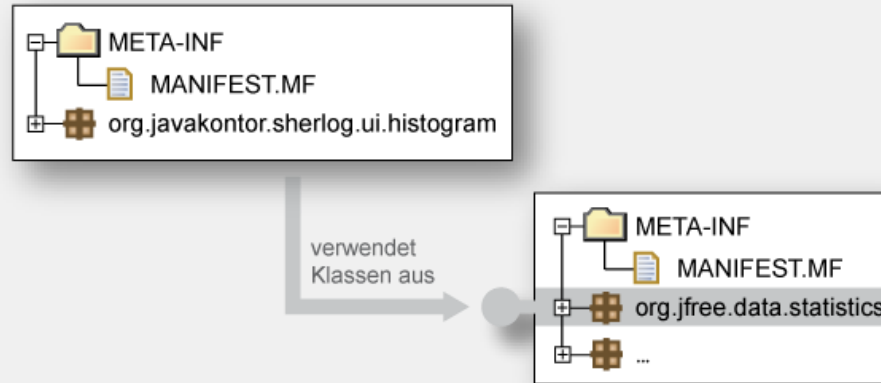
- » Class Loader: Verantwortlich für das Laden von Klassen in einer VM
- » Class Loader-Hierarchie: Zunächst wird Parent nach einer Klasse befragt (Delegationsmodell)
- » OSGi macht intensiv Gebrauch von Class Loadern...
 - » Bundles sehen nur Klassen importierter Packages
 - » Hot-Deployment von Bundles

Exkurs: Class Loading in OSGi



- » Parent Class Loader ist (i.d.R.) der Boot Class Loader der VM
- » Suchreihenfolge:
 - » Parent Class Loader für Klassen aus `java.*`-Packages und Packages, die in „`org.osgi.framework.bootdelegation`“ aufgelistet sind
 - » Klassen importierter Packages werden aus CL des jeweiligen Bundles geladen
 - » Class Loader des eigenen Bundle wird verwendet

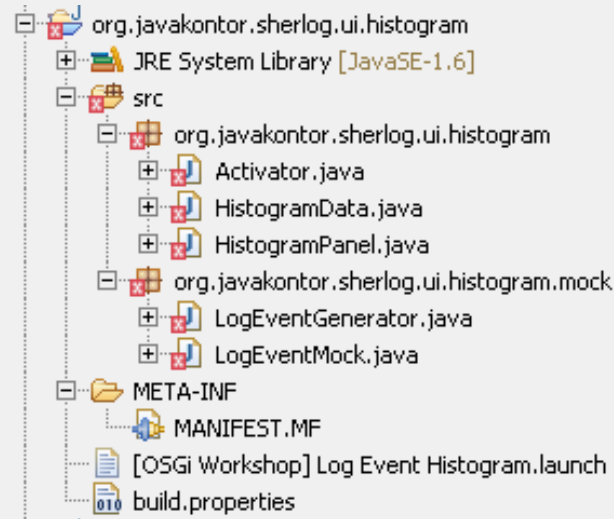
Tutorial 2: Package Dependencies



Aufgabe:

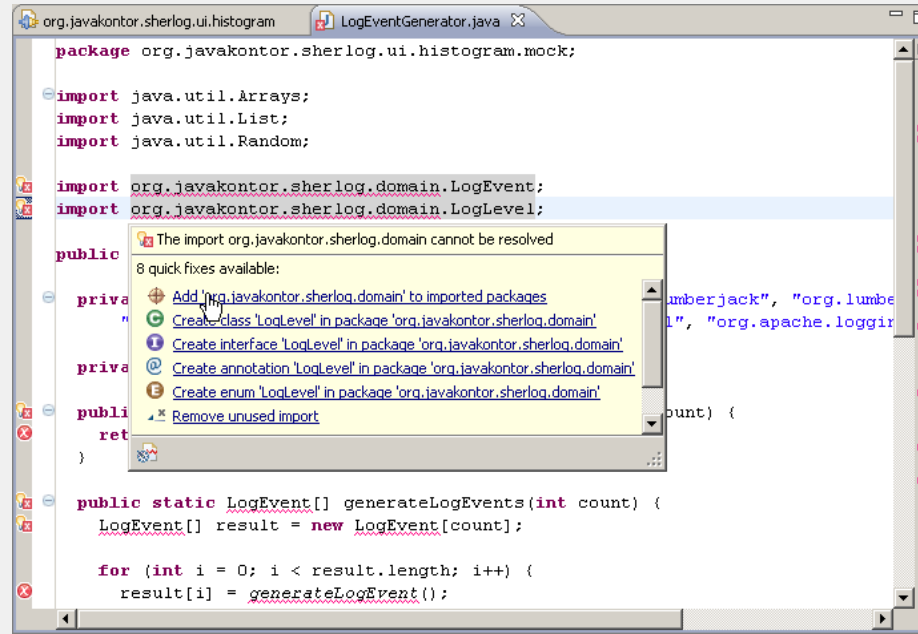
- » Laden Sie das „Tutorial-2_Package-Dependencies“
- » Importieren Sie alle zur Ausführung des Beispiel benötigten Packages
- » Führen Sie das Beispiel aus

2.1 Die Ausgangssituation



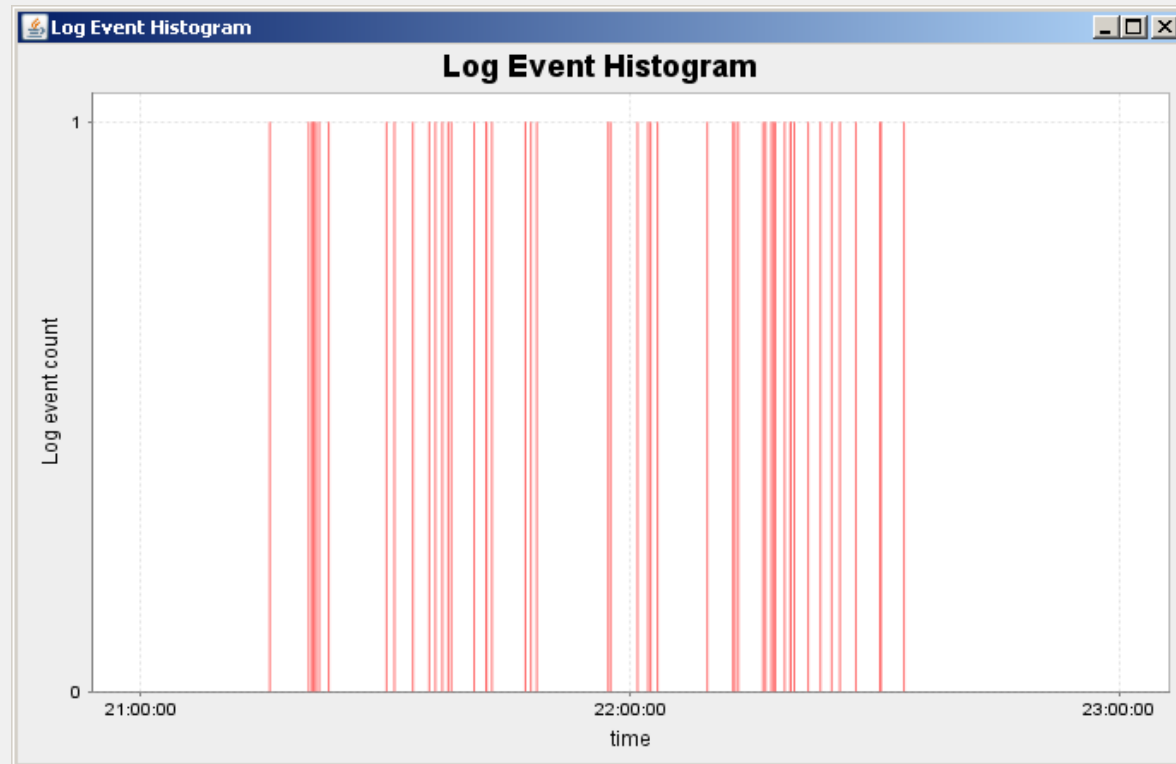
- » Das Projekt „org.javakontor.sherlog.ui.histogram“ benutzt Klassen aus verschiedenen Packages
- » Die entsprechenden Packages müssen noch importiert werden

2.2 Importieren der Packages



- » Importieren Sie alle benötigten Packages (nutzen Sie dazu bspw. den entsprechenden „Quick-Fix“)
- » Achtung: In zwei Fällen müssen Packages importiert werden, die nicht genutzt werden. Welche Packages sind das?

