

3 Bundles

In den bisherigen Ausführungen zur OSGi Service Platform haben Sie das *Bundle* als Modularisierungseinheit von Anwendungen kennengelernt. Dieses Konzept werden wir in diesem Kapitel näher beleuchten und Ihnen den grundsätzlichen Aufbau von Bundles vorstellen. Sie lernen das Bundle Manifest mit seinen verschiedenen Manifest Headern sowie grundlegende Framework-Klassen kennen.

Motivation

Lassen Sie sich bitte nicht verwirren, wenn Sie im Folgenden neben dem Konzept des *Bundles* scheinbar mit einem zweiten Konzept, dem *Eclipse-Plug-in*, konfrontiert werden. Beide Begriffe beschreiben ein und dasselbe Konzept, und innerhalb der Eclipse IDE wird der Begriff »Plug-in« synonym zum Begriff »Bundle« verwendet.

Bundles vs. Plug-ins

Dass innerhalb von Eclipse nicht von Bundles, sondern von Plug-ins die Rede ist, liegt an der Tatsache, dass Eclipse bis zur Version 3.0 eine eigene, proprietäre Modularisierungsinfrastruktur besaß, in der die Module Plug-ins hießen. Mit der Integration des OSGi Frameworks in Eclipse wurde zwar die proprietäre Infrastruktur abgelöst, die Begrifflichkeit wurde jedoch beibehalten.

Die in diesem Kapitel vorgestellten Konzepte sind im Module Layer des OSGi Frameworks angesiedelt (vgl. Abb. 3–1) und können in Kapitel 3 der OSGi Core Specification [OSCS07] nachgelesen werden. Die Eclipse-spezifischen Konzepte wie bspw. das Arbeiten mit Plug-in-Projekten sind in der Online-Hilfe der Eclipse IDE [ECHE33] beschrieben.

Einordnung

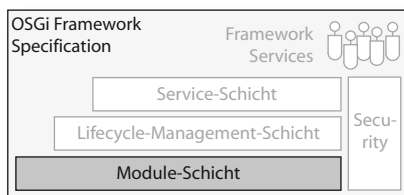


Abb. 3–1
Einordnung in die logischen Framework-Schichten

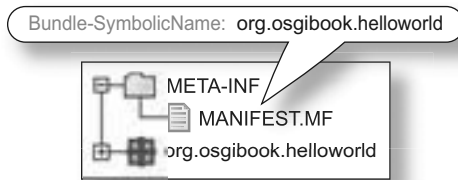
3.1 Tutorial: Ein »Hello World«-Bundle

Überblick Jeder erste Kontakt mit einer Programmiersprache oder einem Framework beginnt mit einem »Hello World«-Programm. Auch wir möchten uns dieser Tradition nicht entziehen und werden Sie nachfolgend durch die Implementierung eines Bundles führen, das eine »Hello OSGi World«-Nachricht auf der Systemkonsole ausgibt.

Das »Hello World«-Bundle enthält ein einziges Package mit genau einer Klasse, dem Bundle-Aktivator (vgl. Abb. 3–2).

Abb. 3–2

Das »Hello World«-
Bundle



Der Bundle-Aktivator implementiert die Ausgabe des »Hello OSGi World«-Grußes, die erfolgen soll, sobald das Bundle gestartet wird. Zusätzlich soll beim Stoppen des Bundles eine »Goodbye OSGi World«-Nachricht ausgegeben werden.

Schritt 1: Anlegen des »Hello World«-Plug-in-Projektes

Ein zu entwickelndes Bundle wird innerhalb der Eclipse IDE immer durch ein *Plug-in-Projekt* repräsentiert. Ein Plug-in-Projekt enthält neben dem Bundle Manifest alle zum Bundle gehörenden Quellen und Ressourcen. Für unser »Hello World«-Beispiel müssen Sie also zunächst ein neues Plug-in-Projekt anlegen. Dazu verwenden Sie den Plug-in Project Wizard, den Sie über **File -> New -> Project ... -> Plug-in Project** aufrufen.

