

NetBeans  
.org

Heiko Böck

# NetBeans Platform 6

Rich-Client-Entwicklung mit Java

- Einführung in NetBeans Platform 6
- Entwurf, Entwicklung, Verteilung
- Eclipse-Applikationen migrieren

Galileo Computing

## Auf einen Blick

1	Einführung .....	19
2	Aufbau der NetBeans Plattform .....	25
3	Das Module System .....	35
4	Aktionen .....	69
5	Anwendungsaufbau .....	87
6	Das Lookup-Konzept .....	127
7	Datenverwaltung und -repräsentation .....	149
8	Grafische Komponenten erstellen .....	183
9	Standard-Komponenten einsetzen und erweitern .....	233
10	Internationalisierung und Lokalisierung .....	273
11	Anwendung erstellen, anpassen und ausliefern .....	283
12	Update einer NetBeans Plattform-Anwendung .....	289
13	Persistenz .....	301
14	Web Services .....	341
15	Die NetBeans IDE erweitern .....	349
16	Von Eclipse zu NetBeans .....	361
17	Tipps und Tricks .....	371
18	Beispielprojekt: MP3-Manager .....	385
A	Anhang .....	429

# Inhalt

Geleitwort .....	13
Vorwort .....	15

## **1 Einführung ..... 19**

1.1 Was ist ein Rich-Client? .....	19
1.2 Was ist eine Rich-Client-Plattform? .....	20
1.3 Vorteile einer Rich-Client-Plattform .....	21
1.4 Eigenschaften der NetBeans Plattform .....	22

## **2 Aufbau der NetBeans Plattform ..... 25**

2.1 NetBeans Plattform Architektur .....	25
2.2 NetBeans Plattform Distribution .....	28
2.3 NetBeans Runtime Container .....	30
2.4 NetBeans Classloader System .....	32
2.4.1 Module Classloader .....	33
2.4.2 System Classloader .....	33
2.4.3 Original Classloader .....	34

## **3 Das Module System ..... 35**

3.1 Überblick .....	35
3.2 Struktur eines Moduls .....	36
3.3 Module konfigurieren und integrieren .....	37
3.3.1 Konfiguration .....	37
3.3.2 Manifest .....	38
3.3.3 Layer .....	44
3.4 Module erstellen .....	50
3.5 Versionen und Abhängigkeiten .....	54
3.5.1 Versionierung .....	54
3.5.2 Definition von Abhängigkeiten .....	56
3.6 Lebenszyklus .....	59
3.7 Module Registry .....	62
3.8 Bibliotheken verwenden .....	63
3.8.1 Library Wrapper Module .....	64
3.8.2 Bibliothek einem Modul hinzufügen .....	66

<b>4</b>	<b>Aktionen .....</b>	<b>69</b>
4.1	Überblick .....	69
4.2	Aktionsklassen erstellen .....	70
4.2.1	CallableSystemAction .....	70
4.2.2	CallbackSystemAction .....	73
4.2.3	CookieAction .....	76
4.2.4	Allgemeine kontextabhängige Aktionsklasse .....	80
4.3	Aktionen registrieren .....	83
4.4	Shortcuts und Mnemonics .....	84
<b>5</b>	<b>Anwendungsaufbau .....</b>	<b>87</b>
5.1	Überblick .....	87
5.2	Menubar .....	88
5.2.1	Funktionsweise .....	88
5.2.2	Menü und Menüeintrag erstellen und hinzufügen .....	88
5.2.3	Separator einfügen .....	90
5.2.4	Vorhandene Menüeinträge ausblenden .....	91
5.2.5	Eigene Menubar erstellen .....	92
5.3	Toolbar .....	92
5.3.1	Toolbars erstellen .....	92
5.3.2	Toolbar-Konfigurationen .....	93
5.3.3	Anpassung durch den Benutzer .....	95
5.3.4	Eigene Toolbars erstellen .....	95
5.3.5	Eigene Steuerelemente verwenden .....	96
5.4	Window System .....	97
5.4.1	Einführung .....	97
5.4.2	Konfiguration .....	98
5.4.3	Fenster – Top Component .....	100
5.4.4	Docking Container – Mode .....	109
5.4.5	Gruppieren von Fenstern – Top Component Group .....	113
5.4.6	Verwaltung – Window Manager .....	116
5.5	Statusbar .....	117
5.5.1	Statusbar verwenden .....	117
5.5.2	Statusbar erweitern .....	118
5.6	Progressbar .....	119
5.6.1	Fortschritt einzelner Aufgaben anzeigen .....	119
5.6.2	Fortschritt von mehreren zusammengehörenden Aufgaben anzeigen .....	122
5.6.3	Progressbar in eigene Komponente integrieren .....	125

<b>6</b>	<b>Das Lookup-Konzept .....</b>	<b>127</b>
6.1	Funktionsweise .....	127
6.2	Services und Extension Points .....	128
6.2.1	Schnittstelle des Services definieren .....	129
6.2.2	Lose Bereitstellung eines Services .....	129
6.2.3	Verschiedene Service Provider bereitstellen .....	131
6.2.4	Verfügbarkeit des Services sicherstellen .....	132
6.3	Globale Services .....	132
6.4	Service Provider registrieren .....	135
6.4.1	Service-Provider-Configuration-Datei .....	135
6.4.2	Services Folder .....	137
6.5	Intermodulkommunikation .....	138
6.6	Java Service Loader .....	146
<b>7</b>	<b>Datenverwaltung und -repräsentation .....</b>	<b>149</b>
7.1	Überblick .....	149
7.2	File Systems API .....	150
7.2.1	Überblick .....	150
7.2.2	Operationen .....	151
7.3	Data Systems API .....	155
7.3.1	Überblick .....	155
7.3.2	Data Object .....	157
7.3.3	Data Loader .....	164
7.4	Nodes API .....	168
7.4.1	Node-Container .....	170
7.4.2	Node- und Children-Klassen implementieren .....	173
7.5	Explorer API .....	178
<b>8</b>	<b>Grafische Komponenten erstellen .....</b>	<b>183</b>
8.1	Dialoge .....	183
8.1.1	Standarddialoge .....	183
8.1.2	Eigene Dialoge .....	187
8.1.3	Wizards .....	189
8.2	Multi Views .....	204
8.3	Visual Library .....	209
8.3.1	Aufbau der Visual Library API .....	209
8.3.2	Die Widget-Klassen .....	210
8.3.3	Ereignisse und Aktionen .....	215

8.3.4	Die Scene – das Wurzelement .....	221
8.3.5	ObjectScene – Model-View Relation .....	224
8.3.6	Graphen .....	226
8.3.7	VMD – Visual Mobile Designer .....	230

## **9 Standard-Komponenten einsetzen und erweitern ..... 233**

9.1	Hilfesystem .....	233
9.1.1	Erstellen und Hinzufügen eines Helpsets .....	233
9.1.2	Links in Hilfeseiten einfügen .....	236
9.1.3	Kontextsensitive Hilfe .....	238
9.1.4	Öffnen des Hilfesystems .....	240
9.2	Output Window .....	240
9.3	Navigator .....	243
9.4	Properties .....	248
9.4.1	Eigenschaften bereitstellen .....	249
9.4.2	Benutzerdefinierter Eigenschaftseditor .....	252
9.5	Optionen und Einstellungen .....	254
9.5.1	Optionspanel erstellen und bedienen .....	254
9.5.2	Einstellungen verwalten .....	260
9.6	Palette .....	262
9.6.1	Palette-Einträge über die Layer-Datei definieren und hinzufügen .....	263
9.6.2	Eine Palette mit eigenen Nodes aufbauen .....	265

## **10 Internationalisierung und Lokalisierung ..... 273**

10.1	Textkonstanten in Quelltexten .....	273
10.2	Textkonstanten in der Manifest-Datei .....	275
10.3	Internationalisierung von Hilfeseiten .....	276
10.4	Andere Ressourcen internationalisieren .....	278
10.4.1	Grafiken .....	278
10.4.2	Beliebige Dateien .....	278
10.4.3	Verzeichnisse und Dateien .....	278
10.5	Verwaltung und Bereitstellung von lokalisierten Ressourcen .....	279

## **11 Anwendung erstellen, anpassen und ausliefern ..... 283**

11.1	Anwendung erstellen .....	283
11.2	Konfiguration und Anpassung .....	285
11.2.1	Anpassung von Plattform-Modulen .....	285

11.2.2	Launcher anpassen .....	286
11.3	Distribution erstellen .....	287
11.3.1	Auslieferung als ZIP-Distribution .....	287
11.3.2	Distribution für Java Web Start .....	288
11.3.3	Mac OS X-Applikation .....	288
<b>12 Update einer NetBeans Platform-Anwendung .....</b>		<b>289</b>
12.1	Der Auto-Update-Service .....	289
12.2	Das NBM-Paket .....	290
12.3	Update-Center .....	294
12.4	Bereitstellung eines Sprachpakets .....	295
12.5	Konfiguration und Installation auf der Client-Seite .....	296
12.5.1	Neues Update-Center .....	298
12.5.2	Automatische Installation von Updates .....	299
<b>13 Persistenz .....</b>		<b>301</b>
13.1	Java DB .....	301
13.1.1	Einbinden der Java DB .....	301
13.1.2	Treiber registrieren .....	302
13.1.3	Erstellen und verwenden einer Datenbank .....	302
13.1.4	Datenbank herunterfahren .....	304
13.1.5	Datenbank entwickeln mithilfe der NetBeans IDE .....	305
13.1.6	Beispielanwendung .....	308
13.2	Hibernate .....	321
13.2.1	Einbinden der Hibernate-Bibliotheken .....	322
13.2.2	Struktur der Beispielanwendung .....	324
13.2.3	Hibernate konfigurieren .....	325
13.2.4	Objekte auf Relationen abbilden .....	326
13.2.5	SessionFactory und Sessions .....	329
13.2.6	Objekte speichern und laden .....	330
13.3	Java Persistence API .....	332
13.3.1	Hibernate und die Java Persistence API .....	333
13.3.2	Java Persistence-Konfiguration .....	334
13.3.3	Entitätsklassen .....	335
13.3.4	EntityManagerFactory und EntityManager .....	337
13.3.5	Objekte speichern und laden .....	339

<b>14 Web Services .....</b>	<b>341</b>
14.1 Web Service Client erstellen .....	341
14.2 Web Service verwenden .....	343
<b>15 Die NetBeans IDE erweitern .....</b>	<b>349</b>
15.1 Palette .....	349
15.1.1 Palette-Einträge definieren und registrieren .....	350
15.1.2 PaletteController erstellen und registrieren .....	352
15.1.3 Bestehende Palette erweitern .....	354
15.2 Task List API .....	354
<b>16 Von Eclipse zu NetBeans .....</b>	<b>361</b>
16.1 NetBeans IDE .....	361
16.1.1 Wo finde ich was? .....	361
16.1.2 Bedienung .....	362
16.2 Vom Eclipse-Plugin zum NetBeans-Modul .....	362
16.2.1 Terminologie und Wizards .....	362
16.2.2 Plugin-Lebenszyklus und seine Ereignisse .....	363
16.2.3 Plugin-Informationen .....	365
16.2.4 Images .....	366
16.2.5 Ressourcen .....	366
16.2.6 Einstellungen .....	367
16.2.7 Anwendungs-Lebenszyklus .....	368
16.2.8 Views und Editors .....	369
<b>17 Tipps und Tricks .....</b>	<b>371</b>
17.1 Asynchrones Initialisieren von grafischen Komponenten .....	371
17.2 Undo/Redo .....	374
17.3 Beenden der Anwendung/Lebenszyklus der Anwendung .....	377
17.4 Warm-Up Tasks .....	378
17.5 System Tray .....	379
17.6 Desktop .....	380
17.7 Logging .....	381
17.7.1 Logger .....	381
17.7.2 LogManager .....	382
17.7.3 Konfiguration .....	382
17.7.4 Fehlermeldungen .....	384