2.3 Ausführen des Beispiels

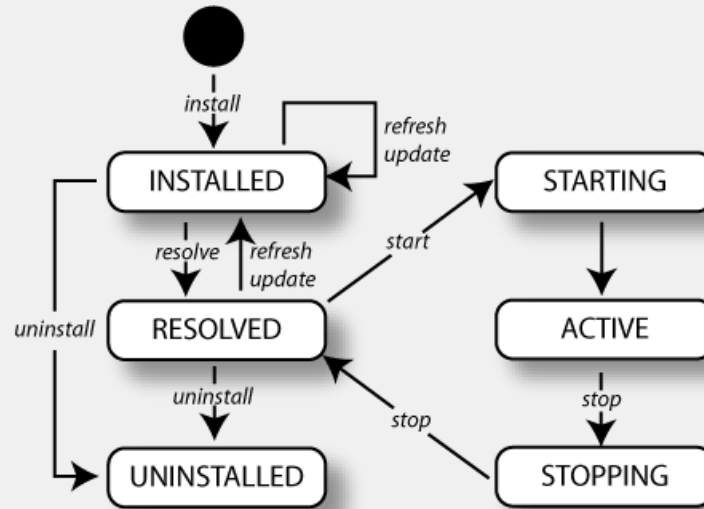


» Führen Sie das Beispiel aus Tutorial 2 aus

Agenda

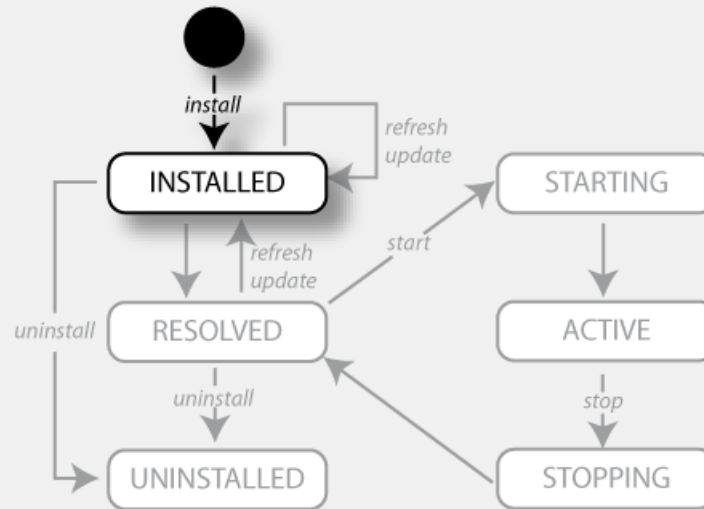
- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogramm-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » **Bundle-Lebenszyklus**
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

Der Bundle Lifecycle



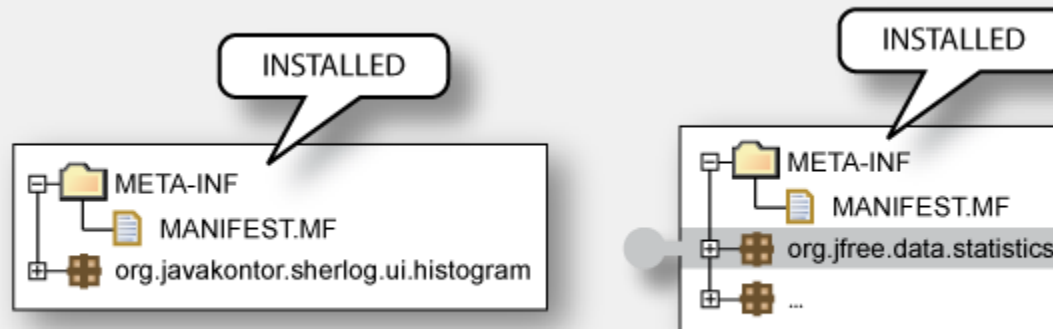
- » Bundles haben einen definierten Lebenszyklus
- » Zustandsänderungen können programmatisch oder durch einen Management Agent getriggert werden
- » Einige Zustandsübergänge erfolgen automatisch

Installieren von Bundles I



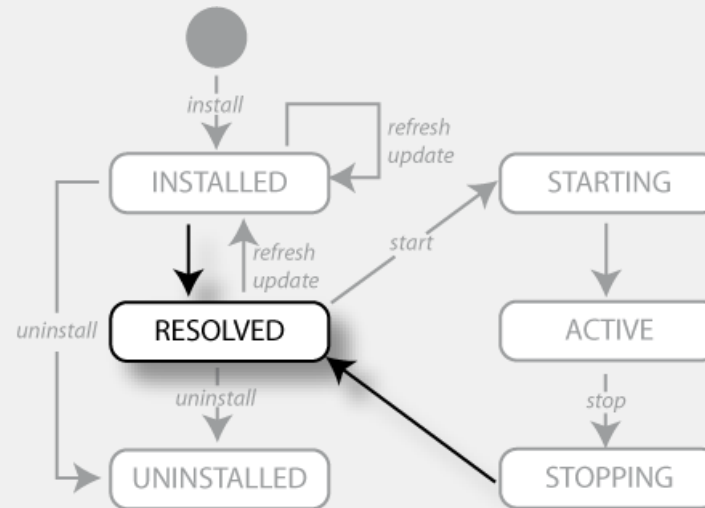
- » Das Bundle wird (persistent) im OSGi Framework verfügbar gemacht
- » Dem Bundle wird ein eindeutiger Bundle identifier (long) zugewiesen
- » Der Bundle-Zustand wird auf `INSTALLED` gesetzt

Installieren von Bundles



- » Ein Bundle im Zustand INSTALLED kann (noch) nicht genutzt werden:
 - » Das Bundle kann nicht gestartet werden
 - » Packages werden nicht exportiert

Resolving



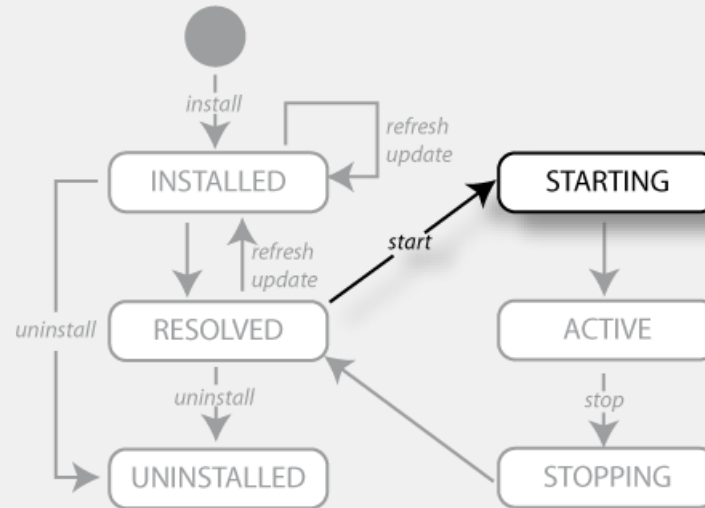
- » Importierten Packages werden exportierte Packages zugeordnet
- » Ist dies erfolgreich, wird der Zustand auf **RESOLVED** gesetzt
- » Das Resolving erfolgt unmittelbar (nach der Installation) oder verzögert

Resolving



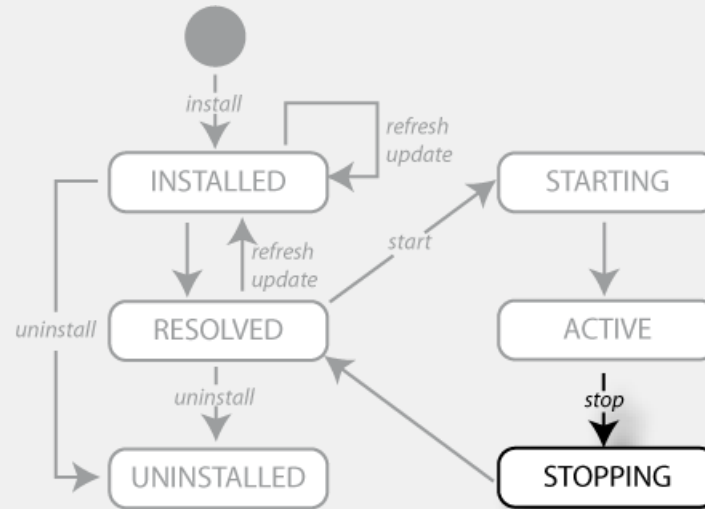
- » Ein Bundle im Zustand RESOLVED kann gestartet werden
- » Exportierte Packages können von anderen Bundles importiert werden

Starten von Bundles



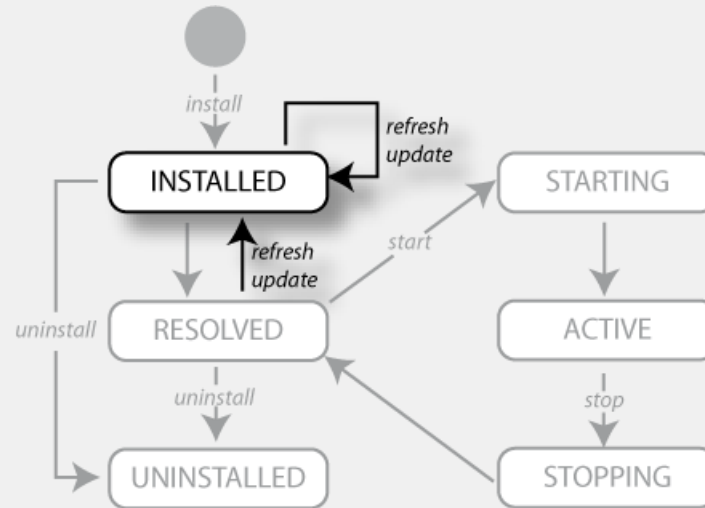
- » Das Bundle wird in den Zustand **STARTING** gesetzt
- » Der Bundle-Aktivator wird instantiiert und die *start()*-Methode aufgerufen
 - » Erfolgreiche Rückkehr: das Bundle in den Zustand **ACTIVE** gesetzt
 - » Fehlerfall: das Bundle in den Zustand **RESOLVED** gesetzt