*Spezifikation des
Plug-in-Projektes*

Auf der Startseite des Wizards (vgl. Abb. 3–3) müssen Sie die verschiedenen Informationen spezifizieren, die sich auf das zu generierende Plug-in-Projekt beziehen:

- den Projektnamen
(hier: `org.osgibook.helloworld`),
- den Namen der Source- und Output-Folder
(hier: `src` bzw. `bin`),
- die Zielplattform für das Bundle
(hier: `an OSGi Framework/standard`)

Über die Auswahl der Zielplattform (Target Platform) legen Sie fest, ob das Bundle innerhalb der Eclipse IDE oder auf Basis eines OSGi

Frameworks ablaufen soll. Abhängig von dieser Einstellung bietet Ihnen der Plug-in Project Wizard auf der letzten Seite des Wizards verschiedene Projekt-Templates zur Auswahl an. In unserem Fall wählen Sie **OSGi Framework** und ändern die Auswahlbox auf **standard**.



Abb. 3-3
Der Plug-in
Project Wizard

Der Zusatz **standard** in der Auswahlbox gibt an, dass innerhalb des Bundle Manifests des zu entwickelnden Bundles keine Equinox-spezifischen Erweiterungen verwendet werden, so dass das Bundle auch mit alternativen R4-kompatiblen OSGi-Implementierungen, wie bspw. dem Knopflerfish OSGi Framework, lauffähig ist.

Wenn Sie alle Einstellungen getroffen haben, können Sie über den Button **Next >** zur nächsten Seite des Wizards wechseln, auf der Sie Informationen bzgl. des *Plug-in-Inhaltes* spezifizieren können. Geben Sie im Feld **Plug-in Name** **Hello World Bundle** ein. Die anderen Inhalte können Sie wie vorgelegt übernehmen und das Plug-in-Projekt durch die Aktivierung der **Finish**-Schaltfläche erzeugen.

*Spezifikation der
Plug-in-Inhalte*

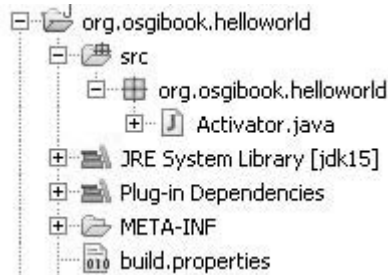
Nach Beenden des Plug-in-Project-Wizards befindet sich in Ihrem Workspace ein neues Projekt, das den in Abb. 3-4 dargestellten Aufbau besitzt. Das generierte Projekt enthält neben dem Bundle Manifest

*Das generierte
Beispielprojekt*

(MANIFEST.MF) die Klasse `Activator`, die im Package `org.osgibook.helloworld` abgelegt ist. Ferner besitzt das Projekt eine Datei `build.properties`, die wir allerdings in diesem Abschnitt nicht näher betrachten werden. Für nähere Informationen zu diesem Thema sei an dieser Stelle auf Kapitel 11 verwiesen.

Abb. 3-4

Das generierte
Beispielprojekt



Schritt 2: Implementierung der Activator-Klasse

Wenn Sie die `Activator`-Klasse mit einem Doppelklick öffnen, dann sehen Sie, dass die Klasse die zwei Methoden `start()` und `stop()` implementiert. Die `start()`-Methode wird aufgerufen, wenn das Bundle im OSGi Framework aktiviert wird. Analog wird die `stop()`-Methode aufgerufen, wenn das Bundle im OSGi Framework gestoppt wird.

Die »Hello World«-Ausgabe haben wir in diesem Beispiel in einer eigenen Methode `greet()` implementiert. Die Methode erwartet als Parameter eine Nachricht sowie Argumente für vorhandene Platzhalter in der Nachricht. Der Aufruf der `greet()`-Methode erfolgt in der `start()`- bzw. der `stop()`-Methode der `Activator`-Klasse, so dass der Gruß grundsätzlich beim Start bzw. beim Stoppen des Bundles ausgegeben wird. Zusätzlich zum »Hello World« geben wir in unserem Beispiel den symbolischen Namen des Bundles aus (vgl. Listing 3-1).

