1 Handbuch

1.1 Quickstart

1. Entpacken Sie die Zip-Datei, in der sich das ModuleSkeleton befindet, an einen Ort auf Ihrer Festplatte.

2. Öffnen Sie das extrahierte Verzeichnis als Netbeans7-Projekt.

3. Passen Sie in der `info.xml-Datei` die Modul-ID und ggf. weitere Details an.

4. Fügen Sie dem Projekt nach Bedarf Java-Beans und JSF-Seiten hinzu und registrieren sie die Beans in der `beans.xml`.

5. Passen Sie die ID auch in der `build.xml` an und führen Sie anschließend per `ant` das Ziel '`createmodule`' aus. Das fertige Archiv befindet sich nun im `dist/`-Ordner.

6. Nun lässt sich das Modul per ModuleManager-Oberfläche installieren.

1.2 Verzeichnisstruktur und besondere Dateien

```
ModuleSkeleton/
├── ant/
│   └── ModuleCreationFramework.jar ... Bibliothek für den Ant Task
│                                       'createmodule'.
├── build.xml ... Ant Buildfile mit Tasks zur Abhängigkeitsauflösung
│                 und Modularchiv-Erstellung.
├── dist/ ... Enthält das erstellte Modularchiv.
├── ivy.xml ... Ermöglicht die Definition von Abhängigkeiten,
│               welche durch Ivy aufgelöst werden sollen.
├── lib/ ... Zielort für eigene bzw. von Ivy aufgelöste
│           Bibliotheken.
│   ├── compile/ ... Enthält Bibliotheken, die nur zur Kompilierung benötigt
│   │                werden. Diese Dateien finden nicht den Weg ins
│   │                Modularchiv sondern müssen zur Laufzeit ggf. separat
│   │                bereitstehen.
│   └── runtime/ ... Enthält jene Bibliotheken, die mit ins Modularchiv
│                    verpackt werden sollen. Diese Biliotheken werden dann
│                    bei der Modul-Aktivierung geladen.
├── src/ ... Hauptverzeichnis für den Java-Quellcode.
└── web/ ... Das Wurzelverzeichnis der Weboberfläche dieses Moduls.
    ├── index.xhtml ... Hauptseite des Moduls. Auf diese Seite wird im
    │                   Modulmenü der Host-Application verwiesen.
    ├── resources/ ... Verzeichnis für statische Inhalte. Hierher
    │                  gehören Bilder etc.
    └── WEB-INF/
        ├── beans.xml ... Eigene Bean-Definitionen des Moduls gemäß dem
        │                 Spring-Schema.
        └── info.xml ... Meta-Informationen des Moduls. Vom Modulnamen,
                         über Rollenbeschränkungen bis hin zur
                         Dokumentation wird hier alles definiert.
```

1.3 info.xml

```
[...]
<!-- BiBiServ2 Admin Module Information File -->
<module xmlns="bibiserv:de.unibi.cebitec.bibiserv.web.
   administration.beans.modulemanager.moduleinfo">
    <!--
    Module ID. Must be unique. Allowed characters: [a-z0-9_]+
    Important: If a module with this ID is already installed on
       the server,
    it will be replaced (as an easy way of updating).
    -->
  <id>skeleton</id>
    <!--
    Full name of the module. For display in menus etc.
    -->
  <name>Skeleton Module</name>
    <!--
    Module version. The version number is shown at installation
       time.
    It has no effect other than to be displayed.
    -->
  <version>0.01</version>
    <!--
    Role restrictions comma-separated. Not yet checked on server-
       side,
    but intended for future use.
    e.g. <role-restrictions>ROLE_ADMIN,ROLE_DEVELOPER</role-
       restrictions>
    -->
  <role-restrictions></role-restrictions>
    <!--
    List of the author(s) of this module.
    Suggested schema: Firstname Lastname &lt;email&gt;
    -->
  <authors>
    <author>Firstname Lastname &lt;flastnam@cebitec.uni-bielefeld
       .de&gt;</author>
    <!-- Add additional authors here as necessary; according to
       the example above. -->
  </authors>
    <!--
    Short description of this module.
    -->
  <description>
```

```
37      Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
           diam nonumy
38      eirmod tempor invidunt ut labore et dolore magna aliquyam
           erat, sed diam voluptua.
39      At vero eos et accusam et justo duo dolores et ea rebum.
40    </description>
41      <!--
42      Admin-Manual displayed in the ModuleManager.
43      -->
44    <manual>
45      <chapter>
46        <title>
47                  Overview
48        </title>
49        <content>
50      The ModuleCreationFramework allows you to create BiBiServ2
           Admin Modules.
51      Admin Modules can be integrated as part of the server at
           runtime.
52      The ModuleSkeleton contains all you need to create your own
           one.
53        </content>
54      </chapter>
55      <chapter>
56        <title>
57                  Quickstart
58        </title>
59        <content>
60          To create a new Admin Module ...(tbc)
61        </content>
62      </chapter>
63      <!-- Add additional chapters here as necessary; according to
           the examples above. -->
64    </manual>
65  </module>
```