<b>18 Beispielprojekt: MP3-Manager</b>	<b>385</b>
18.1 Entwurf	385
18.2 Module Suite erstellen	388
18.3 MP3-Unterstützung	389
18.3.1 JMF-Modul erstellen	389
18.3.2 MP3-Plugin registrieren	389
18.3.3 MP3 File Type	390
18.4 ID3-Support	393
18.4.1 ID3 API	394
18.4.2 ID3-Editor	396
18.5 Media Library	399
18.6 Services	401
18.7 MP3-Player	401
18.7.1 Service Interface	401
18.7.2 Service Provider	405
18.7.3 Wiedergabe von MP3-Dateien	409
18.7.4 Benutzeroberfläche	410
18.8 Playlist	414
18.8.1 Node View	414
18.8.2 Node Container	415
18.8.3 Top Component	416
18.8.4 Drag & Drop	422
18.8.5 Speichern der Playlist	424
<b>A Anhang</b>	<b>429</b>
A.1 Die wichtigsten Platform Extension Points	429
A.2 Die DTDs der wichtigsten Konfigurationsdateien	430
A.2.1 Mode-Definition	430
A.2.2 Zuordnung von Top Component zu Mode	433
A.2.3 Top Component-Gruppendefinition	434
A.2.4 Zuordnung von Top Component zu Gruppe	435
A.2.5 Toolbar-Definition und -Konfiguration	435
A.2.6 Palette Item-Definition	436
Index	437

## Geleitwort

The NetBeans platform has, for a long time, been hidden beneath Sun Microsystem's NetBeans IDE, and the fact that the infrastructure could be reused, and the value of doing so, was a well-kept secret. Despite this, fearless programmers, individually as well as within the context of working for companies, successfully built applications on top of it. They learned the ins and outs of the NetBeans platform by reading the sources of NetBeans IDE. Because NetBeans IDE is a modular development environment, they were able to extract the parts relevant to them and discard the rest. Doing so was possible because, before creating NetBeans IDE, the original architects were far more interested in creating a reusable framework for application development. In fact, the infrastructure concerns, that is, the concerns regarding the »plumbing« of an application, such as a windowing system, were of much greater interest to them than the NetBeans IDE they built on top of it. However, as stated earlier, for many years the power and reusability of the NetBeans IDE platform was not well known. And even those who knew about it had a hard time getting to know it well enough to be able to perform useful functions with it.

This all changed with NetBeans IDE 5.0. With this version, for the first time, a set of tools were provided to create modules on top of the NetBeans platform. What many users found particularly helpful was the inclusion of a set of wizards that generate the source structure of NetBeans modules, as well as stubs for the major NetBeans APIs. Not only was there, as before, a NetBeans platform to provide the infrastructure of Swing desktop applications, but now there were also templates to provide the basic outline of NetBeans modules. Together, these allowed programmers to truly focus on their tasks. In short, NetBeans IDE had become the SDK of the NetBeans platform. From that point on, the number of NetBeans modules that were created grew very quickly, and the »buzz« around these modules further increased their number. Beyond this, the dual function of NetBeans modules has become increasingly apparent to many people. They serve both to extend the functionality of NetBeans IDE, and to extend the functionality of any other application written on top of the NetBeans platform.

Geleitwort

I am extremely happy to welcome this new book on the NetBeans platform into the world. A lot of work has been put into it, and it explains many of the finer details of the NetBeans platform. I even discovered several pieces of information that had not yet been documented anywhere else! With this, I wish you all the best reading this book and exploring the NetBeans platform; there are so many features, it is unlikely that you will need them all. However, this book will definitely give you a good perspective on how to approach the NetBeans platform, and how to make the best use of it for your own applications.

Best wishes and have fun with NetBeans,

**Geertjan Wielenga**

NetBeans Team

## Vorwort

Rich-Client-Plattformen haben in den vergangenen Monaten und Jahren stetig an Bedeutung gewonnen. Allen voran sind die NetBeans Plattform und die Eclipse RCP zu nennen. Die Entwicklung dieser beiden Plattformen wurde vor allem durch die beiden IDEs von NetBeans und Eclipse vorangetrieben, die auf den entsprechenden Plattformen basieren und selbst eine Rich-Client-Anwendung darstellen. Während die Eclipse RCP mit SWT und JFace vermehrt auf eigene Ansätze und Konzepte setzt, basiert die NetBeans Plattform vollständig auf der Java API mit AWT und Swing und integriert die Konzepte der Java Standard Edition.

Rich-Client-Plattformen werden in erster Linie aufgrund stetig steigender Anforderungen an Anwendungen und ihre Architektur und Flexibilität eingesetzt. Ein großer Faktor ist dabei die erhöhte Produktivität und die Flexibilität, ein Produkt für einen bestimmten Einsatzzweck ausstatten und an einen Markt anpassen zu können. Dies spielt natürlich gerade bei größeren und professionellen Anwendungen eine große Rolle.

Meiner Meinung nach lohnt sich dank der umfassenden Unterstützung, wie sie die NetBeans IDE in der Tat bietet, bei clientseitigen Anwendungen fast aller Größenordnungen der Einsatz einer Rich-Client-Plattform – und sei es eine noch so kleine Anwendung. Schon allein durch die Ausführungsumgebung und erst recht durch die zahlreichen APIs, die praktische Lösungen für die bei der Client-Anwendungsentwicklung häufig auftretenden Problemstellungen und Herausforderungen bieten. Diese Lösungen sind dabei sehr anwendungs- und praxisnah und erhöhen die Produktivität in hohem Maße.

Diese Annahme fußt allerdings auf einer Grundbedingung: dem versierten Umgang mit den Konzepten der Rich-Client-Plattform. Dem Anwendungsentwickler sollten zumindest die wichtigsten Kernpunkte vertraut sein, denn nur so können die realen Vorteile in gesteigerte Produktivität und erhöhte Qualität der Software umgesetzt werden.

Die vermeintliche Komplexität der Plattform-Konzepte ist einer der Hauptgründe, warum sich Rich-Client-Plattformen noch nicht als Quasi-Standard bei der Client-Anwendungsentwicklung durchgesetzt haben. Tatsächlich hat ein Entwickler zu Beginn den Eindruck, vor einem »Berg« von APIs und Konzepten zu stehen. Sind diese jedoch erst einmal erlernt bzw. verstanden, so ergeben sich

immense – und zu Beginn vielleicht nicht erahnte – Synergien und Erleichterungen, welche die anfängliche Lernphase schnell wieder ausgleichen.

Betrachtet man die aktuellen Entwicklungen der Java-Plattform im Bereich der Client-Anwendungen, wie etwa die verbesserte Desktop-Integration oder die gesteigerte Performance, und blickt man zudem noch in die Zukunft, so sieht man eindeutig, dass genau die Richtung eingeschlagen wird, in die sich Rich-Client-Plattformen schon seit geraumer Zeit bewegen. Wenn ich dabei von Zukunft spreche, denke ich vor allem an das *Java Module System* (JSR 277), das mit Java 7 Einzug in die Java-Welt halten könnte und damit vielleicht endgültig dem Plattform-Gedanken bei der Client-Anwendungsentwicklung zum Durchbruch verhilft.

Damit möchte ich auf die NetBeans IDE und die NetBeans Plattform überleiten. Die NetBeans IDE setzt mit ihren umfangreichen, hilfreichen und vor allem leicht zu bedienenden Wizards alles daran, dem Entwickler den Einstieg und auch den täglichen Einsatz zu erleichtern. Dabei spielt natürlich auch eine große Rolle, dass sämtliche APIs und Konzepte auf den APIs und Konzepten der Java Standard Edition aufsetzen. Dies macht den Umgang mit ihnen schnell vertraut und ermöglicht auch eine Wiederverwendung von bestehenden Komponenten.

### **Aufbau des Buches**

Dieses Buch richtet sich an Java-Entwickler, die Client-Anwendungen auf Basis der NetBeans Plattform entwickeln wollen. Dabei werden keinerlei Vorkenntnisse im Bereich Rich-Client-Plattformen vorausgesetzt. Vorrangiges Ziel dieses Buchs ist die praxisnahe Vermittlung der Grundideen und Funktionalitäten der NetBeans Plattform. Dabei will ich Ihnen die sehr gute Unterstützung der NetBeans IDE für die Entwicklungsphase Ihrer Anwendung sowie die Schnittstellen und Vorteile der NetBeans Plattform näher bringen und Sie so für den weiterführenden Einsatz motivieren. Dabei wird sich Ihnen (hoffentlich) die Frage aufdrängen, warum Sie Ihre bisherigen Anwendungen nicht auf Basis einer Rich-Client-Plattform entwickelt haben. Oder aber es kommt Ihnen die Erkenntnis, von welchen zahlreichen Vorteilen Sie in der Vergangenheit hätten profitieren können.

Zunächst gehe ich auf die Begriffsdefinition eines Rich-Clients und einer Rich-Client-Plattform ein und zeige Ihnen anschließend anhand der Eigenschaften einer Rich-Client-Plattform im Allgemeinen und der Vorteile der NetBeans Plattform im Speziellen, warum sich ihr Einsatz für Sie lohnt.

Darauf folgt der konzeptionelle Aufbau der NetBeans Plattform, und Sie erfahren, wie eine Rich-Client-Anwendung aufgebaut wird, wie Sie Ihre Anwendungslogik in die Plattform integrieren und wie Sie die NetBeans-Konzepte und -Komponen-

ten effizient einsetzen können. Darüber hinaus zeige ich Ihnen, wie Sie Ihre Anwendung benutzer- und länderspezifisch anpassen, sie verteilen und im Feld aktualisieren können.

Ein wichtiger Punkt bei der Rich-Client-Entwicklung ist die Persistenz. Auch darauf gehe ich ein und erläutere Ihnen in diesem Kontext den Einsatz der *Java Persistence API* in Verbindung mit der objektrelationalen Brücke *Hibernate* und der Client-Persistenz-Lösung *Java DB*.

Außerdem möchte ich Ihnen die neuen Möglichkeiten der Desktop-Integration von *Java 6* näher bringen. Die mächtige *Visual Library*, die mit NetBeans 6 nun zum Umfang der NetBeans Plattform gehört, wollen wir uns ebenso genauer ansehen wie auch das aktuelle Thema *Web Services*.

Schließlich soll ein Kapitel eine Brücke zwischen Eclipse und NetBeans schlagen und damit den Anwendern von *Eclipse* den Einstieg in NetBeans und die Migration von bestehenden Anwendungen auf die NetBeans Plattform erleichtern.

Die einzelnen Kapitel sind so gestaltet, dass sie weitestgehend unabhängig voneinander sind, um Ihnen so die Möglichkeit zu geben, direkt in einzelne Kapitel einzusteigen und Ihnen ein optimales Handbuch für die Entwicklung von Rich-Client-Anwendungen auf Basis der NetBeans Plattform zur Verfügung zu stellen. Um die Kapitel übersichtlich zu halten und den Direkteinstieg zu ermöglichen, werden die Erklärungen innerhalb des Buchs gezielt durch kleine Beispiele ohne größeren Bezug zu einer Gesamtanwendung erläutert. Am Ende des Buchs werde ich Ihnen dann die Erstellung einer kompletten Rich-Client-Anwendung – vom Entwurf über die Erstellung des Grundgerüsts bis hin zur Implementierung der Anwendungslogik – tutorialartig am Beispiel eines *MP3 Managers* darstellen. In diese Anwendung werden wir unter anderem das *Java Media Framework* wie auch das *Java DB*-Datenbanksystem integrieren.

Sämtliche Beispiele und Erläuterungen in diesem Buch basieren auf *Java 6* und NetBeans 6, wobei in den meisten Fällen auch *Java 5* ausreichend ist. Das Java Development Kit (JDK 6) können Sie sich unter <http://java.sun.com> und NetBeans 6 unter <http://netbeans.org> herunterladen. Die Beispiele finden Sie als komplettes NetBeans-Projekt auf der beiliegenden CD.

Vorwort

### **Danksagung**

An dieser Stelle möchte ich den fachlichen Gutachtern Ruth Kusterer und vor allem Geertjan Wielenga danken, die sich bereitwillig zur Verfügung gestellt haben. Vielen Dank auch an meinen Lektor Marc Czesnik für die sehr angenehme und reibungslose Zusammenarbeit.