Listing 3-1

Der `BundleActivator`

```
package org.osgibook.helloworld;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        greet("Hello OSGi-World from bundle %s!",
            context.getBundle().getSymbolicName());
    }

    public void stop(BundleContext context) throws Exception {
        greet("Goodbye OSGi-World from bundle %s!",
            context.getBundle().getSymbolicName());
    }
}
```

```

protected void greet(String msg, Object... args) {
    String message = String.format(msg, args);
    System.out.println(message);
}
}

```

Schritt 3: Starten der OSGi Service Platform

Um das neue Bundle innerhalb des Equinox-OSGi-Frameworks auszuführen, müssen Sie zunächst eine neue Run-Konfiguration anlegen. Dazu öffnen Sie den Run-Dialog (**Run** -> **Run...**) und erzeugen eine neue OSGi Framework Konfiguration mit dem Namen »Hello World Example« (vgl. Abb. 3–5).

*Anlegen einer
Run-Konfiguration*

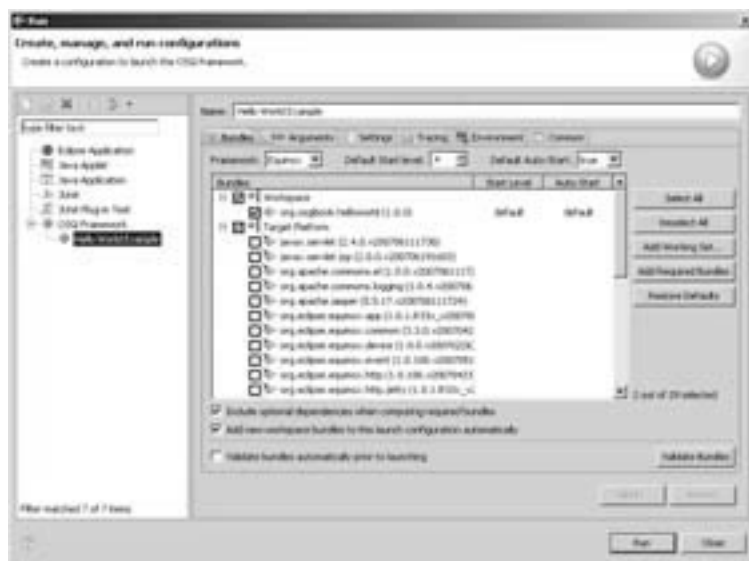


Abb. 3–5
Die »Hello World«-
Run-Konfiguration

Bitte achten Sie darauf, dass in der Liste der ausgewählten Plug-ins nur das Bundle `org.osgi.framework` und das benötigte Bundle `org.eclipse.osgi` ausgewählt ist. Deselektieren Sie dazu alle ausgewählten Bundles (**Deselect All**), um anschließend die beiden Bundles auszuwählen.

Führen Sie die erstellte Run-Konfiguration nun aus, indem Sie den **Run**-Button aktivieren. Auf der Konsole erscheint wie erwartet die »Hello World«-Ausgabe:

*Ausführen der
Run-Konfiguration*

```
osgi> Hello OSGi World from bundle org.osgi.example.helloworld
```

Damit haben Sie Ihr erstes OSGi Bundle entwickelt. In den folgenden Unterkapiteln werden wir die zugrunde liegenden Konzepte detailliert betrachten.

3.2 Bundles im Überblick

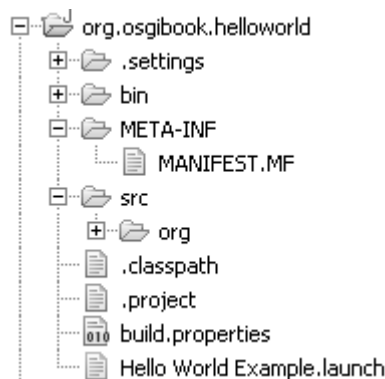
Ein Bundle ist eine fachlich oder technisch zusammenhängende Einheit von Klassen und Ressourcen, die unabhängig im OSGi Framework installiert und deinstalliert werden kann. Technisch gesehen ist ein Bundle ein Java-Archiv (Jar-Datei), das zusätzliche Informationen in Form des Bundle Manifests enthält. In Eclipse Equinox können Bundles auch in »entpackter« Form, also als Verzeichnisstruktur, installiert werden.