Stoppen von Bundles



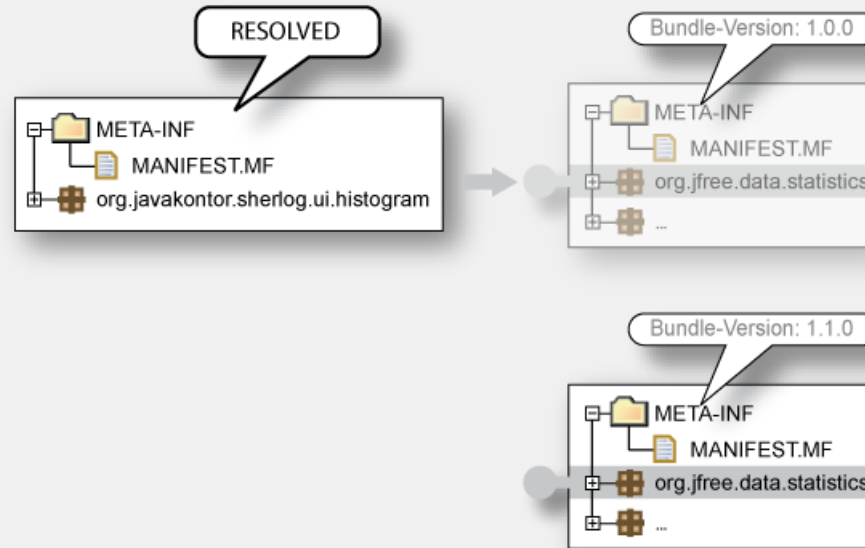
- » Das Bundle wird in den Zustand **STOPPING** gesetzt
- » Falls ein Aktivator deklariert ist, wird `BundleActivator.stop()` aufgerufen
- » Nach der Rückkehr wird das Bundle in den Zustand **RESOLVED** gesetzt

Aktualisieren von Bundles I



- » *Update* liest das Bundle erneut ein und aktualisiert es
 - » Falls das Bundle Packages exportiert: Bestehende Package-Zuordnungen bleiben bestehen (stale packages)!
- » *Refresh* führt dazu, dass importierte Packages neu zugeordnet werden

Aktualisieren von Bundles II



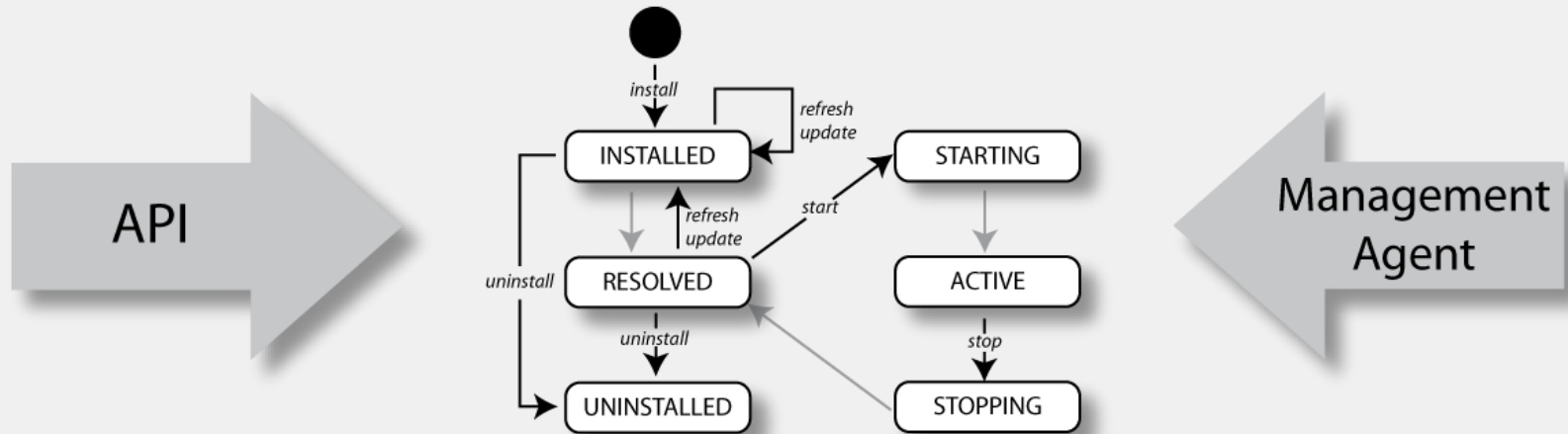
- » Package-Abhängigkeiten werden nicht automatisch neu aufgelöst.
- » Explizites „Refresh“ notwendig.

Deinstallieren von Bundles



- » Package-Abhängigkeiten werden nicht automatisch neu aufgelöst.
- » Explizites „Refresh“ notwendig.

Den Bundle-Lebenszyklus manipulieren



- » Zustandsänderungen können programmatisch oder durch einen Management Agents überwacht und verändert werden

Die Equinox-Konsole

```
C:\WINDOWS\system32\cmd.exe - java -jar plugins/org.eclipse.osgi_3.3.0.v20070530.jar -console

C:\equinox>java -jar plugins/org.eclipse.osgi_3.3.0.v20070530.jar -console
osgi> ss
Framework is launched.
id      State      Bundle
0       ACTIVE     org.eclipse.osgi_3.3.0.v20070530
osgi> install file:/c:/bundles/de.serversideeclipse.helloworld_1.0.0.jar
Bundle id is 1
osgi> ss
Framework is launched.
id      State      Bundle
0       ACTIVE     org.eclipse.osgi_3.3.0.v20070530
1       INSTALLED  de.serversideeclipse.helloworld_1.0.0
osgi> start 1
```

- » Kommandobasierter Management Agent
- » Integraler Bestandteil von Eclipse Equinox
- » Wird gestartet, wenn Equinox mit *-console* oder *-Dosgi.console* gestartet wird

Equinox-Konsole: Wichtige Kommandos

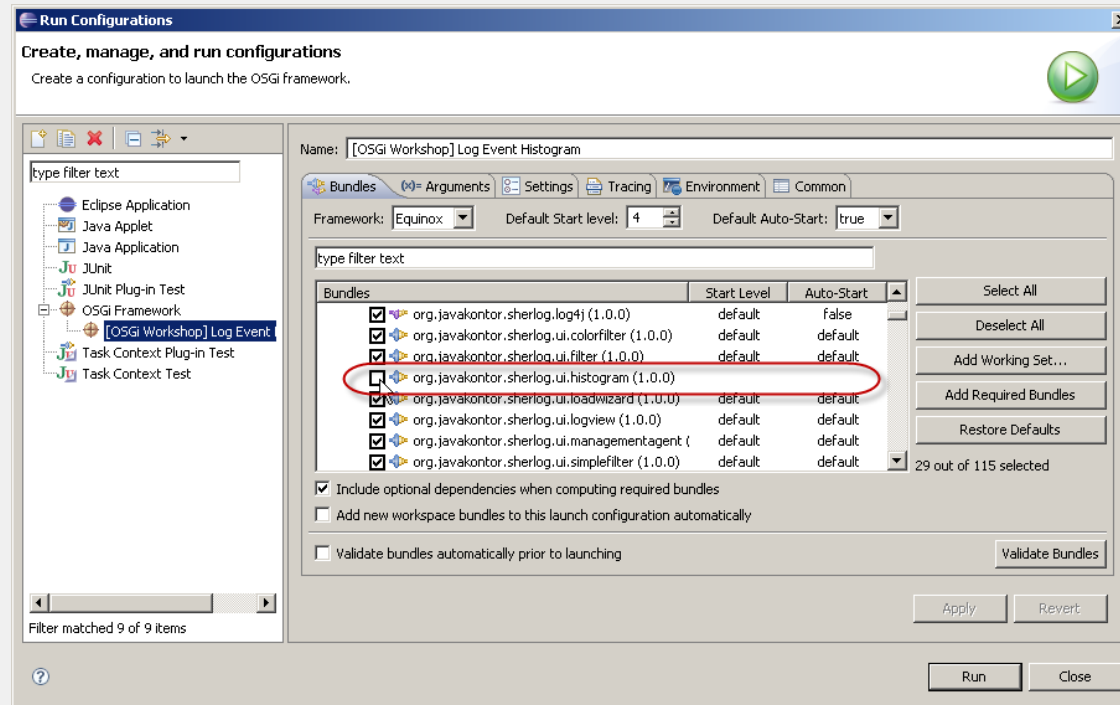
Command	Effect
ss	Zeigt alle installierte Bundle mit Id und Zustand an
install <url>	Installiert das angegebene Bundle
update <id>	Aktualisiert das angegebene Bundle
refresh <id>	Aktualisiert das angegebene Bundle
start <id>	Startet das angegebene Bundle
stop <id>	Stoppt das angegebene Bundle
bundle <id>	Zeigt Details des angegebene Bundles an
services [filter]	Zeigt registrierte Services an
help	Zeigt die Hilfe an