Dank gebührt auch den zahlreichen Menschen, dabei vor allem meiner Familie, die mich immer wieder in meiner Arbeit bestärkt und ihr volles Vertrauen in mich gesetzt haben.

**Heiko Böck**

*Das Lookup-Konzept ist ein ebenso bedeutendes wie einfaches Konzept, das an sehr vielen Stellen innerhalb der NetBeans Plattform Anwendung findet. Es ist universell nutzbar, sodass Sie es effizient in Ihren Anwendungsmodulen einsetzen können. Wie dieses Konzept funktioniert und wie die typischen Anwendungsfälle aussehen, zeige ich Ihnen in diesem Kapitel.*

## 6 Das Lookup-Konzept

### 6.1 Funktionsweise

Das *Lookup* ist eine zentrale Komponente und ein weit verbreitetes Konzept innerhalb der NetBeans Plattform. Seine Aufgabe ist es, Instanzen von Objekten zu verwalten. Vereinfacht gesagt ist das Lookup eine Map, bei der als Schlüssel `Class`-Objekte verwendet werden, und die Werte sind Instanzen der `Class`-Objekte.

Der wesentliche Gedanke, der hinter dem Lookup steht, ist die *Entkopplung von Komponenten*. Es bietet also eine Art Kommunikation zwischen Modulen, die in einem komponentenbasierten System, wie es Anwendungen auf Basis der NetBeans Plattform sind, eine wichtige und zentrale Rolle spielt. Module können über das Lookup sowohl Objekte bereitstellen als auch Objekte aufsuchen und verwenden.

Der Vorteil des Lookups ist dessen Typsicherheit, die dadurch erreicht wird, dass `Class`-Objekte statt Strings als Schlüssel verwendet werden. Somit wird bereits durch den Schlüssel festgelegt, von welchem Typ die gelieferte Instanz ist. Es kann somit nur eine Instanz angefordert werden, deren Typ dem Modul bekannt ist, wodurch die Anwendung robuster wird. Fehler wie etwa eine `ClassCastException` können dann nicht auftreten. Das Lookup ist auch in der Lage, für einen Schlüssel, also einen bestimmten Typ, mehrere Instanzen zu verwalten. Diese zentrale Verwaltung von konkreten Instanzen kann für verschiedene Zwecke genutzt werden. Das Lookup kann zum Auffinden von Service Providern genutzt werden, wobei dabei *deklaratives Hinzufügen* und *Lazy Loading* von Instanzen ermöglicht wird. Außerdem können über das Lookup Instanzen von einem Modul zum anderen »weitergereicht« werden, ohne dass sich die Module kennen müssen. Somit wird eine Art Intermodulkommunikation ermöglicht. Auch kontextsensitive Aktionen lassen sich mittels der Lookup-Komponente realisieren.

Um ein nahe liegendes Missverständnis auszuräumen: Innerhalb einer Anwendung kann es auch mehrere Lookups geben. Das zumeist verwendete Lookup ist ein globales, das von der NetBeans Plattform standardmäßig zur Verfügung gestellt wird. Weiterhin gibt es Komponenten, wie z. B. Top Components, die ihr eigenes Lookup haben, dies sind dann *lokale Lookups*. Wie Sie in Abschnitt 6.5 sehen werden, können Sie auch eigene Lookups erstellen und Ihre eigenen Komponenten mit einem Lookup ausstatten.

Das Lookup-Konzept ist ein recht einfaches und dennoch sehr effizientes und praktisches Konzept. Haben Sie das Prinzip erst einmal verstanden, können Sie es in vielen Bereichen einsetzen. In den nachfolgenden Abschnitten will ich Ihnen die Verwendung des Lookups in seinen Hauptanwendungsgebieten zeigen.

## 6.2 Services und Extension Points

Eine der Hauptanwendungen des Lookups ist das Aufsuchen und Bereitstellen von Services. Dabei übernimmt das Lookup die Aufgabe des *dynamischen Service Locators* und ermöglicht somit die Trennung von *Service Interface* und *Service Provider*. Ein Modul kann also eine Funktionalität nutzen, ohne dessen Implementation zu kennen. Dadurch entsteht eine lose Kopplung zwischen den Modulen.

Mithilfe des Lookups und eines Service Interfaces lassen sich auch sehr schön *Extension Points* für grafische Komponenten erstellen. Ein gutes Beispiel dafür ist die *NetBeans Statusbar*. Diese definiert das Interface `StatusLineElementProvider`. Über dieses Interface und die *Service Provider Registration* lässt sich die Statusbar beliebig um eigene Komponenten erweitern (ein Beispiel dazu finden Sie in Abschnitt 5.5.2), ohne dass die Statusbar die Komponenten kennen oder in Abhängigkeit zu diesen stehen muss.

Damit Services dynamisch und flexibel bereitgestellt und auch ausgetauscht werden können, werden diese dem Lookup deklarativ anstatt direkt im Quelltext hinzugefügt. Dazu stehen Ihnen zwei Möglichkeiten zur Verfügung. Sie können Ihre Implementierung eines Services mit einer *Service-Provider-Configuration*-Datei im Verzeichnis `META-INF/services` hinzufügen oder aber über die Layer-Datei Ihres Moduls. Diese beiden Varianten der Registrierung zeige ich Ihnen in Abschnitt 6.4.

Die NetBeans Plattform stellt ein globales Lookup zur Verfügung, das Sie mit der statischen Methode `Lookup.getDefault()` erhalten. Dieses globale Lookup ist in der Lage, Services zu finden und zu liefern, die über eine der beiden deklarativen Möglichkeiten registriert wurden. Dabei ist es auch möglich, dass für einen Service mehrere Implementationen registriert werden. Dadurch, dass Services dekla-

rativ registriert werden, können Sie vom Lookup genau dann erstellt werden, wenn Sie das erste Mal angefordert werden. Genau dies versteht man unter *Lazy Loading*.

Damit Sie sich das Prinzip der Bereitstellung und Verwendung von Services besser vorstellen können und einen praktischen Bezug bekommen, wollen wir nun ein Beispiel anhand einer Suchliste für MP3-Dateien erstellen.

### 6.2.1 Schnittstelle des Services definieren

Modul A sei ein Modul, das dem Benutzer eine Oberfläche zur Verfügung stellt, auf der er mit bestimmten Suchkriterien nach MP3 Dateien suchen kann. Die Suchresultate werden in einer Liste angezeigt. Damit nun dieses Modul unabhängig vom verwendeten Suchalgorithmus bleibt und wir die Möglichkeit haben, mehrere Suchvarianten dynamisch zur Verfügung zu stellen und diese auch austauschen können, spezifizieren wir in Modul A lediglich ein *Service Interface* `Mp3Finder`, das die Schnittstelle für die Suche von MP3-Dateien festlegt. Die eigentliche Suchlogik wird in einem separaten Modul B implementiert und deklarativ zur Verfügung gestellt.

### 6.2.2 Lose Bereitstellung eines Services

Modul B sei ein *Service Provider* und stelle eine Implementation des Interfaces `Mp3Finder` bereit. Hier im Beispiel nehmen wir an, dass dieses Modul in einer Datenbank nach MP3-Dateien sucht. Auf diese Weise können jetzt beliebig viele Service Provider bereitgestellt werden. Diese können sich dabei in diesem Modul oder auch in verschiedenen Modulen befinden. Damit die Service Provider-Klasse `Mp3DatabaseFinder` das Interface `Mp3Finder` von Modul A implementieren kann, muss Modul B eine Abhängigkeit von Modul A definieren. Modul A, also die Suchliste, benötigt aber keine Abhängigkeit zu Modul B, da das Lookup den Service Provider anhand des Interfaces liefert. Somit ist Modul A vollkommen unabhängig von der Implementation des Services und kann diesen transparent verwenden.

In Modul A spezifizieren wir das Service Interface `Mp3Finder` und implementieren die Benutzerschnittstelle zum Suchen und Darstellen von MP3-Dateien. Um einen Service Provider zu erhalten, müssen Sie dem Lookup lediglich ein `Class`-Objekt des Interfaces `Mp3Finder` übergeben, das Ihnen dann eine Instanz liefert. Das Interface `Mp3Finder` stellt somit einen *Extension Point* von Modul A dar, für den Sie beliebige andere Module registrieren können.

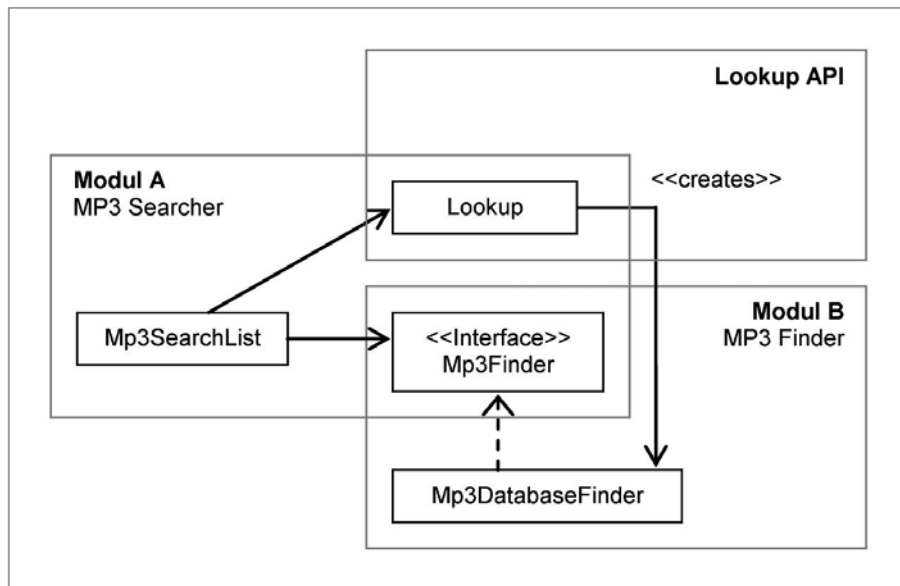


Abbildung 6.1 Service-Lookup-Muster

```
public interface Mp3Finder {
    public List<Mp3FileObject> find(String search);
}

public class Mp3SearchList {
    public void doSearch(String search) {
        Mp3Finder finder =
            Lookup.getDefault().lookup(Mp3Finder.class);
        List<Mp3FileObject> list = finder.find(search);
    }
}
```

Listing 6.1 Modul A – MP3 Searcher

Modul B stellt einen Service Provider bereit, der nach MP3-Dateien in einer Datenbank sucht. Dazu implementiert dieser das Interface `Mp3Finder`, das von Modul A spezifiziert wurde. Modul B ist somit eine *Extension* von Modul A am *Extension Point* `Mp3Finder`.

```
public class Mp3DatabaseFinder implements Mp3Finder {
    public List<Mp3FileObject> find(String search) {
```

```

        // search in database for mp3 files
    }
}

```

**Listing 6.2** Modul B – MP3 Finder

Nachdem wir einen Service Provider erstellt haben, müssen wir diesen nun noch registrieren, damit er vom Lookup gefunden werden kann. Dazu erstellen wir im Verzeichnis *META-INF/services* eine Datei mit dem vollständigen Namen des Interfaces (*com.galileo.netbeans.modulea.Mp3Finder*) und schreiben in die Datei den Namen der Implementation:

```
com.galileo.netbeans.moduleb.Mp3DatabaseFinder
```

### 6.2.3 Verschiedene Service Provider bereitstellen

Jetzt wäre es natürlich wünschenswert, wenn wir mehrere verschiedene MP3-Suchdienste zur Verfügung stellen und verwenden könnten. Dies kann denkbar einfach realisiert werden. Dazu implementieren wir einfach weitere Service Provider, die das Interface *Mp3Finder* implementieren, z. B.:

```

public class Mp3FilesystemFinder implements Mp3Finder {
    public List<Mp3FileObject> find(String search) {
        // search in local filesystem for mp3 files
    }
}

```

Den Namen dieses Services fügen wir der zuvor angelegten *Service Provider-Configuration*-Datei im Verzeichnis *META-INF/services* hinzu:

```
com.galileo.netbeans.moduleb.Mp3FilesystemFinder
```

Damit wir alle Services verwenden können, müssen wir jetzt noch das Auffinden der Services über das Lookup anpassen. Anstatt mit der *lookup()*-Methode nur nach einem Service zu suchen, lassen wir uns mit der Methode *lookupAll()* alle verfügbaren Services anzeigen und rufen dann die *find()*-Methode aller gelieferten Services auf:

```

public class Mp3SearchList {
    public void doSearch(String search) {
        Collection<? extends Mp3Finder> finder =
            Lookup.getDefault().lookupAll(Mp3Finder.class);
        List<Mp3FileObject> list = new ArrayList<Mp3FileObject>();
    }
}

```

```

        for(Mp3Finder f : finder) {
            list.addAll(f.find(search));
        }
    }
}

```

### 6.2.4 Verfügbarkeit des Services sicherstellen

Nun würde ein Suchmodul dem Benutzer natürlich wenig nutzen, wenn kein Service verfügbar ist, der nach MP3 Dateien sucht. Damit Modul A sicherstellen kann, dass mindestens ein Service verfügbar ist, stellt das NetBeans Module System die zwei Attribute `OpenIDE-Module-Provides` und `OpenIDE-Module-Requires` zur Verfügung, mit denen in der Manifest-Datei eines Moduls ein Service bereitgestellt und verlangt werden kann (diese und weitere Attribute der Manifest-Datei werden ausführlich in Abschnitt 3.3.2 beschrieben).