1.4 beans.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans [...]>
3  <!-- All beans that are part of this module go here. -->
4    <bean id="sampleBean" class="de.unibi.cebitec.bibiserv.web.
        modules.skeleton.SampleBean" scope="request"/>
5  </beans>
```

1.5 Auflösung von Abhängigkeiten per Ivy

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ivy-module version="2.0">
3
4   <info organisation="de.unibi.cebitec.bibiserv"
5                module="adminmodule" revision="1.0"/>
6
7   <configurations>
8    <conf name="compile" extends="runtime"
9                description="compile environment"/>
10   <conf name="runtime" description="runtime environment"/>
11  </configurations>
12
13  <publications/>
14
15  <dependencies>
16   <!-- runtime dependencies -->
17
18   <!-- compile time dependencies -->
19   <dependency org="org.primefaces" name="primefaces"
20        rev="latest.integration" conf="compile->default"/>
21
22  </dependencies>
23
24 </ivy-module>
```

Zur komfortablen und einheitlichen Auflösung der Abhängigkeiten des Moduls findet "Apache Ivy" Einsatz.
Dazu wird über die `ivy.xml`-Konfigurationsdatei die Liste der benötigten Bibliotheken definiert.
Hierbei sind zwei Arten von Abhängigkeiten, nämlich `compile`-Abhängigkeiten und `runtime`-Abhängigkeiten, zu unterscheiden:

compile Abhängigkeiten zur Kompilierungszeit. Das bedeutet, diese Bibliotheken werden *nur* zur Übersetzung des Quellcodes benötigt, jedoch zur späteren Laufzeit ggf. global extern bereitgestellt.

runtime Abhängigkeiten zur Laufzeit. Diese Dateien sollen *nicht nur* zur Kompilierungszeit bereitstehen, sondern darüber hinaus auch *mit* dem Modul mitgeliefert werden. Somit ist sichergestellt, dass diese Bibliotheken zur Laufzeit verfügbar sind. Hierbei sei aber besondere Umsicht geboten, da die Gefahr von Kollisionen mit am Zielort bereits vorhandenen Bibliotheken besteht.

1.6 Erstellung des Modul-Archivs

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project name="ModuleSkeleton" default="all" basedir="."
3         xmlns:ivy="antlib:org.apache.ivy.ant">
4   <property name="id" value="moduleskeleton" />
5
6   [...]
7
8   <!-- include mcf -->
9   <taskdef
10 resource="de/unibi/cebitec/bibiserv/mcf/ant/tasks.properties"
11 classpath="${basedir}/ant/ModuleCreationFramework.jar"/>
12
13  [...]
14
15  <target name="createmodule" depends="clean,resolve">
16
17   <mkdir dir="${classes}"/>
18   <javac srcdir="${src}" destdir="${classes}" debug="on"
19                                 includeantruntime="false"/>
20
21   <createmodule archive="${archive}">
22    <fileset dir="${basedir}" includes="web/**/**" />
23    <fileset dir="${build}" includes="classes/**/**" />
24    <fileset dir="${basedir}">
25     <include name="lib/runtime/**" />
26    </fileset>
27   </createmodule>
28
29  </target>
30
31 [...]
```

Die Eigenschaft ‘id’ (`build.xml:4`) bestimmt den Dateinamen des späteren Archivs. Idealerweise sollte dieser mit der ID in der `info.xml`-Datei übereinstimmen.
Das Ziel ‘`createmodule`’ kompiliert den Code (`build.xml:18`) und erstellt das Modul-Archiv (`build.xml:21`). Hierbei werden standardmäßig 3 Verzeichnisse inkludiert:

- Web-Verzeichnis (incl. WEB-INF) (`build.xml:22`)
- Kompilierte Klassen (`build.xml:23`)
- Laufzeitbibliotheken (`build.xml:25`)

Bei Veränderungen an diesen Anweisungen ist das jeweils angegebene Basis-Verzeichnis (`dir=""`) unbedingt zu beachten, um zu gewährleisten, dass der spätere Pfad in der Zip-Datei unverändert bleibt.
Ein erfolgreicher Durchlauf des Build-Prozesses sieht bspw. so aus:

```
[...]
[ivy:retrieve]  1 artifacts copied, 0 already retrieved (1683kB/113ms)

createmodule:
    [mkdir] Created dir: ModuleSkeleton/build/classes
    [javac] Compiling 1 source file to ModuleSkeleton/build/classes
    [createmodule] (xml-validation): web/WEB-INF/info.xml -> OK
    [null] Building jar: ModuleSkeleton/dist/moduleskeleton.car