Bundles und Eclipse-
Plug-in-Projekte

Innerhalb der Eclipse IDE werden Bundles immer als *Plug-in-Projekte* entwickelt. Plug-in-Projekte enthalten das Bundle Manifest sowie alle zum Bundle gehörenden Sourcen und Ressourcen. Die Struktur eines Plug-in-Projektes (Abb. 3–6) weicht in einigen wesentlichen Punkten von der Struktur eines Bundles ab:

- **Verzeichnisstruktur vs. Jar-Archiv:** Plug-in-Projekte liegen als einfache Verzeichnisse im Dateisystem vor, während Bundles in Form von Jar-Archiven existieren.
- **Source- und Output-Verzeichnisse:** Die Sourcen eines Plug-in-Projektes sind gewöhnlich in Source-Verzeichnissen (z.B. src), die kompilierten Klassen analog in Output-Verzeichnissen (z.B. bin) enthalten. Demgegenüber enthält ein Bundle in Form eines Jar-Archives die kompilierten Klassen im Normalfall direkt im Wurzelverzeichnis. Die Sourcen sind i.d.R. nicht im Bundle enthalten.
- **Projektinformationen:** Ein Plug-in-Projekt enthält eine Reihe von Projektinformationen, die bspw. in der .classpath- und der .project-Datei im Wurzelverzeichnis des Projektes abgelegt sind. Diese Dateien sind in Bundles nicht enthalten.

Abb. 3–6
Struktur eines
Plug-in-Projektes



Die Überführung eines Plug-in-Projektes in ein Bundle erfolgt, indem Sie ein Plug-in-Projekt explizit aus der IDE exportieren. Wir werden dieses Thema im Detail in Kapitel 11 behandeln.

Um die Entwicklung von Bundles in Form von Plug-in-Projekten möglichst komfortabel zu gestalten und lange Turn-around-Zyklen zu vermeiden, unterstützt Eclipse Equinox neben der Verwendung von Bundles auch die direkte Verwendung von Plug-in-Projekten: Sie können also Eclipse-Plug-in-Projekte direkt im OSGi Framework installieren und starten, ohne dass diese vorher in Bundles konvertiert werden müssten. Dies stellt zur Entwicklungszeit eine erhebliche Vereinfachung dar, da das Umbauen von Plug-in-Projekten zu Bundles eine gewisse Zeit benötigt und so die Entwicklung ausbremst. Durch die direkte Verwendung von Plug-in-Projekten in Eclipse Equinox zur Entwicklungszeit entfällt dieser Schritt.

*Installation von
Plug-in-Projekten in
Eclipse Equinox*

3.3 Das Bundle Manifest

Im Verzeichnis META-INF unseres Beispielprojektes befindet sich das Bundle Manifest, das die im Plug-in-Project-Wizard spezifizierten Einträge enthält. Über das Bundle Manifest stellt ein Bundle zusätzliche deskriptive Informationen über sich selbst bereit, die vom OSGi Framework ausgewertet und verarbeitet werden können.

3.3.1 Der Plug-in Manifest Editor

Mit einem Doppelklick können Sie das Bundle Manifest im *Plug-in Manifest Editor* öffnen und bearbeiten (siehe Abb. 3-7).



Abb. 3-7
*Der Plug-in
Manifest Editor*

Der Plug-in Manifest Editor ist ein Bestandteil des Plugin Development Environments (PDE) der Eclipse IDE und stellt einen komfortablen Editor zur Bearbeitung des Bundle Manifests zur Verfügung.

3.3.2 Das Bundle Manifest in textueller Form

Wenn Sie im Plug-in Manifest Editor den Reiter »MANIFEST.MF« auswählen, dann können Sie das Bundle Manifest mit dem verwendeten Bundle Manifest Header in einer übersichtlichen Textform betrachten (siehe Listing 3–2).

Listing 3–2

Das Bundle Manifest
in textueller Form

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Hello World Bundle
Bundle-SymbolicName: org.osgibook.helloworld
Bundle-Version: 1.0.0
Bundle-Activator: org.osgibook.helloworld.Activator
Import-Package: org.osgi.framework;version="1.4.0"
Bundle-ClassPath: .
```

Die möglichen Bundle Manifest Header und ihre jeweilige Bedeutung sind in der OSGi-Core-Spezifikation [OSCS07] innerhalb des Kapitels 3 (*Modules*) spezifiziert. Die meisten der dort definierten Header sind optional und ggf. mit einem Default-Wert belegt. Lediglich der symbolische Name eines Bundles (`Bundle-SymbolicName`), der zusammen mit der Version des Bundles die eindeutige Identifikation des Bundles innerhalb des Frameworks ermöglicht, muss im Bundle Manifest angegeben werden.

Syntax

Viele Manifest-Header-Werte besitzen eine Syntax, die dem folgenden Aufbau entspricht (vgl. bspw. `Import-Package-Header` in Listing 3–2):