In der Manifest-Datei von Modul A verlangen wir das Vorhandensein von mindestens einem `Mp3Finder`-Service Provider durch folgenden Eintrag:

**OpenIDE-Module-Requires:** `com.galileo.netbeans.modulea.Mp3Finder`

Damit das Module System beim Laden der Module weiß, dass Modul B den Service `Mp3Finder` zur Verfügung stellt, fügen wird der Manifest-Datei von Modul B folgenden Eintrag hinzu:

**OpenIDE-Module-Provides:** `com.galileo.netbeans.modulea.Mp3Finder`

Sollte kein Modul diesen Eintrag in der Manifest-Datei aufweisen, ist also kein Service Provider vorhanden, würde das Module System einen Fehler melden und Modul A nicht laden.

## 6.3 Globale Services

*Globale Services*, also *Services*, die von mehreren Modulen verwendet werden und von denen meistens nur ein Service Provider vorhanden ist, werden typischerweise als abstrakte (Singleton-)Klassen implementiert. So können sie ihre Implementierungen selbst verwalten und falls kein Service Provider vorhanden ist, können sie eine eigene Standard-Implementation (meist als Inner Class) liefern. Dies hat den großen Vorteil, dass der Benutzer stets eine Referenz auf einen Service Provider und nie einen `null`-Wert zurückgeliefert bekommt.

Als Beispiel wäre ein MP3-Player-Service denkbar, der von verschiedenen anderen Modulen, wie z. B. einer Suchliste oder einer Playlist, verwendet werden soll. Die Implementation des Players soll aber austauschbar sein.

```

public abstract class Mp3Player {

    public abstract void play(Mp3FileObject mp3);
    public abstract void stop();

    public static Mp3Player getDefault() {

        Mp3Player player =
            Lookup.getDefault().lookup(Mp3Player.class);

        if(player == null) {
            player = new DefaultMp3Player();
        }

        return(player);
    }

    private static class DefaultMp3Player extends Mp3Player {

        public void play(Mp3FileObject mp3) {
            // send file to an external player or
            // provide own player implementation or
            // show a message that no player is available
        }

        public void stop() {}
    }
}

```

**Listing 6.3** MP3-Player als globaler Service in Modul MP3 Services

Dieser Service, der als abstrakte Klasse ausgeführt ist, spezifiziert zum einen durch die abstrakten Methoden seine Schnittstellen und stellt zugleich mit der statischen Methode `getDefault()` einen Service Provider zur Verfügung. Dies hat vor allem den großen Vorteil, dass die Anwendungen, die den Service nutzen, sich nicht um die Lookup API kümmern müssen. So bleibt die Anwendungslogik schlanker und unabhängiger.

Diese abstrakte Klasse sollte sich idealerweise in einem Modul befinden, das zum Standardumfang der Anwendung gehört (hier im Beispiel ist dies das Modul MP3 Services). Der Service Provider, also die Klassen, die das tatsächliche Abspielen der MP3-Datei innerhalb der Anwendung realisieren, können in einem separaten Modul gekapselt werden. Dies ist hier im Beispiel die Klasse `MyMp3Player`, deren Grundgerüst wir nachfolgend erstellen und die wir Modul C hinzufügen wollen.

```

public class MyMp3Player extends Mp3Player {

```

## 6 | Das Lookup-Konzept

```
public void play(Mp3FileObject mp3) {  
    // play file  
}  
  
public void stop() {  
    // stop player  
}  
}
```

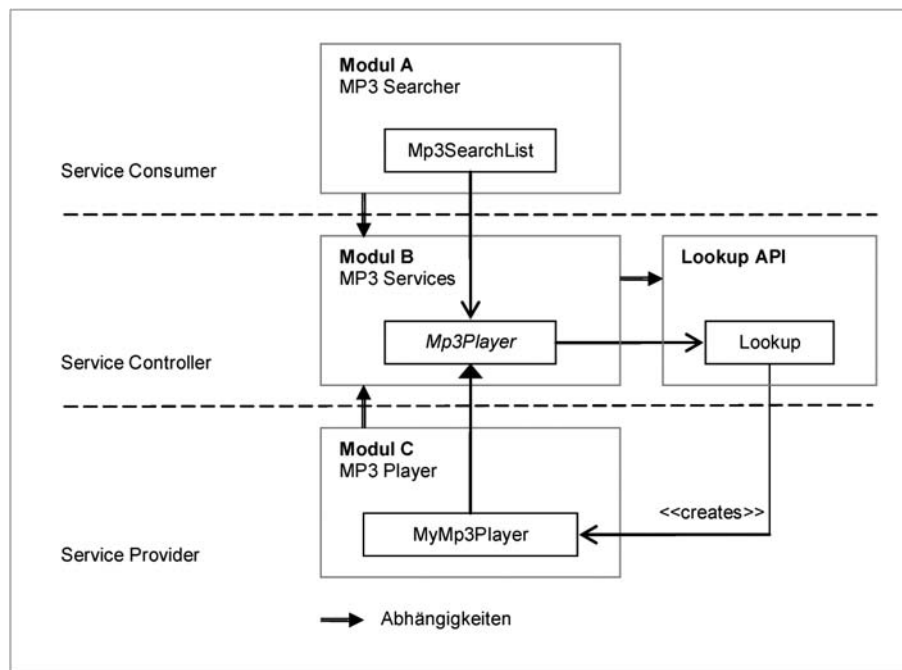
**Listing 6.4** MP3-Player Service Provider in Modul MP3 Player

Der Service Provider `MyMp3Player` muss nun noch registriert werden. Das kann – wie Sie bereits im vorhergehenden Abschnitt gesehen haben – über eine Service-Provider-Configuration-Datei im Verzeichnis `META-INF/services` erfolgen (siehe Abschnitt 6.4.1), die folgendermaßen aussieht:

```
com.galileo.netbeans.mp3services.Mp3Player
```

```
com.galileo.netbeans.mp3player.MyMp3Player
```

Die Zusammenhänge und die Abhängigkeiten der Module sind in Abbildung 6.2 dargestellt:



**Abbildung 6.2** Abhängigkeiten und Zusammenhänge von globalem Service, Service Provider und Anwendungsmodul

Gute Beispiele für globale Services innerhalb der NetBeans Platform sind z. B. auch die Klassen `StatusDisplayer` oder `IOProvider`. Die Klasse `IOProvider` bietet Zugriff auf das *Output*-Fenster. Der Service Provider, der also die Ausgaben tatsächlich auf das Output-Fenster schreibt, befindet sich in der separaten Klasse `NbIOProvider` in einem eigenen Modul. Ist dieses Modul vorhanden und der Service Provider registriert, wird dessen Implementation von der Methode `IOProvider.getDefault()` geliefert. Ist das Modul hingegen nicht verfügbar, wird die Standard-Implementation geliefert, welche die Ausgaben auf die Standardausgabe (`System.out` und `System.err`) schreibt.

## 6.4 Service Provider registrieren

Damit Service Provider einem System dynamisch und flexibel hinzugefügt werden können, also auch dann noch, wenn eine Anwendung bereits ausgeliefert wurde, und erst dann geladen werden, wenn sie auch wirklich benötigt werden, werden diese deklarativ, also über Konfigurationsdateien, registriert.

Services, die innerhalb einer auf der NetBeans Platform basierten Anwendung bereitgestellt und über das Lookup verfügbar sein sollen, können auf zwei verschiedene Arten registriert und somit dem System bekannt gemacht werden. Diese beiden Möglichkeiten stelle ich Ihnen nachfolgend vor.

### 6.4.1 Service-Provider-Configuration-Datei

Bevorzugt sollten Service Provider mittels einer *Service-Provider-Configuration-Datei* registriert werden. Diese Variante ist Teil der *Java JAR File Specification*. Eine solche Datei benennt mit ihrem Namen einen Service und listet mit ihrem Inhalt alle Service Provider auf. Diese Datei muss sich im Verzeichnis *META-INF/services* befinden, das im *src/*-Verzeichnis eines Moduls, oder mit anderen Worten, auf dem Klassenpfad eines Moduls, liegen muss.

```
src/META-INF/services/com.galileo.netbeans.module.Mp3Finder
```

```
com.galileo.netbeans.module.Mp3DatabaseFinder
com.galileo.netbeans.module.Mp3FileSystemFinder
```

In diesem Beispiel werden zwei Service Provider für den Service, also das Interface oder die abstrakte Klasse, `Mp3Finder` registriert. Um die so zur Verfügung gestellten Services und Service Provider einzusehen, stellt Ihnen die NetBeans IDE einen *META-INF services Browser* zur Verfügung. Diesen finden Sie in der Projektansicht Ihres Moduls unter **IMPORTANT FILES • META-INF SERVICES**. Dort sehen Sie unter **<EXPORTED SERVICES>** die Services, die von Ihrem Modul regist-

riert wurden. Unter <ALL SERVICES> finden Sie alle Services und Service Provider, die innerhalb der Module Suite zur Verfügung stehen, also auch die von Plattform-Modulen und von anderen eigenen Modulen. In dieser Ansicht können Sie sich einen Überblick verschaffen, welche Services von der Plattform bereitgestellt werden. Sie können hier zudem direkt über das Kontextmenü für einen bestimmten Service einen Service Provider hinzufügen.

Das globale Lookup, also das Standard-Lookup kann die im Verzeichnis *META-INF/services* vorhandenen Services auffinden und die jeweiligen Provider instanziiieren. Dazu ist es wichtig, dass jeder Service Provider einen Standard-Konstruktor besitzt, damit diese vom Lookup erzeugt werden können.

Auf Basis dieser Spezifikation der Service-Provider-Configuration-Datei stellt die NetBeans Plattform zwei Erweiterungen zur Verfügung, mit denen es möglich ist, vorhandene Service Provider zu entfernen und die Reihenfolge der Service Provider zu beeinflussen. Damit die Datei aber weiterhin mit der Java-Spezifikation konform ist, wird den Erweiterungen das Kommentarzeichen »#« vorangestellt. Somit werden diese Zeilen von der JDK-Implementation ignoriert.

#### Entfernen eines Service Providers

Es ist möglich einen Service Provider, der von einem anderen Modul registriert wurde, zu entfernen. Dies kann z. B. dazu verwendet werden, die Standard-Implementation eines Services der NetBeans Plattform durch eine eigene Implementation zu ersetzen.