Tutorial 3: Lebenszyklus Bundle

Aufgabe:

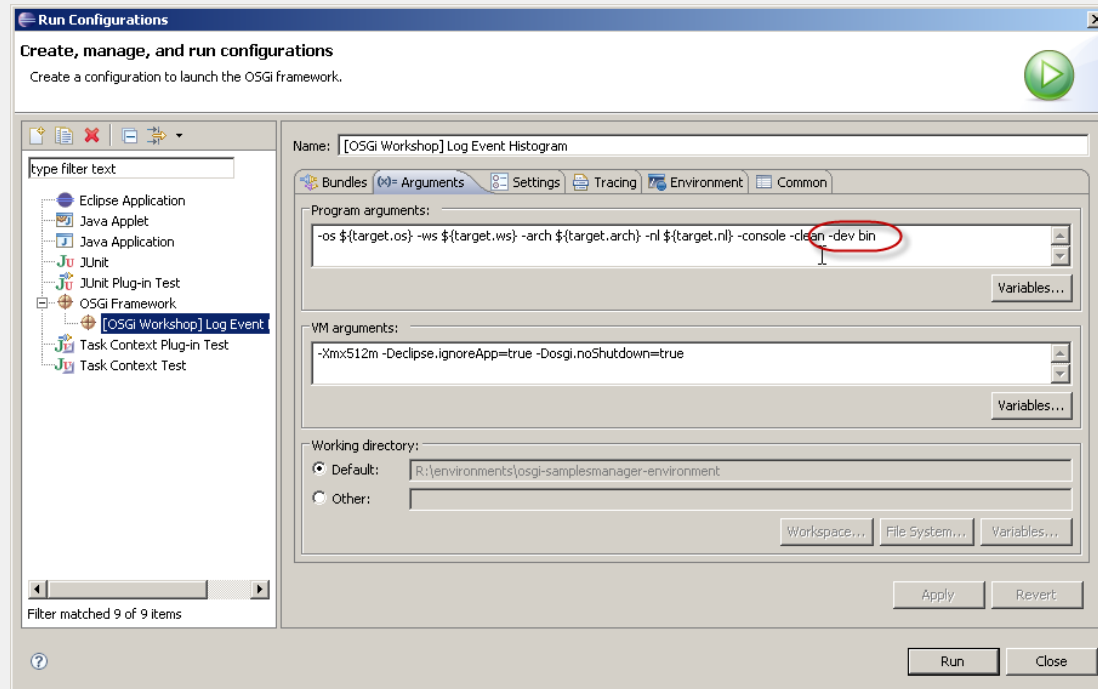
- » Entfernen Sie das „Histogram“-Bundle aus der Launch-Konfiguration und starten Sie das OSGi-Framework.
- » Installieren und starten Sie das Bundle im laufenden Framework.

3.1 Anpassen der Launch-Konfiguration I



» Entfernen Sie das „Histogram“-Bundle aus der Launch-Konfiguration

3.2 Anpassen der Launch-Konfiguration II



» Fügen Sie der Launch-Konfiguration die Start-Parameter „-dev bin“ hinzu

3.3 Argumente spezifizieren

» Programmargumente:

```
-os ${target.os} -ws ${target.ws} -arch  
  ${target.arch}  
-nl ${target.nl} -console -clean -dev bin
```

» VM-Argumente:

```
-Declipse.ignoreApp=true -Dosgi.noShutdown=true
```

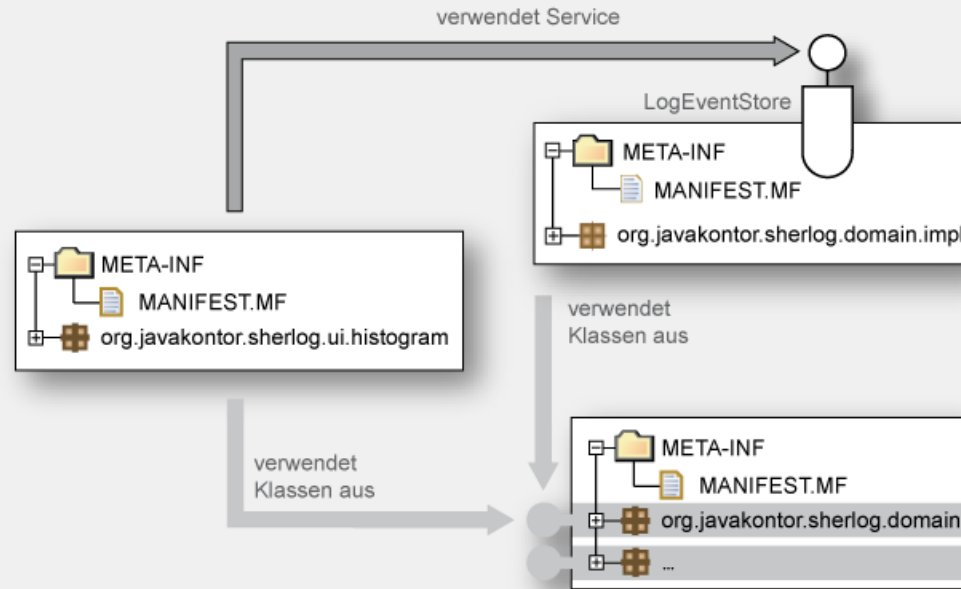
3.4 Arbeiten mit der Konsole

Command	Effect
ss	Zeigt alle installierte Bundle mit Id und Zustand an
install <url>	Installiert das angegebene Bundle
update <id>	Aktualisiert das angegebene Bundle
refresh <id>	Aktualisiert das angegebene Bundle
start <id>	Startet das angegebene Bundle
stop <id>	Stoppt das angegebene Bundle
bundle <id>	Zeigt Details des angegebene Bundles an
services [filter]	Zeigt registrierte Services an
help	Zeigt die Hilfe an

Agenda

- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogramm-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » **OSGi Services**
 - » *Tutorial: Services anmelden und abfragen*
- » Umgang mit dynamischen Services
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

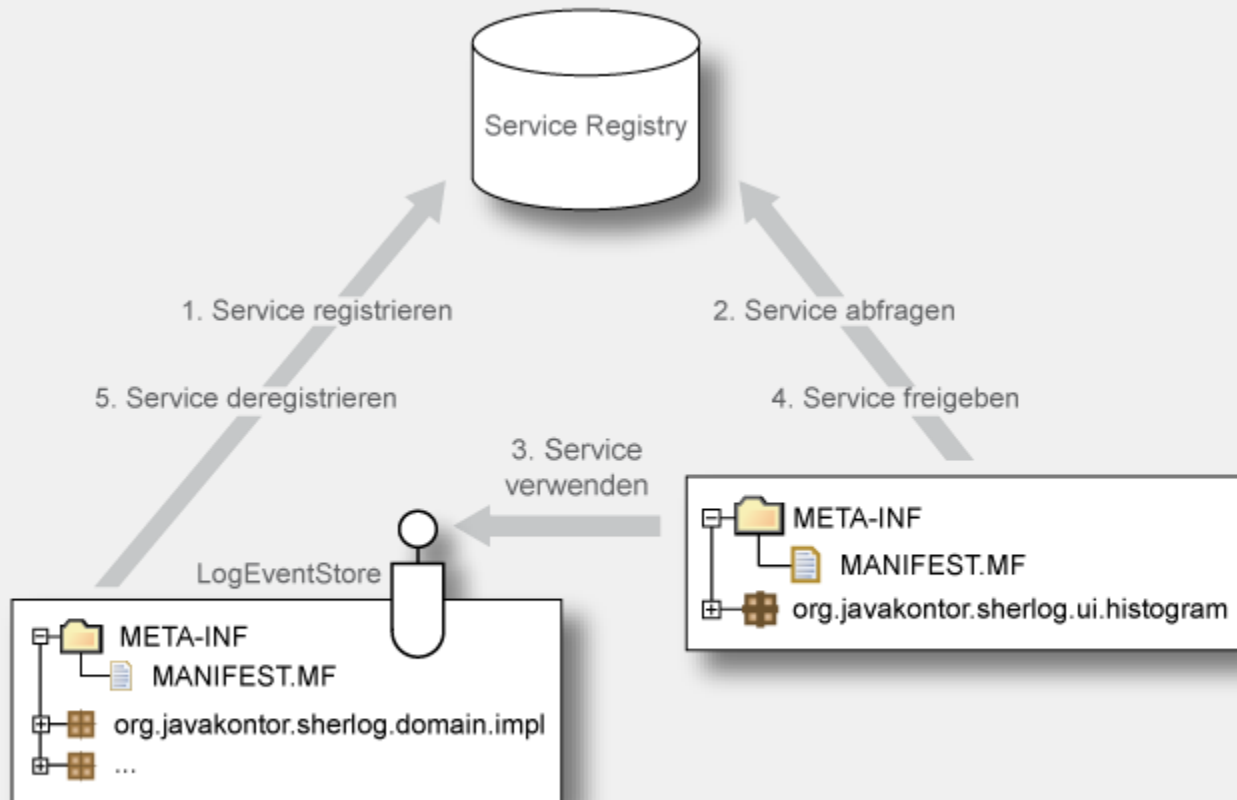
OSGi Services



Ein OSGi Service...