```
header ::= clause ( ',' clause ) *
clause ::= path ( ';' path ) *
         ( ';' parameter ) *
```

Ein Parameter kann dabei entweder eine *Direktive* oder ein *Attribut* sein. Direktiven besitzen eine definierte Semantik für das Framework und werden mit »:=« zugewiesen. Attribute hingegen werden benutzt, um Zuordnungen und Vergleiche zu realisieren, und mit »=« zugewiesen. In Listing 3–2 wird z.B. dem Attribut `version` des zu importierenden Packages `org.osgi.framework` der Wert »1.4.0« zugewiesen, um das Package näher zu beschreiben.

3.3.3 Bedeutung der verwendeten Bundle Manifest Header

Die Bedeutungen der in unserem Beispiel verwendeten Bundle Manifest Header sind in der nachfolgenden Tabelle (Tabelle 3–1) aufgelistet. Der Manifest-Header `Manifest-Version` ist Bestandteil der Jar File Specification von SUN [JARSPEC] und spezifiziert die Version der Manifest-Datei. Wir werden diesen Header hier nicht weiter betrachten.

Bundle Manifest Header	Pflicht	Beschreibung
Bundle-ManifestVersion	nein	Definiert die Version der OSGi-Spezifikation, auf die sich das Bundle bezieht. Für Release-4-konforme Bundles muss die Manifest-Version den Wert '2' besitzen. Der Default-Wert ist '1'. Die Bundle-Manifest-Version wird beim Anlegen eines Plug-in-Projektes automatisch auf den Wert 2 gesetzt und sollte von Ihnen nicht geändert werden.
Bundle-Name	nein	Legt einen sprechenden, menschenlesbaren Namen fest, der das Bundle beschreibt und den Sie frei wählen können.
Bundle-SymbolicName	ja	Der symbolische Name ermöglicht – zusammen mit der Bundle-Version – die eindeutige Identifikation eines Bundles innerhalb des Frameworks. Die OSGi-Spezifikation empfiehlt, dass der symbolische Name der Reverse-Domainname-Konvention (z.B. <code>org.osgi.book.xyz</code>) genügt.
Bundle-Version	nein	Spezifiziert die Version eines Bundles. Die Default-Version lautet 0.0.0 (siehe Unterkapitel 5.5).
Bundle-Activator	nein	Der Name einer Klasse, deren <code>start()</code> - bzw. <code>stop()</code> -Methoden ausgeführt werden, wenn das Bundle gestartet bzw. gestoppt wird (siehe Unterkapitel 3.4).
Import-Package	nein	Die importierten Packages des Bundles, die zu dessen Ausführung benötigt werden (siehe Unterkapitel 5.3).
Bundle-Classpath	nein	Definiert eine kommaseparierte Liste mit Jar-Dateien oder Verzeichnissen innerhalb des Bundles, die Klassen und Ressourcen enthalten (siehe Unterkapitel 3.6).

Tab. 3–1

Verwendete Bundle Manifest Header

3.4 Der Bundle-Aktivator

3.4.1 Das Interface BundleActivator

Mithilfe eines Bundle-Aktivators kann ein Bundle beim Start und beim Stopp beliebige Operationen ausführen. Der Bundle-Aktivator muss das Interface `org.osgi.framework.BundleActivator` implementieren. Dieses Interface definiert die vom Bundle-Entwickler zu implementierenden Methoden, die beim Start und beim Stopp eines Bundles vom Framework aufgerufen werden (vgl. Listing 3–3).

Listing 3–3
Das Interface BundleActivator

```
package org.osgi.framework;

public interface BundleActivator {
    public void start(BundleContext context) throws Exception;
    public void stop(BundleContext context) throws Exception;
}
```

Die Aktivatorklasse muss einen öffentlichen Konstruktor ohne Parameter (also den Default-Konstruktor) implementieren, damit das OSGi Framework ein Exemplar dieser Klasse mit der Methode `Class.newInstance()` erzeugen kann.

Der Header Bundle-Activator Spezifiziert wird der Aktivator durch den Bundle-Activator-Manifest-Header, wobei pro Bundle nur maximal ein Aktivator definiert werden kann. Der Aufbau des Bundle Manifest Headers entspricht der folgenden Form:

```
Bundle-Activator: classname
```

wobei *classname* der vollqualifizierte Java-Klassenname ist.

Package-Konventionen Der Bundle-Aktivator ist in der Regel kein Bestandteil der öffentlichen Schnittstelle eines Bundles. Sie sollten deshalb dafür sorgen, dass Ihre Activator-Klasse in einem nichtöffentlichen Package liegt, das von Ihrem Bundle nicht exportiert wird (siehe dazu auch Unterkapitel 5.2).

3.5 Der Bundle-Kontext

3.5.1 Das Interface BundleContext

Das Interface `BundleContext` stellt die Schnittstelle zur Verfügung, über die ein Bundle auf das OSGi Framework zugreifen kann, und wird vom OSGi Framework implementiert (vgl. Listing 3–4).

```
package org.osgi.framework;

import java.io.File;
import java.io.InputStream;
import java.util.Dictionary;
```

```
public interface BundleContext {
```

Installation von und Zugriff auf Bundles

```
public Bundle getBundle();
public Bundle installBundle(String location)
    throws BundleException;
public Bundle installBundle(String location, InputStream input)
    throws BundleException;
public Bundle getBundle(long id);
public Bundle[] getBundles();
```

An- und Abmelden von Listnern

```
public void addServiceListener(ServiceListener listener,
    String filter) throws InvalidSyntaxException;
public void addServiceListener(ServiceListener listener);
public void removeServiceListener(ServiceListener listener);
public void addBundleListener(BundleListener listener);
public void removeBundleListener(BundleListener listener);
public void addFrameworkListener(FrameworkListener listener);
public void removeFrameworkListener(FrameworkListener listener);
```

An- und Abmelden eigener sowie Zugriff auf fremde Services

```
public ServiceRegistration registerService(String[] clazzes,
    Object service, Dictionary properties);
public ServiceRegistration registerService(String clazz,
    Object service, Dictionary properties);
public ServiceReference[] getServiceReferences(String clazz,
    String filter) throws InvalidSyntaxException;
public ServiceReference[] getAllServiceReferences(String clazz,
    String filter) throws InvalidSyntaxException;
public ServiceReference getServiceReference(String clazz);
public Object getService(ServiceReference reference);
public boolean ungetService(ServiceReference reference);
```

Weitere Aufgaben