Einen Service Provider können Sie durch folgenden Eintrag in Ihrer Service-Provider-Configuration-Datei entfernen. Dabei können Sie dann gleichzeitig Ihren eigenen Provider angeben:

```
# remove the other implementation (by prefixing the line with #-)
#-org.netbeans.core.ServiceImpl

# provide my own
com.galileo.netbeans.module.MyServiceImpl
```

#### Reihenfolge der Service Provider

Die Reihenfolge, in der die Service Provider vom Lookup zurückgeliefert werden, kann durch eine Positionsangabe für jeden einzelnen Providereintrag gesteuert werden.

Dies ist z. B. notwendig für die bestimmte Platzierung von zusätzlichen Komponenten in der Statusbar (siehe dazu Abschnitt 5.5.2) oder aber auch um sicherzustellen, dass die eigene Implementation vor der Plattformimplementation aufge-

rufen wird. Es sind auch negative Werte erlaubt. Dabei sorgt die NetBeans Plattform dafür, dass eine Instanz mit kleinerer Nummer vor der mit einer größeren geliefert wird. Dazu wird der Service-Provider-Configuration-Datei Folgendes hinzugefügt:

```
com.galileo.netbeans.module.MyServiceImpl
#position=20

com.galileo.netbeans.module.MyImportantServiceImpl
#position=10
```

Dabei ist es empfehlenswert, die Positionswerte – wie hier im Beispiel – in größeren Abständen zu vergeben. Das spätere Einfügen weiterer Implementierungen ist so leichter möglich.

### 6.4.2 Services Folder

Eine weitere Möglichkeit, einen Service Provider bereitzustellen, ist die Registrierung über den `Services`-Folder in der Layer-Datei eines Moduls.

```
<folder name="Services">
  <folder name="Mp3Services">
    <file name="com-galileo-netbeans-module-
      Mp3DatabaseFinder.instance">
      <attr name="instanceOf" stringvalue="com-galileo-
        netbeans-module-Mp3Finder"/>
    </file>
  </folder>
</folder>
```

**Listing 6.5** Registrieren von Service Providern in der Layer-Datei

Wird ein Service Provider über das Standard-Lookup verlangt, sucht dieses im `Services`-Folder und dessen Unterverzeichnissen nach Instanzen, die mit dem verlangten Service Interface übereinstimmen. Das heißt, die `Services` können beliebig in eigenen Foldern, wie hier im Beispiel im Folder `Mp3Services`, gruppiert werden.

Im Unterschied zur Registrierung über eine Service Provider-Configuration-Datei, bei der der Service Provider stets einen Standard-Konstruktor besitzen muss, können Sie hier mit dem `instanceCreate`-Attribut eine statische Methode angeben, mit der der Service Provider erzeugt werden kann. Angenommen der

zuvor erstellte Provider `Mp3DatabaseFinder` verfügt über eine statische Methode `getDefault()`, die eine Instanz zurückliefert, kann folgendes Attribut hinzugefügt werden:

```
<attr name="instanceCreate" methodvalue="com.galileo.netbeans.
    module.Mp3DatabaseFinder.getDefault"/>
```

Somit wird der Service Provider nicht durch den Standard-Konstruktor, sondern durch die statische Methode `getDefault()` erzeugt (ausführliche Informationen zu diesem Attribut und den damit in Verbindung stehenden `.instance`-Dateien finden Sie in Abschnitt 3.3.3).

Auch bei der Registrierung über den `Services`-Folder haben Sie die Möglichkeit, vorhandene Service Provider zu entfernen und die Reihenfolge der Provider festzulegen. In beiden Fällen sind das Standardereignisse der Layer-Datei. Entfernen können Sie einen Service Provider durch das Anfügen von `_hidden` an seinen Namen, wie das z.B. auch bei Menüeinträgen vorgenommen wird (siehe Abschnitt 5.2.4).

```
<file name="com-galileo-netbeans-module-ServImp.instance_hidden">
```

Die Reihenfolge, in der die Service Provider geliefert werden, können Sie durch die Verwendung des `position`-Attributs festlegen, so wie auch bei anderen Einträgen der Layer Datei vorgegangen wird (siehe dazu Abschnitt 3.3.3).

```
<folder name="Services">
    <file name="com-galileo-netbeans-module-ServImp.instance">
        <attr name="position" intvalue="10"/>
    </file>
    <file name="com-galileo-netbeans-module-ServImp2.instance">
        <attr name="position" intvalue="20"/>
    </file>
</folder>
```

In diesem Beispiel wird durch die `position`-Attribute festgelegt, dass der Service Provider `ServImp` vor `ServImp2` ausgegeben wird.

## 6.5 Intermodulkommunikation

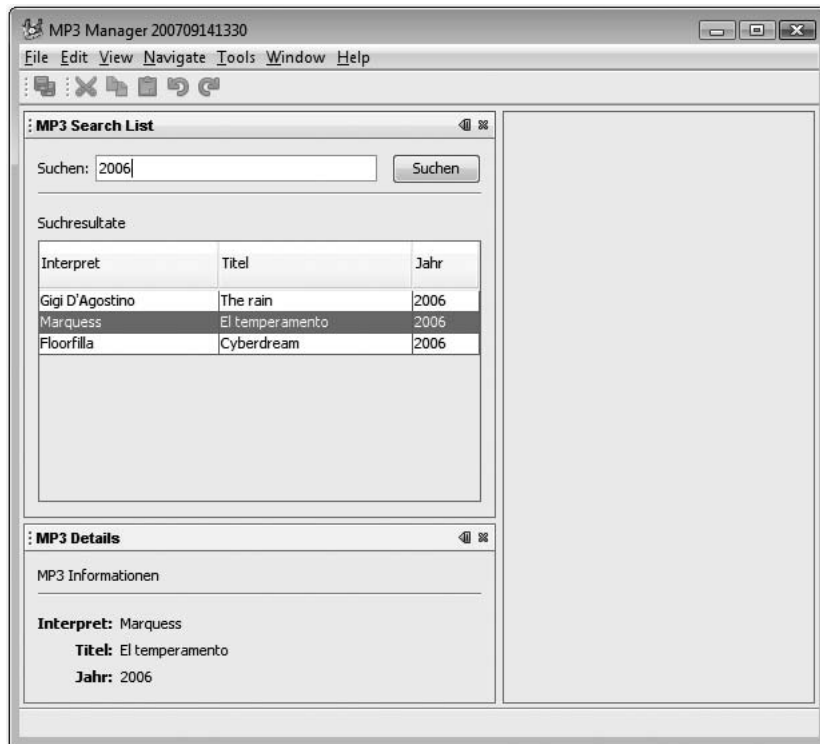
Neben dem globalen Lookup, das von der NetBeans Plattform zur Verfügung gestellt wird und Zugriff auf alle registrierten Services bietet, besteht die Möglichkeit, dass Sie Ihre eigenen Komponenten mit einem lokalen Lookup ausstatten. Die Lookup API bietet neben einer Factory zum Erzeugen eigener Lookups auch

die Möglichkeit, auf Veränderungen in einem Lookup zu reagieren. Mit der Klasse `ProxyLookup` können Sie außerdem einen Proxy für mehrere Lookups erstellen. Diese Funktionalität der Lookup API und SPI wollen wir im Folgenden dazu nutzen, eine Kommunikation zwischen Komponenten verschiedener Module zu realisieren, ohne dass diese in Abhängigkeit zueinander stehen müssen.

Ein typischer Anwendungsfall für die Kommunikation von lose gekoppelten Modulen ist die Darstellung von Detailinformationen eines ausgewählten Objektes, wobei die Auswahl des Objektes und die Darstellung der Informationen von verschiedenen Modulen erfolgen sollen. Als Beispiel können Sie sich wieder eine Liste vorstellen, in der die Suchresultate von MP3-Dateien dargestellt werden. Wählt man nun einen Eintrag in dieser Liste aus, soll der selektierte Eintrag über ein Lookup zur Verfügung gestellt werden, sodass andere auf diesen Eintrag zugreifen und die entsprechenden Informationen darstellen können. Es handelt sich hier also gewissermaßen um ein Observer-Muster. Das Modul, das die Objekte bereitstellt, also die Suchliste, ist das Subject, und das Modul, das die Informationen darstellt, ist ein Observer. Es können somit beliebig viele Module vorhanden sein, die die Daten bzw. Detailinformationen in entsprechender Form darstellen können. Der Vorteil ist wieder die lose Kopplung der Module: Sie sind vollständig unabhängig voneinander. Der einzige gemeinsame Nenner ist das jeweils zur Verfügung gestellte Objekt, dessen Informationen verarbeitet werden sollen. Diese lose Kopplung wird erreicht, indem sich der Observer nicht direkt beim Subject registriert, sondern bei einer Proxy-Komponente.

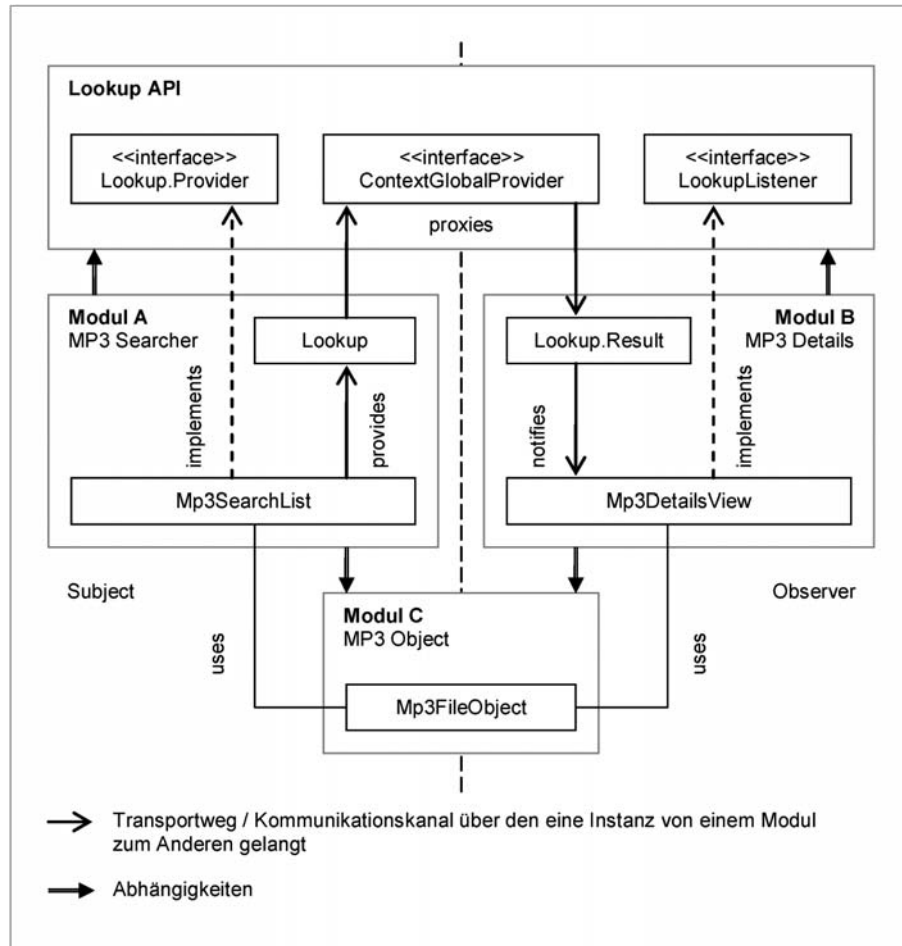
In Abbildung 6.3 ist das Beispiel dargestellt, das wir nachfolgend implementieren werden. Dabei befinden sich die beiden angezeigten Fenster jeweils in einem eigenen Modul und sind nicht voneinander abhängig. Das heißt, die Module könnten beliebig ausgetauscht bzw. neue und andere können hinzugefügt werden.