- » ...ist ein einfaches Java-Objekt (POJO),
- » ...wird an der zentralen Service Registry registriert,
- » ...kann von der zentralen Service Registry abgefragt werden.

OSGi Service Model



Services registrieren (1)

» Registrieren über den BundleContext:

```
package org.osgi.framework;

[...]
```

```
public interface BundleContext {

    [...]

    public ServiceRegistration registerService(
        String clazz, Object service, Dictionary properties);

    public ServiceRegistration registerService(
        String[] clazzes, Object service, Dictionary properties);

    [...]
}
```

Services registrieren (2)

» Die *ServiceRegistration* ...

- » ...ist für die private Nutzung innerhalb des registrierenden Bundles gedacht
- » ...ermöglicht das (nachträgliche) Ändern von Service-Properties
- » ...ermöglicht das manuelle Deregistrieren von Service

```
package org.osgi.framework;

public interface ServiceRegistration {

    public ServiceReference getReference();

    public void setProperties(Dictionary properties);

    public void unregister();
}
```

Services registrieren (3)

- » Services können nur im Zustand STARTING oder ACTIVE registriert werden:
 - » Typisches Szenario: *BundleActivator.start()*
 - » Registrierung kann durch beliebige Ereignisse getriggert werden.
- » Automatische Deregistrierung, wenn das registrierende Bundle gestoppt wird.

Services verwenden (1): Service-Referenzen

- » Der Zugriff auf einen OSGi Service erfolgt unter Verwendung einer Service-Referenz.
- » Die Service-Referenz...
 - » ... wird durch das Interface `org.osgi.framework.ServiceReference` repräsentiert
 - » ...kapselt die Metainformationen, die zu einem Service verfügbar sind
- » Abfragen der Service-Referenz über den Bundle-Kontext:

```
ServiceReference serviceReference = context.  
    getServiceReference(LogEventStore.class.getName());
```

- » Falls kein Service registriert ist, liefert die Methode null zurück.

Services verwenden (2)

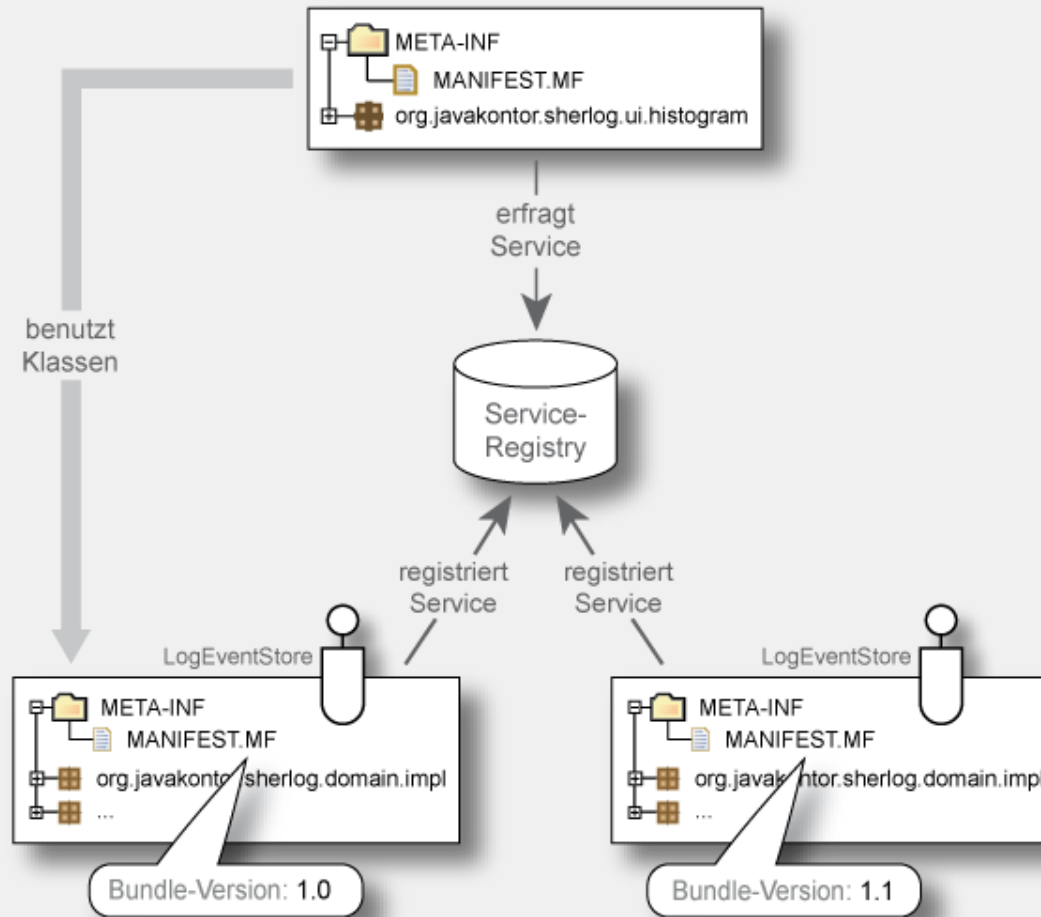
- » Über die Service Referenz kann der Service von der Service Registry abgefragt werden:

```
LogEventStore logEventStore =  
    (LogEventStore) context.getService(serviceReference);
```

- » Falls der Service deregistriert wurde, liefert die Methode null zurück
- » Nach Gebrauch müssen Services explizit freigegeben werden:

```
context.ungetService(serviceReference);
```

Package- und Service-Abhängigkeiten



Service Properties

- » Aufgaben:
 - » **Beschreibung von Services** - Anzeige von Properties in Management Agents
 - » **Filtern von Service** - Verwendung von Properties in Filter-Ausdrücken beim Abfragen von Services
- » Service-Properties werden in einem Dictionary-Objekt zusammengefasst (Key-Value-Paare)
- » Es können beliebige benutzerdefinierte Properties gesetzt werden
- » Daneben sind verschiedene Standard-Properties definiert

Wichtige Standard Service Properties

Service Property	Meaning
objectClass: String[]	Namen der bei der Registrierung angegebenen Service Interfaces (wird durch das Framework gesetzt)
service.id: Long	Eindeutiger Identifier (wird durch das Framework gesetzt)
service.ranking: Integer	Das Ranking eines Services

Filter (1)

- » Basieren auf RFC 1960, “String Representation of LDAP Search Filters”
- » **Kriterien** bestehen aus einer Forderung an ein LDAP-Attribut:
 - » (property = value)
 - » (property <= value)
 - » (property >= value)
 - » (property =*)
- » **Operatoren** verknüpfen Kriterien (polnische Notation):
 - » UND: (& (...) (...))
 - » ODER: (| (...) (...))
 - » NICHT: (! (...))
- » Beispiel: “(&(objectClass=net*)(service.ranking>=10))”

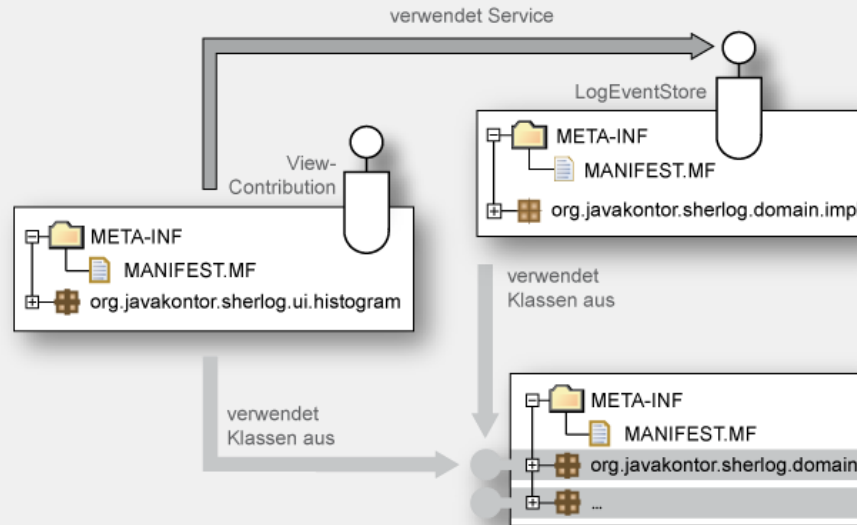
Filter (2)

- » Filter können verwendet werden, um einen abzufragenden Service näher zu beschreiben:

```
String filter =
    "&(objectClass=net*)(service.ranking>=10)";

ServiceReference[] serviceReferences =
    context.getServiceReferences(
        TranslationService.class.getName(),
        filter);
```

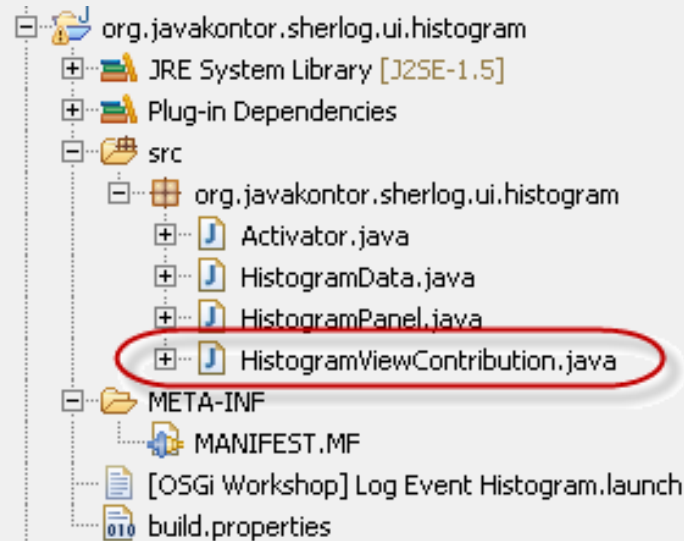
Tutorial 4: OSGi Services



Aufgabe:

- » Melden Sie für den Histogram View eine View Contribution als OSGi Service an
- » Verwenden Sie den zentralen Log Event Store, um Zugriff auf die Log Events zu erhalten

4.1 Exkurs: View Contribution



» *ViewContributions* werden innerhalb von Sherlog verwendet, um Fenster in das Application Framework einzubetten.