```
public String getProperty(String key);
public File getDataFile(String filename);
public Filter createFilter(String filter)
    throws InvalidSyntaxException;
}
```

Listing 3-4

Das Interface

BundleContext

Der Bundle-Kontext wird dem Bundle in der `start()`- bzw. in der `stop()`-Methode des Aktivators übergeben und kann dort gespeichert werden, um einen Zugriff auf den Bundle-Kontext während der Laufzeit des Bundles zu ermöglichen.

Die Kommunikation mit dem Framework ist ausschließlich über den Bundle-Kontext möglich, eine direkte Repräsentation des OSGi Frameworks gibt es in der OSGi-Spezifikation nicht. Der Bundle-Kontext ist somit eine zentrale Schnittstelle innerhalb des OSGi Frameworks und wird Ihnen im weiteren Verlauf des Buches häufig wiederbegegnen.

Über den Bundle-Kontext können Sie neue Bundles im Framework installieren und auf im System installierte Bundles zugreifen. Sie können Listener zur Verarbeitung von OSGi-Events anmelden, eigene Services an- und abmelden und bestehende Services abfragen. Darüber hinaus besitzt das Interface `BundleContext` Methoden, die bspw. das Auslesen von Properties oder das Erstellen von Service-Filtern erlauben.

3.6 Der Bundle-Klassenpfad

Für gewöhnlich enthält ein Bundle Klassen und Ressourcen, die direkt im Wurzelverzeichnis des Bundles liegen. Falls Sie jedoch innerhalb des Bundles Klassen oder Ressourcen verwenden, die außerhalb dieser Verzeichnisse liegen (bspw. in einer weiteren enthaltenen Jar-Datei), müssen Sie den Klassenpfad Ihres Bundles entsprechend anpassen.

*Der Header
Bundle-Classpath*

Der Bundle-Klassenpfad legt fest, wie das OSGi Framework innerhalb eines Bundles nach Klassen sucht. Sie können den Bundle-Klassenpfad im Bundle Manifest über den OSGi-Header `Bundle-Classpath` spezifizieren. Der `Bundle-Classpath`-Header erwartet eine kommaseparierte Liste mit den relativen Pfaden der Jar-Dateien bzw. der Verzeichnisse innerhalb eines Bundles, in denen nach Klassen und Ressourcen gesucht werden soll. Dabei wird das Wurzelverzeichnis eines Bundles durch einen Punkt (`».«`) gekennzeichnet.

Beispiel

Wenn Sie in Ihrem Bundle bspw. die »Simple Logging Facade for Java« [SLF4J] für die Ausgabe der Grußbotschaften verwenden möchten, können Sie deren Jar-Dateien `slf4j-api-1.4.3.jar` und `slf4j-simple-1.4.3.jar` in Ihr Bundle aufnehmen und den Klassenpfad entsprechend setzen:

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

In Abb. 3–8 ist der resultierende Klassenpfad des »Hello World«-Bundles grau unterlegt dargestellt.

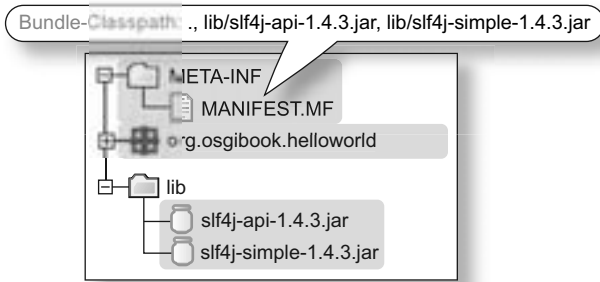


Abb. 3-8

Der Bundle-Klassenpfad

Alle Einträge im Bundle-Classpath müssen relativ zum Root-Verzeichnis Ihres Bundles angegeben werden. Die Einträge dürfen sich außerdem nur auf Verzeichnisse und Jar-Dateien beziehen, die sich innerhalb Ihres Bundles befinden. Ressourcen außerhalb eines Bundles können nicht referenziert werden. Enthält Ihr Klassenpfad Einträge, die nicht innerhalb des Bundles oder eines seiner Fragment-Bundles (vgl. Kapitel 8) existieren, werden diese vom Framework ignoriert.

3.7 Zusammenfassung

Dieses Kapitel hat Ihnen einen ersten Eindruck darüber vermittelt, wie Sie innerhalb der Eclipse IDE Bundles in Form von Plug-in-Projekten implementieren und in einem OSGi Framework ausführen können. Schon mit dem hier vorgestellten, einfachen Beispiel haben Sie eine Reihe von grundlegenden Konzepten von Bundles kennengelernt:

- Mit dem *Bundle Manifest* werden innerhalb eines Bundles zusätzliche deskriptive Informationen angegeben, die vom OSGi Framework ausgelesen und interpretiert werden können. Viele der in der OSGi-Spezifikation beschriebenen Manifest-Header werden wir im weiteren Verlauf des Buches detailliert vorstellen und erläutern.
- Um innerhalb eines Bundles zusätzliche Verzeichnisse oder Jar-Dateien verfügbar zu machen, können Sie den lokalen Klassenpfad eines Bundles über den Manifest-Header `Bundle-Classpath` anpassen.
- Mit dem *Bundle-Aktivator* haben Sie die Möglichkeit, beim Start und beim Stopp eines Bundles beliebige Operationen auszuführen. Der Bundle-Aktivator muss das Interface `org.osgi.framework.BundleActivator` implementieren und über den Manifest-Header `Bundle-Activator` spezifiziert werden.
- Der `BundleContext`, der als Parameter in der `start()`- und in der `stop()`-Methode des Bundle-Aktivators übergeben wird, erlaubt Ihnen den Zugriff auf das OSGi Framework. Über ihn können neue Bundles im System installiert, diverse Listener registriert und deregistriert sowie Services angemeldet und abgefragt werden.