Die Struktur dieses Konzeptes ist in Abbildung 6.4 dargestellt. Die Klasse `Mp3SearchList` in Modul A stellt eine Liste mit Suchergebnissen dar. Ein Eintrag der Suchergebnisse wird durch die Klasse `Mp3FileObject` repräsentiert, die sich in einem separaten Modul befindet, da diese Klasse gewissermaßen den kleinsten gemeinsamen Nenner aller beteiligten Module darstellt. Wird nun ein Eintrag aus der Liste selektiert, wird diese aktuelle `Mp3FileObject`-Instanz im lokalen Lookup bereitgestellt. Für die Entkopplung von Modul A und B benötigen wir einen zentralen Vermittler, d. h. eine Proxy-Komponente – im Bild durch das Interface `ContextGlobalProvider` dargestellt –, der das lokale Lookup von Modul A Modul B zur Verfügung stellt, das ja die aktuell selektierte Instanz beinhaltet. Damit nun diese zentrale Proxy-Komponente auf das lokale Lookup der Klasse `Mp3SearchList` zugreifen kann, spezifiziert die Lookup API das Interface `Lookup.Provider`. Dieses Interface muss von der Klasse `Mp3SearchList` implementiert werden.



**Abbildung 6.3** Typisches Anwendungsbeispiel, bei dem Daten zwischen zwei Modulen ausgetauscht werden sollen, ohne dass diese voneinander abhängen

Über die Methode `getLookup()` kann dann das lokale Lookup zur Verfügung gestellt werden. Das `Lookup.Provider`-Interface wird von der Klasse `TopComponent`, von der ja alle im NetBeans Window System darstellbaren Fenster abgeleitet werden, so auch die Klasse `Mp3SearchList`, bereits implementiert. Das NetBeans Window System stellt uns praktischerweise eine Instanz der zentralen Proxy-Komponente zur Verfügung, das ist die Klasse `GlobalActionContextImpl`. Diese liefert ein Proxy-Lookup, das Zugriff auf das lokale Lookup der Klasse `TopComponent` hat, auf der gerade der Fokus liegt. An dieses Lookup gelangen wir ganz einfach über die statische Hilfsmethode `Utilitites.actionsGlobalContext()`. Somit müssen wir uns um die `ContextGlobalProvider`-Instanz überhaupt nicht kümmern und bekommen direkt das globale Proxy-Lookup geliefert. Für denjenigen, der an dieser Stelle noch tiefer einsteigen und sich mit diesem Konzept genauer befassen möchte, lohnt sicher auch ein Blick in den Quelltext der genannten Methoden und Klassen.



**Abbildung 6.4** Struktur des Intermodulkommunikationskonzeptes mit einem lokalen Lookup über eine Proxy-Komponente zur Entkopplung von Subject und Observer

Die Klasse `Mp3DetailsView` besorgt sich also Zugriff auf das lokale Lookup von `Mp3SearchList` über den Aufruf von `Utilities.actionsGlobalContext()`. Von diesem globalen Proxy-Lookup erzeugen wir nun ein `Lookup.Result` für den Typ `Mp3FileObject`. Ein Objekt der Klasse `Lookup.Result` stellt eine Untermenge eines Lookups für einen ganz bestimmten Typ zur Verfügung. Der große Vorteil ist dabei, dass auf Änderungen in dieser Untermenge über ein `LookupListener` reagiert werden kann. Meine Komponente wird also benachrichtigt, sobald in `Mp3SearchList` ein anderes `Mp3FileObject` selektiert wurde oder wenn das Fenster von `Mp3SearchList` nicht mehr im Fokus steht und somit z. B. keine MP3-Detailinformationen angezeigt werden sollen.

Nachfolgend sehen Sie die Klassen dieser Beispielanwendung. Dabei sind nur die jeweils wichtigsten Teile der Klassen abgebildet. Das komplette, lauffähige Projekt finden Sie auf der beiliegenden CD.

Zunächst haben wir die Klasse `Mp3SearchList`, die ein Fenster darstellt und deshalb von der Basisklasse `TopComponent` ableitet. Damit wir auf das Selektieren eines Eintrags aus der Ergebnisliste reagieren können, implementieren wir das `ListSelectionListener`-Interface. Als private Datenelemente haben wir ein Datenmodell, das die Daten der Tabelle verwaltet. Zu Demonstrationszwecken habe ich hier ein einfaches Datenmodell angelegt, das in seinem Konstruktor exemplarisch drei Objekte der Klasse `Mp3FileObject` erzeugt und hinzufügt. Diese Daten würden natürlich gewöhnlich vom Suchalgorithmus geliefert werden. Das zweite private Datenelement ist eine `InstanceContent`-Instanz. Über dieses Objekt haben wir die Möglichkeit, den Inhalt unseres lokalen Lookups dynamisch zu verändern. Im Konstruktor von `Mp3SearchList` erstellen wir dann mit der Klasse `AbstractLookup` unser lokales Lookup, dem wir das `InstanceContent`-Objekt übergeben. Mit der Methode `associateLookup()` wird das lokale Lookup gesetzt, sodass es von der `getLookup()`-Methode geliefert wird.

In der Methode `valueChanged()`, die aufgerufen wird, wenn ein Datensatz in der Tabelle selektiert wird, holen wir uns den selektierten Datensatz aus dem Datenmodell, verpacken diesen in eine `Collection` und übergeben diese unserer `InstanceContent`-Instanz, die sich ja im Lookup befindet. Auf diese Weise liegt das selektierte Element stets im lokalen Lookup.

```
public class Mp3SearchList extends TopComponent
    implements ListSelectionListener {

    private Mp3SearchListModel model = new Mp3SearchListModel();
    private InstanceContent content = new InstanceContent();

    private Mp3SearchList() {

        initComponents();
        searchResults.setModel(model);
        searchResults.getSelectionModel().
            addListSelectionListener(this);

        associateLookup(new AbstractLookup(content));
    }

    public void valueChanged(ListSelectionEvent event) {

        if(!event.getValueIsAdjusting()) {
            Mp3FileObject mp3 =
                model.getRow(searchResults.getSelectedRow());
```

```

        content.set(Collections.singleton(mp3), null);
    }
}

```

**Listing 6.6** Mp3SearchList stellt die Suchergebnisse in einer Tabelle dar und fügt den jeweils selektierten Datensatz dem Lookup hinzu.

Das Datenmodell Mp3SearchListModel der Tabelle mit den Suchergebnissen ist hier exemplarisch sehr klein und einfach gehalten. Im Konstruktor werden direkt drei Objekte vom Typ Mp3FileObject erzeugt.

```

public class Mp3SearchListModel extends AbstractTableModel {
    private String[] columns = {"Interpret", "Titel", "Jahr"};
    private Vector<Mp3FileObject> data =
        new Vector<Mp3FileObject>();
    public Mp3SearchListModel() {
        data.add(new Mp3FileObject(
            "Gigi D'Agostino", "The rain", "2006"));
        data.add(new Mp3FileObject(
            "Marquess", "El temperamento", "2006"));
        data.add(new Mp3FileObject(
            "Floorfilla", "Cyberdream", "2006"));
    }
    public Mp3FileObject getRow(int row) {
        return data.get(row);
    }
    public Object getValueAt(int row, int col) {
        Mp3FileObject mp3 = data.get(row);
        switch(col) {
            case 0: return(mp3.getArtist());
            case 1: return(mp3.getTitle());
            case 2: return(mp3.getYear());
        }
        return "";
    }
}

```

**Listing 6.7** Vereinfachtes Datenmodell, das die Daten für die Ergebnisliste verwaltet und bereitstellt.

Die Klasse `Mp3DetailsView` ist das Fenster, das die Informationen des in `Mp3SearchList` selektierten Eintrags darstellt. Damit wir über die Veränderungen im Lookup benachrichtigt werden, wenn also ein anderer Eintrag selektiert wurde, implementieren wir das `LookupListener`-Interface. Als privates Datenelement existiert ein `Lookup.Result`, mit dem wir die Möglichkeit haben, auf Veränderungen eines ganz bestimmten Typs – hier `Mp3FileObject` – im Lookup zu reagieren. Wird das Fenster in der Anwendung geöffnet, wird die Methode `componentOpened()` aufgerufen. In dieser holen wir uns das Lookup der Proxy-Komponente, die stets an das lokale Lookup der gerade aktiven `TopComponent` weiter delegiert. Dies erreichen wir über die Methode `Utilities.actionsGlobalContext()`. Auf dieses Proxy-Lookup erstellen wir nun unser `Lookup.Result` für den Typ `Mp3FileObject` und registrieren darauf einen `LookupListener`. Liegt nun auf einer `TopComponent` der Fokus, die in ihrem lokalen Lookup eine oder mehrere Instanzen von diesem Typ bereithält, wird die Methode `resultChanged()` aufgerufen. In dieser müssen wir dann nur noch die Instanzen auslesen und können die Informationen entsprechend darstellen.

```
public class Mp3DetailsView extends TopComponent
    implements LookupListener {

    private Lookup.Result<Mp3FileObject> result = null;

    private Mp3DetailsView() {

        initComponents();
    }

    public void componentOpened() {

        result = Utilities.actionsGlobalContext().
            lookupResult(Mp3FileObject.class);
        result.addListener(this);
        resultChanged(null);
    }

    public void resultChanged(LookupEvent event) {

        Collection<? extends Mp3FileObject> mp3s =
            result.allInstances();

        if(!mp3s.isEmpty()) {
            Mp3FileObject mp3 = mp3s.iterator().next();

            artist.setText(mp3.getArtist());
            title.setText(mp3.getTitle());
            year.setText(mp3.getYear());
        }
    }
}
```

```

    }
}
}

```

**Listing 6.8** Im Fenster `Mp3DetailsView` werden die Informationen des in `Mp3SearchList` selektierten `Mp3FileObject` angezeigt.

Die Daten, die von `Mp3SearchList` bereitgestellt und von `Mp3DetailsView` dargestellt werden, befinden sich in der Klasse `Mp3FileObject`. Diese Klasse sollte sich für eine optimale Kapselung und Wiederverwendung in einem separaten Modul befinden, wie das hier mit Modul C der Fall ist. Damit die Module A und B Zugriff auf diese Klasse haben, müssen diese eine Abhängigkeit auf Modul C beinhalten. Wird die Klasse `Mp3FileObject` ausschließlich von Modul A bereitgestellt, wäre es auch denkbar, sie in Modul A zu verschieben.

```

public class Mp3FileObject {
    private String artist = new String();
    private String title = new String();
    private String year = new String();

    public Mp3FileObject(String artist,
                        String title,
                        String year) {

        this.artist = artist;
        this.title = title;
        this.year = year;
    }

    public String getArtist() {
        return this.artist;
    }

    public String getTitle() {
        return this.title;
    }

    public String getYear() {
        return this.year;
    }
}

```

**Listing 6.9** `Mp3FileObject` stellt die Daten zur Verfügung

Als Proxy-Komponente verwenden wir hier im Beispiel das von der NetBeans Platform zur Verfügung gestellte globale Proxy-Lookup, das an das lokale Lookup der jeweils aktiven `TopComponent` delegiert. In Abbildung 6.4 wird dies durch das

Interface `ContextGlobalProvider` dargestellt. An die Stelle dieses globalen Proxy-Lookups der Plattform könnte auch problemlos eine eigene Implementation treten. Diese muss lediglich das lokale Lookup, der Komponente, die die Daten bereitstellt, also das Subject, den Observer-Komponenten zugänglich machen.

## 6.6 Java Service Loader

Mit Version 6 stellt die Java API mit der Klasse `ServiceLoader` eine Art Lookup zur Verfügung. Die Klasse `ServiceLoader` kann Service Provider laden, die über das `META-INF/services`-Verzeichnis registriert wurden. Sie entspricht somit in der Funktion dem NetBeans Standard Lookup, das Sie über `Lookup.getDefault()` laden können. Ein *Service Loader* wird stets für einen speziellen Klassentyp über das `Class`-Objekt des Service Interfaces oder der abstrakten Service-Klasse angelegt. Erzeugen können Sie einen Service Loader mit einer statischen Methode. In Abhängigkeit davon, mit welchem Classloader die Service Provider geladen werden sollen, stehen Ihnen drei verschiedene Methoden zur Verfügung.