4.2 Die HistogramViewContribution registrieren

```
package org.javakontor.sherlog.ui.histogram;

[...]
```

```
public class Activator implements BundleActivator {

    private HistogramViewContribution _histogramViewContribution;

    public void start(BundleContext ctx) throws Exception {
        _histogramViewContribution = new HistogramViewContribution();
        context.registerService(ViewContribution.class.getName(),
            _histogramViewContribution, null);
        [...]
    }

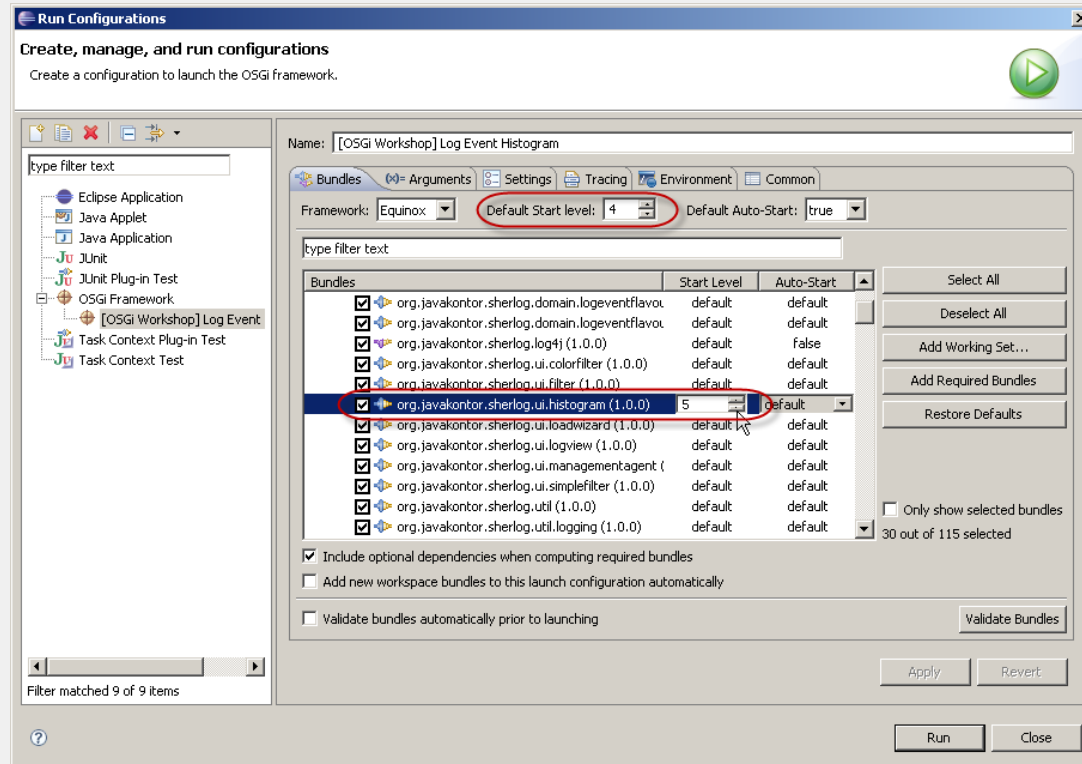
    public void stop(BundleContext context) throws Exception {
        // Deregistrierung erfolgt automatisch
    }
}
```

4.3 Den Log Event Store verwenden

```
public class Activator implements BundleActivator {
    [...]
    public void start(BundleContext ctx) throws Exception {
        ServiceReference serviceReference =
            context.getServiceReference(LogEventStore.class.getName());
        if (serviceReference != null) {
            LogEventStore logEventStore =
                (LogEventStore) context.getService(serviceReference);
            if (logEventStore != null) {
                _histogramViewContribution.bindLogEventStore(logEventStore);
            } else { [...] }
        } else { [...] }
    }

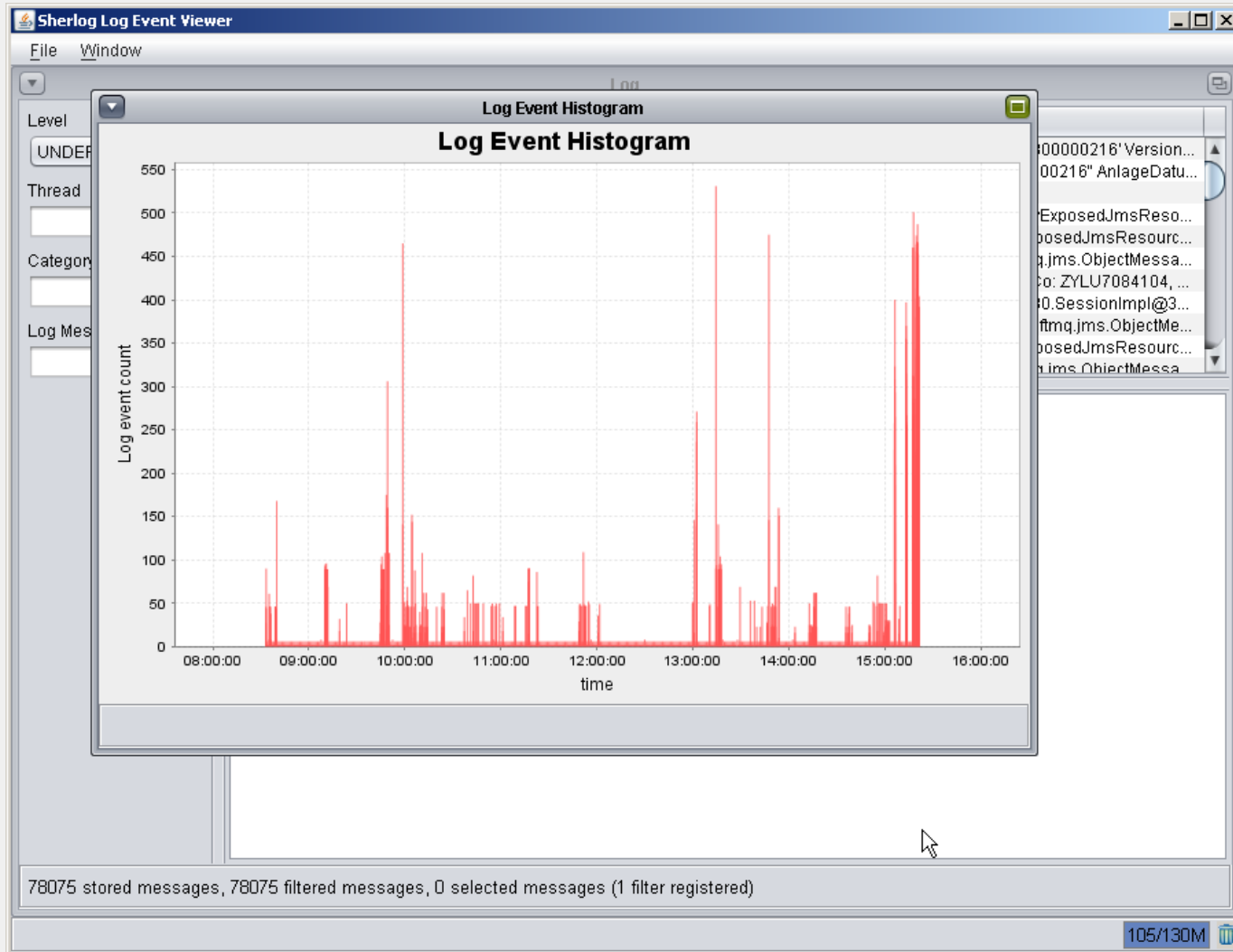
    public void stop(BundleContext context) throws Exception {
        _histogramViewContribution.unbindLogEventStore(null);
    }
}
```


4.3 Bundle-Startreihenfolge



» Achtung: Für dieses Beispiel muss die Bundle-Startreihenfolge festgelegt werden!