BUILD SUCCESSFUL
```

1.7 Inhaltsstruktur der Archiv-Datei

```
moduleskeleton.car/
├── classes/
├── lib/
│   └── runtime/
└── web/
    ├── resources/
    └── WEB-INF/
```

1.8 Elemente der ModuleManager-Oberfläche

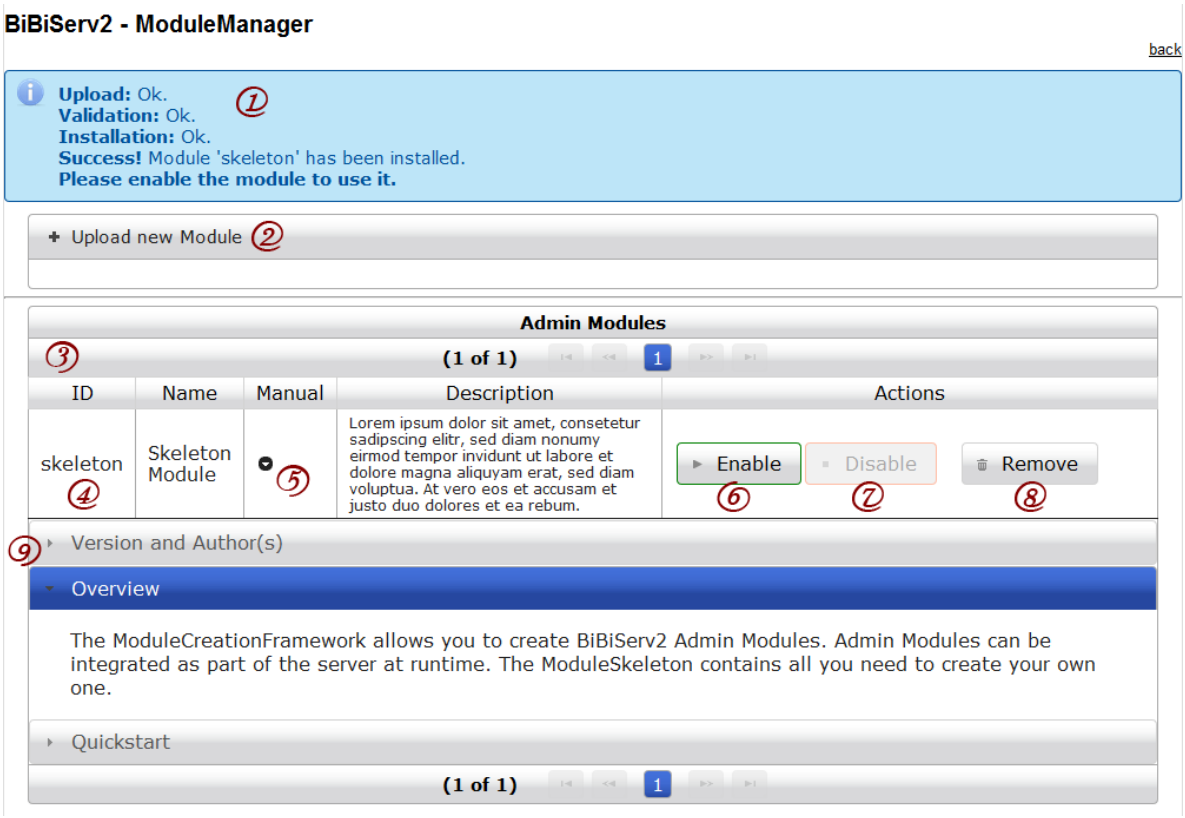

Abbildung 1: Web-Oberfläche des ModuleManagers

1. **Statusbereich** - Bietet Informationen über ausgeführte Aktionen.
2. **Uploadfunktion** - Neue Modul-Archive werden hiermit hochgeladen und installiert bzw. aktualisiert.
3. **Liste der installierten Module** - Liste aller Module die sich in der Datenbank befinden.
4. **Modul-ID** - Eindeutiger Bezeichner des Moduls. Dieser bestimmt u. a. auch die URL des Modulbereichs.
5. **Erweiterungsknopf für die Modul-Dokumentation** - Bei Klick auf diesen Knopf öffnet sich (9).
6. **Aktivierungsknopf** - Deployt das Modul und macht es verfügbar.
7. **Deaktivierungsknopf** - Undeployt das Modul und inaktiviert es.

8. **Löschknopf** - Entfernt das Modul vollständig und damit auch aus der Datenbank.
9. **Modul-Dokumentation** - Präsentiert die in der `info.xml` des Moduls definierte Dokumentation.

1.9 interne technische Vorgänge der Host-Application bei Install/Remove Activate/Deactivate

TODO

1.10 Welchen Beitrag leistet das ModuleCreationFramework?

Das ModuleCreationFramework (MCF) übernimmt die Validierung und Verpackung eines Moduls.
Die Struktur der das Modul beschreibenden `info.xml`-Datei folgt einem festgelegten Schema. Erst dies ermöglicht die Modularität des Systems und lässt die Datei somit zum universellen Verbindungspunkt werden. Um schon vor Installation eines Moduls Fehler in diesem kritischen Bereich zu minimieren, validiert das MCF die Datei vor der Generierung des Modul-Archivs gegen ein XML-Schema (`bibiserv2-admin-module-info.xsd`). Erst im Anschluss daran verpackt das MCF alle benötigten Dateien in einem Zip-Archiv mit der Endung '`*.car`'.

1.11 Troubleshooting

Problem	Lösung
to	do
to	do

Tabelle 1: Troubleshooting

1.12 Quellen

TODO: Ivy, Ant, Custom Ant Tasks, Spring (beans.xml)
TODO: Bibliography