Standardmäßig werden Service Provider über den Context Classloader des aktuellen Threads geladen. Innerhalb der NetBeans Plattform ist dies der System Classloader (Informationen zum NetBeans Classloader System finden Sie in Abschnitt 2.4), d. h., es können Service Provider aus allen Modulen geladen werden. Dafür erzeugen Sie den Service Loader wie folgt:

```
ServiceLoader<Mp3Finder> s = ServiceLoader.load(Mp3Finder.class);
```

Um Service Provider über einen speziellen Classloader zu laden, also z. B. über den Module Classloader, damit nur Service Provider aus Ihrem eigenen Modul geladen werden, können Sie folgende Methode zum Erzeugen Ihres Service Loaders verwenden.

```
ServiceLoader<Mp3Finder> s = ServiceLoader.load(
    Mp3Finder.class, this.getClass().getClassLoader());
```

Darüber hinaus haben Sie die Möglichkeit, einen Service Loader zu erstellen, der nur installierte Service Provider lädt, d. h. Service Provider, deren JAR-Archiv sich im Verzeichnis `lib/ext` oder im systemweiten, plattformspezifischen Extension-Verzeichnis befindet. Service Provider auf dem Klassenpfad werden ignoriert. Diesen Service Loader erstellen Sie wie folgt:

```
ServiceLoader<Mp3Finder> s = ServiceLoader.loadInstalled(
    Mp3Finder.class);
```

Abrufen können Sie die Service Provider über einen Iterator. Dieser lädt die Provider dann dynamisch, also erst, wenn sie erfragt werden. Die geladenen Provider werden in einem internen Cache gehalten. Der Iterator liefert zunächst die Service Provider aus dem Cache, die durch einen früheren Aufruf bereits geladen wurden, erst dann werden die verbleibenden, noch nicht geladenen Provider ausgegeben. Den internen Cache können Sie mit der Methode `reload()` löschen, somit werden alle Service Provider erneut geladen.

```
Iterator<Mp3Finder> i = s.iterator();  
  
if(i.hasNext()) {  
    Mp3Finder finder = i.next();  
}
```



# Index

@Basic 335  
@Column 336  
@Entity 335  
@Id 335  
@ManyToOne 336

## A

---

Abfragedialog 185  
Abhängigkeit 40, 56  
Abhängigkeiten 26, 33, 35, 40, 54  
AbstractAction 69  
AbstractFileSystem 150  
AbstractLookup 142  
AbstractNode 78, 169, 417  
Action 69  
ActionMap 73  
Action-Performer 73  
    *Registrierung* 75  
Actions 69  
Actions API 73  
ActiveEditorDrop 351  
AggregateProgressFactory 122  
AggregateProgressHandle 122  
Aktionen 69  
    *Always Enabled* 70  
    *Conditionally Enabled* 70  
    *erstellen* 70  
    *globale* 73  
    *Icon* 72  
    *integrieren* 70  
    *kontextabhängige* 73, 80, 160, 163, 409  
    *registrieren* 83  
Aktionsbehandlung 69  
Aktualisierung → Update  
Amazon E-Commerce Service 341  
Amazon ECS API 342  
Anwendung  
    *anpassen* 285  
    *beenden* 377  
    *erstellen* 283  
    *konfigurieren* 285  
Anwendungsaufbau 87  
Anwendungs-Lebenszyklus 368, 377  
Assistent → Wizard

AsyncGUIJob 371  
Asynchrones Initialisieren 371, 378  
Autoload 37

## B

---

BeanNode 169  
BeanTreeView 178, 179  
Benutzeroberfläche 87  
Bibliothek  
    *direkt einbinden* 66  
Bibliotheken 63  
binary-origin 66  
Branding  
    *Modul* 399  
    *Token* 399  
Branding ID 285  
Branding Name 284  
Browser öffnen 381  
Bundle.properties 52

## C

---

CallableSystemAction 70  
CallbackSystemAction 73  
ChangeListener 195, 201  
Children 170, 415  
    *implementieren* 173  
ChoiceView 178  
Class 127  
Classloader  
    *Application* 34  
    *Context* 33  
    *Module* 32  
    *Multi-Parent* 32  
    *Original* 32  
    *System* 32, 33  
Classloader System 26, 32  
CLASSPATH 34  
class-path-extension 66  
Client-Datenbanklösung 301  
Context Classloader 33, 146  
ContextAction 81  
ContextAwareAction 80  
ContextGlobalProvider 139

## Index

ControllerListener 407  
CookieAction 76, 160, 163, 409  
Cookies 76, 155, 158  
    *dynamische* 161  
    *hinzufügen* 159  
    *implementieren* 159  
    *verwenden* 160  
CookieSet 158  
CopyAction 76  
CutAction 76

## D

---

Data Loader 164  
    *Pool* 167  
    *registrieren* 166  
Data Systems API 150, 155  
DataLoader 155, 164, 391  
DataLoaderPool 167  
DataNode 169, 392  
DataObject 155, 157, 390  
    *erstellen* 164  
    *Unterschied zu FileObject* 158  
Datei-Operationen 151  
Dateisysteme 151  
Daten speichern 424  
Datenverwaltung 149  
DeleteAction 76  
Dependency → Abhängigkeit  
Derby 301  
Desktop 380  
DialogDescriptor 187  
DialogDisplayer 184  
Dialoge 183  
    *eigene* 187  
    *erstellen* 319  
    *Login* 187  
    *Standarddialoge* 183  
Dialogs API 183, 419  
Distribution  
    *Java Web Start* 288  
    *Mac OS X* 288  
    *ZIP* 287  
Docking Container 109  
DocumentListener 192  
Drag & Drop 414, 422  
DriverManager 426  
DropTarget 423  
Drucken 381

## E

---

Eager 37  
Eclipse 361  
    *Activator vs Installer* 363  
    *Keymap* 362  
    *Plugin vs Modul* 362  
    *Views und Editors* 369  
EditCookie 78  
Eigenschaften → Properties  
Eingabedialog 186  
Einstellungen 254, 260, 367  
EndOfMediaEvent 407  
EntityManager 335, 337, 338  
EntityManagerFactory 335, 337  
Entkopplung 127, 139  
Entkopplung von Modulen 387  
Explorer API 178, 414  
Explorer Folder 173, 180  
Explorer View 168, 179  
ExplorerManager 179, 417, 418  
ExplorerManager.Provider 179, 418  
ExplorerUtils 179  
Extension 130  
Extension Point 107, 129, 174  
    *Actions* 83, 429  
    *AutoupdateType* 429  
    *benutzerdefinierter* 107, 173, 379  
    *Editors* 353  
    *JavaHelp* 429  
    *Loaders* 166  
    *Menu* 89, 429  
    *MIMEResolver* 429  
    *Navigator/Panels* 246, 429  
    *OptionsDialog* 260, 429  
    *Services* 137, 429  
    *Services/AutoupdateType* 298  
    *Services/JavaHelp* 233  
    *Services/MIMEResolver* 165  
    *Shortcuts* 84, 429  
    *TaskList/Groups* 358, 429  
    *TaskList/Scanners* 358, 430  
    *TaskList/Scopes* 358, 430  
    *Toolbars* 92, 94, 430  
    *WarmUp* 378, 430  
    *Windows2* 98  
    *Windows2/Components* 98, 102, 430  
    *Windows2/Groups* 99, 430  
    *Windows2/Modes* 98, 102, 430

Extension Points 45, 50, 128, 386, 429  
 ExtensionList 165

## F

---

Favorites 161, 399  
 Fehlermeldungsdialog 186  
 Fenster 87, 100  
 Fenstergruppen 113  
 File Systems API 150  
 File Type 156, 390  
   *Aktion* 409  
 FileChangeAdapter 154  
 FileChangeListener 154  
 FileEntry 157  
 FileLock 153  
 FileObject 151  
   *erzeugen* 152  
   *lesen und schreiben* 153  
   *löschen* 152  
   *überwachen* 154  
   *umbenennen* 152  
   *Unterschied zu DataObject* 158  
   *verschieben* 153  
 FileSystem 150  
 FileTaskScanner 355  
 FileUtil 153  
 FilterNode 169  
 Friends 56

## G

---

Graphen 226  
 Group Configuration 114  
 Group Reference Configuration 115

## H

---

HelpCtx.Provider 238  
 Helpset erstellen 233  
 Hibernate 321  
   *Abhängigkeiten* 324, 337  
   *Beispiel* 324  
   *einbinden* 322  
   *konfigurieren* 325  
   *Logging* 326  
   *Mapping* 326  
   *Objektabbildung* 326  
   *Objekte speichern* 330

hibernate.cfg.xml 325, 329  
 hibernate.properties 329  
 Hilfe  
   *kontextsensitiv* 238  
 Hilfesystem 23, 233  
   *Inhaltsverzeichnis* 235  
   *Links* 236  
 Hinweisdialog 185

## I

---

ID3  
   *API* 394  
   *Bibliothek* 394  
   *Editor* 396  
   *ID3v1* 393  
   *ID3v2* 393  
   *Support* 393  
 Implementation Dependency 57  
 Implementation Version 40, 55  
 Important Files 52  
 Index.ArrayChildren 415  
 IndexedNode 169  
 InplaceEditor 252  
 Install-After 167  
 Install-Before 167  
 Installer → Module Installer  
 instance 47  
 instanceClass 47  
 InstanceContent 142  
 InstanceCookie 47  
 instanceCreate 47  
 instanceOf 48  
 Instanzen registrieren 47  
 Intermodulkommunikation 138, 141  
 Internationalisierung → Lokalisierung  
 Internationalization Wizard 275

## J

---

JarFileSystem 150  
 Java DB 301, 424  
   *Client-Treiber* 306  
   *Datenbank anlegen* 304  
   *Datenbank erstellen* 302  
   *Datenbank herunterfahren* 305  
   *einbinden* 301  
   *Fremdschlüssel* 309  
   *herunterfahren* 304, 310

## Index

*hochfahren* 302, 303  
*NetBeans IDE-Unterstützung* 305  
*Primärschlüssel* 308  
*Serververbindung* 307  
*Systemverzeichnis* 302  
*Tabellenstruktur ermitteln* 307  
*Treiber* 302  
*Verbindungsparameter* 304  
Java Media Framework 389, 405  
Java Persistence API 332  
    *Annotations* 336  
    *Entitäten* 335  
    *Implementation* 333  
    *Konfiguration* 334  
    *Objekte speichern* 339  
    *Objektrelationale Abbildung* 332  
    *Query Language* 332  
Java Service Loader 146  
Java Sound SPI 389  
Java Version 40  
Java Web Start 288  
JavaHelp 233  
JavaHelp → Hilfesystem  
JAXB 344  
JAX-WS 344  
JMF → Java Media Framework  
JPA 332  
JPQL 332, 339

## K

---

Kapselung 26, 32, 39, 145  
Keymap 362  
Keystore 292  
Kommandozeilenparameter → Launcher anpassen  
Kompatibilität 54  
Kontextmenü 106

## L

---

Laufzeitkonfiguration 44  
Launcher 31  
Launcher anpassen 286  
Layer 36, 39, 44, 91  
    *Reihenfolge* 45  
    *Verknüpfung* 49  
layer.xml 35  
Layer-Tree 46, 52, 91

Lazy Loading 127  
Library Wrapper 389, 394, 424  
Library Wrapper Module 64  
LifecycleManager 187, 368, 377  
Locale 273  
Locale Extension Archive 33, 280, 285  
Locale-Einstellung 286  
LocalFileSystem 150  
Logging 381  
    *API* 381  
    *ein-/ausschalten* 383  
    *Fehlermeldungen* 384  
    *Konfiguration* 382  
    *Level* 383  
    *Logger* 381  
    *Manager* 382  
Login-Dialog 187  
Lokalisierung 273  
    *beliebige Dateien* 278  
    *bereitstellen* 279, 295  
    *Grafiken* 278  
    *Hilfeseiten* 276  
    *Manifest* 275  
    *Textkonstanten* 273  
Lookup 48, 80, 127, 402  
    *Funktionsweise* 127  
    *globales* 128  
    *globales Proxy* 140  
    *Typsicherheit* 127  
    *überwachen* 141  
Lookup.Provider 140  
Lookup.Result 144  
LookupListener 80, 141, 144  
Lookups 380  
Lose Kopplung 27, 127, 128, 139