4.4 Ausführen den Beispiels



Agenda

- » OSGi-Technologie im Überblick
 - » *Tutorial: Eclipse IDE / Beispiele installieren*
- » Bundles
 - » *Tutorial: Ein Histogramm-Bundle*
- » Package-Abhängigkeiten zwischen Bundles
 - » *Tutorial: Definition von Package-Abhängigkeiten*
- » Bundle-Lebenszyklus
 - » *Tutorial: Bundle zur Laufzeit installieren*
- » OSGi Services
 - » *Tutorial: Services anmelden und abfragen*
- » **Umgang mit dynamischen Services**
 - » *Tutorial: Arbeiten mit dem ServiceTracker*

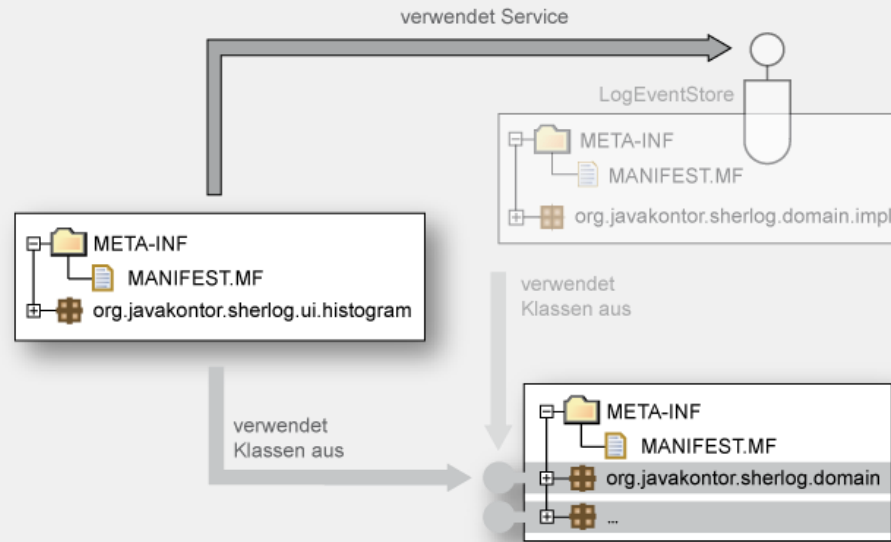
Dynamische Services

- » Bisher vorgestellte Services entkoppeln Bundles
- » Implementierung entkoppelt vom Interface
- » Die Verbindung zu den Services ist aber statisch

- » Das „echte“ Leben ist dynamisch
- » Problem insbesondere bei großen Anwendungen
 - » Die komplette Anwendung muss neu gestartet werden
 - » Die Startreihenfolge muss exakt eingehalten werden

- » Dynamische Services sind ein komplexes Thema
- » Wir zeigen die Grundlagen

Services können kommen und gehen



- » Die Suche nach einem Service kann scheitern, weil ...
 - » ... das implementierende Bundle noch nicht gestartet ist
 - » ... der Service noch nicht registriert ist
 - » ... das implementierende Bundle beendet wurde

ServiceListener

- » „Basismechanismus“ zum Verfolgen von Änderungen an der Service Registry:

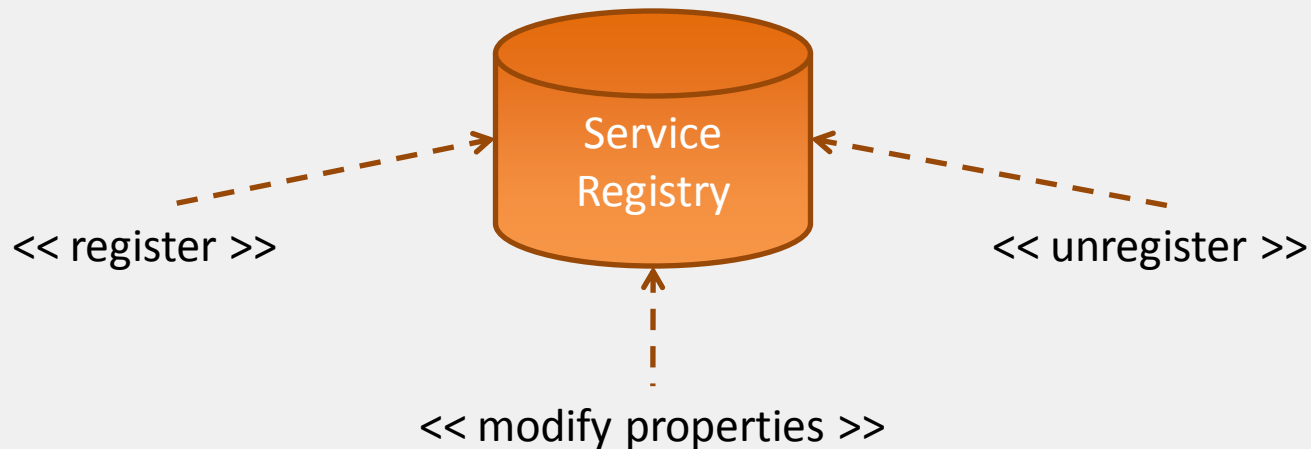
```
package org.osgi.framework;  
  
public interface ServiceListener extends EventListener {  
    public void serviceChanged(ServiceEvent event);  
}
```

- » ServiceListener werden über den BundleContext angemeldet:

```
ServiceListener listener = ... ;  
  
context.addServiceListener(listener);
```

Service Events

Service Event	Meaning
REGISTERED	Service was registered
MODIFIED	Service properties were modified
UNREGISTERING	Service is about to be unregistered



Reagieren auf dynamische Services (1)

- » Typische Aufgabe: Beobachte alle Services eines bestimmten Typs
 - » Alle Services die bereits im System vorhanden sind
 - » Alle Services, die zukünftig registriert werden
 - » Alle Services, die zukünftig deregistriert werden
- » Mögliche Lösung:
 - » (1) Suche alle bestehende Services
 - » (2) Melde einen Service Listener an
- » Problem:
Änderungen im Zeitfenster zwischen (1) und (2) bekommt man nicht mit!

Reagieren auf dynamische Services (2)

- » Alternative: Umgekehrte Reihenfolge
 - » (1) Erst Service Listener anmelden
 - » (2) Dann bestehende Services abfragen
- » Problem:
Änderungen im Zeitfenster zwischen (1) und (2)
können zu Duplikaten führen!

Best practice:

- » Service Listener sind fehleranfällig und schwierig zu benutzen.
- » Service Tracker vereinfachen den programmatischen Umgang mit dynamischen Services.

Der Service Tracker (1)

- » Auf Service Listener basierende Hilfsklasse:
`org.osgi.util.tracker.ServiceTracker`
- » Services können über den Namen des Service Interface oder einen Filter spezifiziert werden:

```
serviceTracker = new ServiceTracker(context,  
    LogEventStore.class.getName(), null);
```

- » Service Tracker müssen geöffnet werden, bevor sie benutzt werden können:

```
serviceTracker.open();
```

- » Service Tracker müssen explizit geschlossen werden:

```
serviceTracker.close();
```

Der Service Tracker (2)

- » Einfacher Zugriff auf Service Referenzen und Service Objekte:

```
public Object[] getServices;  
public Object getService();  
public ServiceReference[] getServiceReferences();  
public ServiceReference getServiceReference();  
public Object getService(ServiceReference reference);  
  
public Object waitForService(long timeout)  
    throws InterruptedException;
```

Der Service Tracker Customizer

- » Stellt Hooks für den Service Lifecycle bereit:
 - » *addingService()*: Ein Service ist in der Service Registry vorhanden.
 - » *modifiedService()* : Die Properties eines Service haben sich geändert.
 - » *removedService()*: Ein Service wurde aus der Registry entfernt.
- » Achtung: Das übergebene Objekt muss nicht der Service sein!

Der Service Tracker Customizer

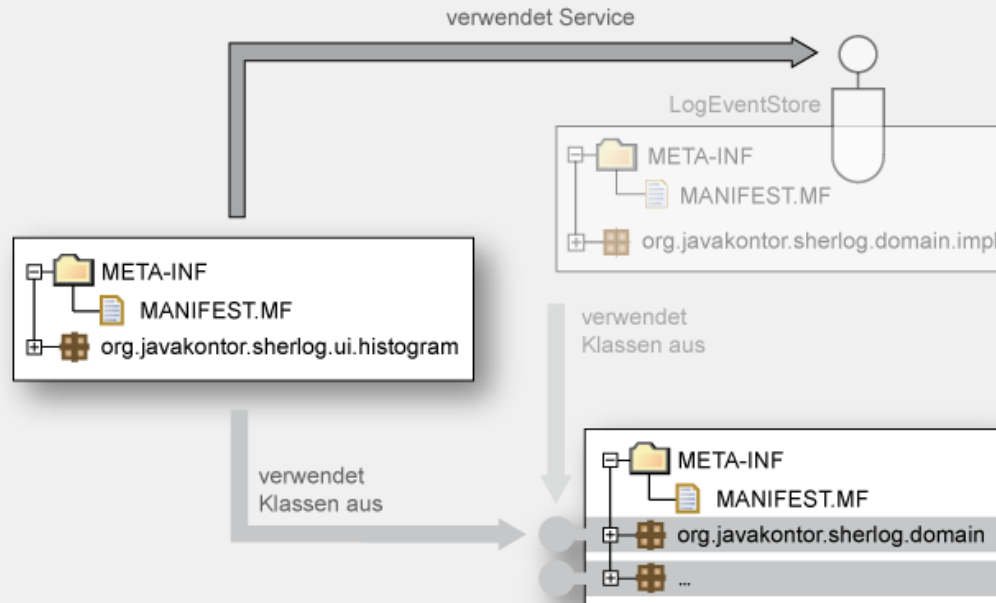
```
package org.osgi.util.tracker;

public interface ServiceTrackerCustomizer {
    public Object addingService(ServiceReference reference);
    public void modifiedService(ServiceReference reference,
        Object service);
    public void removedService(ServiceReference reference,
        Object service);
}
```