## M

---

Major Release Version 54, 57  
Manifest 35, 36, 38  
Manifest-Palette 349  
Media Library 399  
Menü 88  
    *Menüeintrag erstellen* 88  
    *Menüeinträge ausblenden* 91  
    *Menüeinträge sortieren* 90  
    *Separator* 90  
    *Untermenü* 90  
MenuBar 92

- Menubar 88
    - erstellen* 92
  - MenuView 178
  - META-INF services Browser 135
  - META-INF/services 131, 135, 146
  - methodvalue 48
  - MIME-Type 156
  - Mnemonics 85
  - Mode 97, 109
    - DTD* 430
    - Editor* 97
    - editor* 110
    - erstellen* 110
    - Größe und Position* 112
    - Konfigurationsdatei* 110
    - sliding* 110
    - View* 97
    - view* 110
  - Modul 36
    - Abhängigkeit* 395
    - Abhängigkeiten* 284
    - betriebssystemabhängig* 43
    - Container* 51
    - Informationen* 365
    - Konfigurationsdatei* 36, 37
    - laden* 37
    - Lebenszyklus* 39, 59
    - Ressourcen laden* 366
    - Struktur* 36
    - Wizard* 51
  - Modulares System 54
  - Modularität 26
  - Module 19, 25
    - erstellen* 50
    - integrieren* 37
    - konfigurieren* 37
    - verwenden* 56
  - Module Classloader 32
  - Module Installer 31, 39, 59, 389, 424
  - Module Registry 31, 62
  - Module Suite 51, 283
    - erstellen* 388
  - Module System 35
  - ModuleInfo 62, 365
  - ModuleInstall 59
  - Modullebenszyklus 39
  - MP3 File Type 390
  - MP3-Dateien
    - wiedergeben* 409, 412
  - MP3-Manager 385
    - Entwurf* 385
  - MP3-Player 401
  - Mp3Player 402
  - MP3-Plugin 389
    - registrieren* 389
  - MP3-Unterstützung 389
  - Multi Views API 204
  - MultiDataObject 157
  - MultiDataObject.Entry 157, 165
  - MultiFileLoader 164
  - MultiFileSystem 150
  - Multi-Parent Classloader 32
  - MultiViewElement 205
- N**
- 
- Namensräume 33
  - Navigator 243
  - Navigator API 243
  - NbBundle 273, 285
  - NBM-Paket 289, 290
    - erstellen* 291
    - signieren* 292
  - NbPreferences 261, 368
  - NetBeans IDE
    - erweitern* 349
  - NetBeans Platform 22
    - Architektur* 25
    - beenden* 377
    - Distribution* 28
    - Eigenschaften* 22
    - Oberflächengestaltung* 22
    - Schritt für Schritt* 385
    - Vorteile* 22
  - NetBeans Protokolle
    - nbdocs* 234, 238
    - nbres* 278
    - nbresloc* 278
  - NetBeans Runtime Container 26, 30
    - Bootstrap* 30
    - File System* 31
    - Module System* 30
    - Startup* 30
    - Utilities* 31
  - Node 76, 168, 169
    - Aktionen* 166, 171
    - aktive* 179, 181
    - Children* 173

## Index

- Container* 170
- Event-Handling* 172
- Icons* 171
- implementieren* 173
- Kontextmenü* 166, 171
- Standardaktionen* 181
- Node Container 415
- NodeAction 76
- NodeAdapter 172
- NodeListener 172
- Nodes
  - aktive* 412
- Nodes API 150, 168
- noIconInMenu 73
- NotifyDescriptor 183, 184
  - Confirmation* 185
  - Exception* 186
  - InputLine* 186
  - Message* 185

## O

---

- ObjectScene 224
- Objektrationale Abbildung 327
- Objektrationale Brücke 322
- OpenIDE-Module 38
- OpenIDE-Module-Build-Version 41
- OpenIDE-Module-Class 166
- OpenIDE-Module-Deprecated 41
- OpenIDE-Module-Deprecation-Message 41
- OpenIDE-Module-Display-Category 39
- OpenIDE-Module-Friends 39
- OpenIDE-Module-Implementation-Version 40
- OpenIDE-Module-Install 39, 61
- OpenIDE-Module-Java-Dependencies 40
- OpenIDE-Module-Layer 39
- OpenIDE-Module-Localizing-Bundle 40
- OpenIDE-Module-Long-Description 39
- OpenIDE-Module-Module-Dependencies 40
- OpenIDE-Module-Module-Dependency-Message 41
- OpenIDE-Module-Name 38
- OpenIDE-Module-Needs 42
- OpenIDE-Module-Package-Dependencies 40

- OpenIDE-Module-Package-Dependency-Message 41
- OpenIDE-Module-Provides 42, 132
- OpenIDE-Module-Public-Packages 39
- OpenIDE-Module-Recommends 42
- OpenIDE-Module-Requires 42, 132
- OpenIDE-Module-Requires-Message 42
- OpenIDE-Module-Short-Description 38
- OpenIDE-Module-Specification-Version 40
- Optionen 254
- Options Dialog API and SPI 254
- Optionspanel
  - erstellen* 254
  - hinzufügen* 260
  - Kategorie* 259
- Original Classloader 32
- originalFile 49
- Output Window 240

## P

---

- Palette 262, 349
  - bestehende erweitern* 354
  - Controller registrieren* 353
  - Drag & Drop* 269
  - Einträge hinzufügen* 263
  - erstellen* 264, 265
  - Item* 350
  - Item DTD* 436
  - Item internationalisieren* 350
  - Item registrieren* 352
- Palette API 352
- PaletteController 352
- PaletteFactory 352
- Patch Archive 33
- Persistence 337
- Persistence Units 334, 338
- persistence.xml 334, 338
- Persistenz 301, 424
- Persistenzmodell 332
- Perspectives 362
- Plattform-Module
  - anpassen* 285
- Plattformunabhängigkeit 21
- Playlist 414
  - speichern* 424
- Plugin Manager 28, 296
- Plugins → Module

Preferences API 254, 255, 260  
 Presenter 96  
     *Toolbar* 96  
 Primary File 157  
 Progress API 119  
 Progressbar 119, 120  
     *Darstellungsvarianten* 119  
     *einzelne Aufgaben* 120  
     *integrieren* 125  
     *komplexe Aufgaben* 122  
     *überwachen* 124  
 ProgressContributor 122  
 ProgressHandle 120  
 ProgressHandleFactory 120  
 ProgressMonitor 124  
 Project Metadata 67  
 Properties 248, 396  
     *benutzerdefinierter Editor* 252  
 Properties Sheet 396  
 PropertyChangeListener 193, 201  
 PropertyEditorSupport 253  
 PropertySupport 398  
 Proxy-Lookup 140  
 ProxyLookup 139, 247  
 Public Packages 55, 393

## Q

---

Query 339

## R

---

RDBMS 301  
 Redo 374  
 Regular 37  
 Representation Class 165  
 Resource Bundle 273  
 Resource Management 149  
     *Aufbau* 149  
 ResourceBundle 273  
 Ressourcen laden 366  
 Rich-Client 19  
     *Eigenschaften* 19  
     *Module* 19  
 Rich-Client Plattform 20  
     *Architektur* 20  
     *Lose Kopplung* 20  
     *Vorteile* 21

Runtime Container → NetBeans Runtime  
     Container  
 runtime-relative-path 66

## S

---

Scene 221  
     *exportieren* 223  
     *Satellite View* 223  
 Secondary Files 157  
 Service  
     *Standard-Implementation* 403  
 Service Interface 128, 129, 387, 401  
 Service Loader → Java Service Loader  
 Service Provider 128, 129, 135, 401, 405  
     *entfernen* 136  
     *registrieren* 135, 408  
     *sortieren* 136  
 Service Provider Configuration 131, 135  
 Services 128, 401  
     *anfordern* 132  
     *auffinden* 129, 131  
     *bereitstellen* 129  
     *globale* 132  
     *registrieren* 135  
 Services Folder 137  
 Services/MIMEResolver 165  
 Session 330  
 SessionFactory 329  
 Set 249  
 settings 49  
 shadow 49  
 Sheet 249  
 Shortcuts 84  
 Specification Version 40, 55  
 SPI 128  
 Splash Screen 284  
 SQL ausführen 306  
 Standalone Application 53, 284, 388  
 Standard.xml 93  
 Standard-Folder 45, 53  
 Startbedingungen 59  
 Statusbar 117  
     *erweitern* 118  
     *verwenden* 117  
 StatusDisplayer 117  
 StatusLineElementProvider 118  
 stringValue 48  
 Support Klasse 159

## Index

SwingWorker 344, 372  
System Classloader 32, 33  
System Filesystem 44  
    *Eigene Inhalte* 50  
    *Reihenfolge* 45  
System Tray 379  
    *Icon hinzufügen* 379

## T

---

Task 356  
Task List 354  
    *API* 355  
    *Extension Points* 358  
    *Group* 358  
    *Scanner* 355  
    *Scanner registrieren* 358  
    *Scope* 355  
    *Task erstellen* 356  
Tipps und Tricks 371  
Toolbar 92  
    *anpassen* 95  
    *DTD* 435  
    *dynamisch ändern* 93  
    *eigene Steuerelemente* 96  
    *eigene Toolbars* 95  
    *erstellen* 92  
    *Konfigurationen* 93  
    *kontextabhängig* 106  
ToolbarPool 94  
Top Component 100, 416  
    *erstellen* 100  
    *in Mode docken* 104  
    *Kontextmenü* 106  
    *mehrere Instanzen* 416  
    *Persistenz* 107  
    *Registry* 108  
    *Zustände* 104  
Top Component Group 113  
    *erstellen* 114  
TopComponent.Registry 108, 412  
TopComponentGroup 113  
    *DTD* 434  
Transferable 269, 271, 422  
TransferHandler 271  
TrayIcon 379  
TreeTableView 414  
Tutorials 182

## U

---

Undo 374  
Undo/Redo  
    *für Textkomponenten* 376  
    *Manager* 375  
UndoableEdit 374  
UndoManager 374  
UndoRedo.Manager 374  
UniFileLoader 164  
Update 21, 289  
Update-Center 289, 294  
    *ausliefern* 298  
    *Descriptor erstellen* 295  
    *hinzufügen* 298  
Update-Center-Descriptor 294  
Updates  
    *automatisch installieren* 299  
    *installieren* 296  
urlvalue 48  
Utilities.actionsGlobalContext() 140

## V

---

Versionen 40, 54  
Versionierung 54  
Visual Library  
    *Aufbau* 209  
    *Graphen* 226  
    *ObjectScene* 224  
    *Scene* 221  
    *VMD Klassen* 230  
    *Widgets* 210, 211  
Visual Library API 209  
Visual-Panel 191, 194

## W

---

Warm-Up Tasks 378  
Web Services 341  
    *Client* 341  
    *verwenden* 343  
Widgets 210, 211  
    *Abhängigkeiten* 213  
    *Aktionen* 215  
    *Ereignisse* 215  
    *Layout* 213  
    *Rahmen* 213  
Wiederverwendbarkeit 22

- Wiederverwendung 145
  - Window Manager 116
  - Window System 87, 97
    - Konfiguration* 98
  - WindowManager 108, 116
  - Wizard 189
    - Architektur* 190
    - Daten auswerten* 200
    - Datenmodell* 196, 199
    - Ereignisse* 196
    - erzeugen und aufrufen* 200
    - Event-Handling* 201
    - Fortschrittsanzeige* 204
    - Hilfe* 195
    - Iterator* 199
    - Iteratoren* 203
    - Panels erstellen* 192
    - Panels zusammenfügen* 199
    - Reihenfolge der Panels* 203
    - Validierung von Daten* 197
    - Visual-Panel* 193
    - vorzeitig beenden* 202
    - zusätzliche Validierung* 203
  - WizardDescriptor 190, 199
    - Panel* 191, 194
  - Wizard-Panel 191, 194
  - Wizards 189
  - Wrapper Style 342
  - WSDL 341
  - wsggrp 114
  - wsmode 110, 112
  - wstcgrp 115
  - wstcref 103, 113
- X**
- 
- XMLFileSystem 150