Aufbauende Frameworks

- » **Declarative Services**
 - » Teil der OSGi-Spezifikation
 - » Deklarative Beschreibung von Services mit XML
 - » Kapitel 12 in “Die OSGi Service Platform”
- » **Spring Dynamic Modules**
 - » Spring wird mittels OSGi dynamisch
 - » <http://www.springframework.org/osgi>
 - » Blueprint Service in OSGi Specification R 4.2
- » **iPojo**
 - » “Original” DI framework for OSGi
 - » <http://ipojo.org>
- » **Guice - Peaberry**
 - » Guice: Performantes, leichtgewichtiges DI Framework
 - » Peaberry: Erweiterung zu Guice für OSGi
 - » <http://code.google.com/p/peaberry/>
 - » <http://code.google.com/p/google-guice/>

Tutorial 5: Dynamische Services



Aufgabe:

- » Verwenden Sie den ServiceTracker, um dynamisch auf die (De-)Registrierung des Log Event Store zu reagieren.

5.1 Implementierung des ServiceTrackers

```
package org.javakontor.sherlog.ui.histogram;

class LogEventStoreServiceTracker extends ServiceTracker {

    public LogEventStoreServiceTracker(BundleContext context) {
        super(context, LogEventStore.class.getName(), null);
    }

    public Object addingService(ServiceReference reference) {
        LogEventStore logEventStore =
            (LogEventStore) super.addingService(reference);
        Activator.this._viewContribution.bindLogEventStore(logEventStore);
        return logEventStore;
    }

    public void removedService(ServiceReference reference, Object service) {
        Activator.this._viewContribution.unbindLogEventStore(null);
        super.removedService(reference, service);
    }
}
```


5.2 Öffnen/Schließen des Service Trackers

```
package org.javakontor.sherlog.ui.histogram;

[...]
```

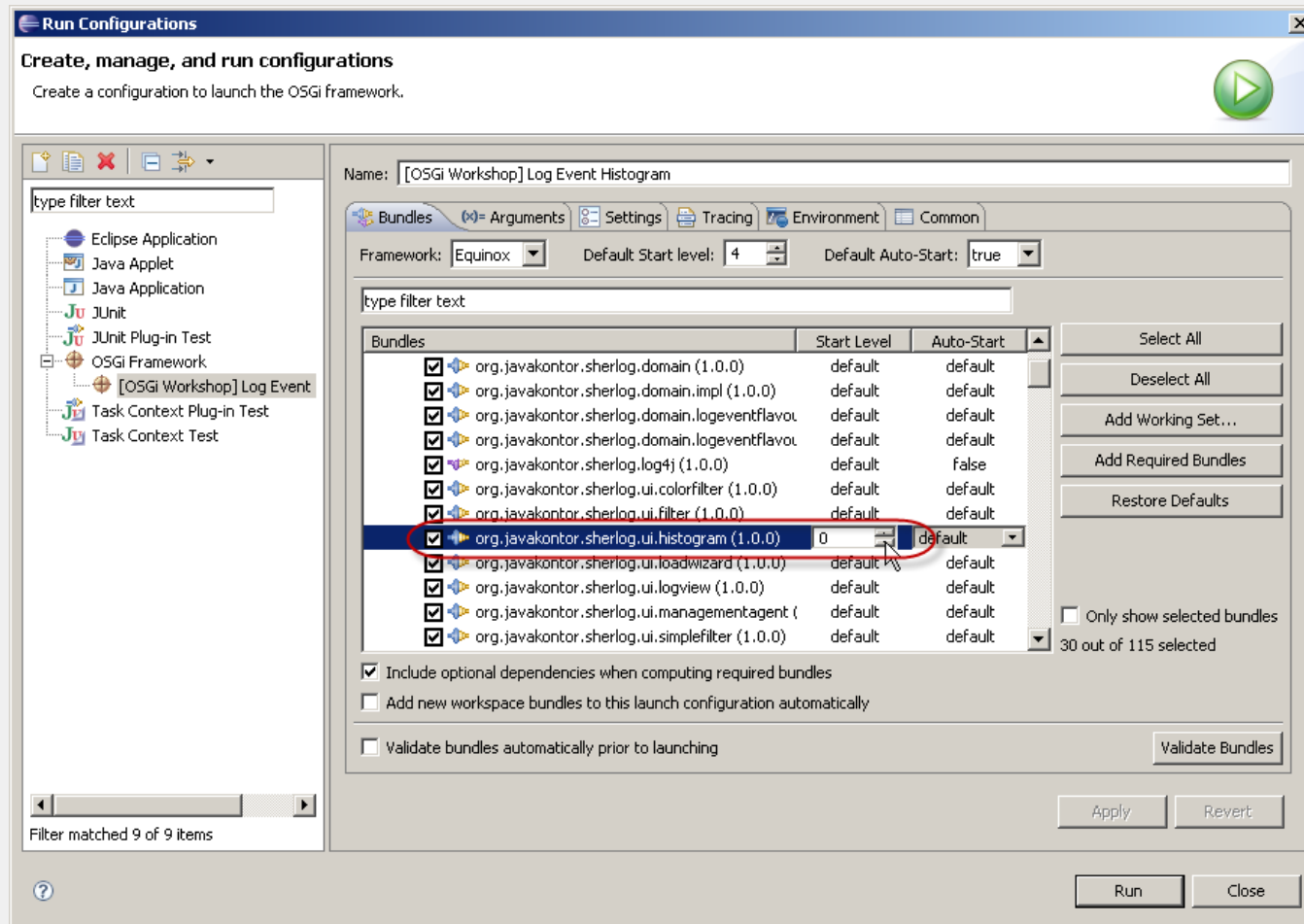
```
public class Activator implements BundleActivator {

    private ServiceTracker _serviceTracker;

    public void start(BundleContext context) throws Exception {
        [...]
        _serviceTracker = new LogEventStoreServiceTracker(context);
        _serviceTracker.open();
    }

    public void stop(BundleContext context) throws Exception {
        _serviceTracker.close();
    }
}
```

5.3 Anpassen der Launch-Konfiguration



5.4 Ausführen des Beispiels

The screenshot displays the Sherlog Log Event Viewer application. The main window is titled "Sherlog Log Event Viewer" and contains a menu bar with "File" and "Window". On the left, there are filters for "Level" (set to UNDEF), "Thread", "Category" (set to g.springframework), and "Log Message". The central area shows a table of log events with columns for Time, LogLevel, Tra..., and Category. The table contains several rows of log entries from 2008-10-21 08:00:00, all at the DEBUG level, from the org.springframework category.

On the right, there is a "Log Event Histogram" window showing a bar chart of event counts over time. The y-axis is labeled "event count" and ranges from 120 to 260. The x-axis is labeled "time" and ranges from 10:00:00 to 16:00:00. The histogram shows several peaks, with the highest peak reaching approximately 260 events around 13:00:00.

In the foreground, a "Management Agent" window is open, displaying a table of bundles. The table has columns for Bundl..., Symbolic name, Version, State, and Loc... The bundles listed are:

Bundl...	Symbolic name	Version	State	Loc...
10	org.javakontor.sherlog.application	1.0.0	ACTIVE	initi...
13	org.javakontor.sherlog.application.mvc	1.0.0	ACTIVE	initi...
21	org.javakontor.sherlog.domain	1.0.0	ACTIVE	initi...
8	org.javakontor.sherlog.domain.impl	1.0.0	ACTIVE	initi...
26	org.javakontor.sherlog.domain.loge			initi...
14	org.javakontor.sherlog.domain.loge			initi...
15	org.javakontor.sherlog.log4j			ED initi...
30	org.javakontor.sherlog.ui.colorfilter			initi...

A context menu is open over the bundle with symbolic name "org.javakontor.sherlog.domain.impl", showing options: "Start Bundle", "Stop Bundle", "Update Bundle", "Uninstall Bundle", and "Copy Location to Clipboard". Below the table, the details for the selected bundle are shown:

Identifier: 8
Symbolic Name: org.javakontor.sherlog.dom...
Version: 1.0.0
Location: initial@reference:file:///.../org.javakontor.sherlog.domain.impl/
Status: ACTIVE

At the bottom of the Management Agent window, it says "org.javakontor.sherlog.domain.impl (3 services registered, 2 services in use)".

The system tray at the bottom right shows the date and time "11/8/2008 11:18:17 AM" and a memory usage indicator "118/179M".

Vielen Dank! Fragen?

