# Faktor-IPS Produktdaten in der Datenbank

# Table of Contents

# Einleitung

Faktor-IPS speichert mit dem StandardBuilder alle Produktdaten in XML-Dateien. Diese können zur Laufzeit von einem `ClassLoaderProductDataProvider` aus dem Classpath geladen werden. Alternativ gibt es mit der `faktorips-runtime-jpa`-Erweiterung die Möglichkeit, die Produktdaten in einer Datenbank zu speichern. Um diese auszulesen gibt es den DbProductDataProvider. Zum Deployment in die Datenbank gibt es mehrere Clients, die im Kapitel Deployment der Produktdaten beschrieben werden.

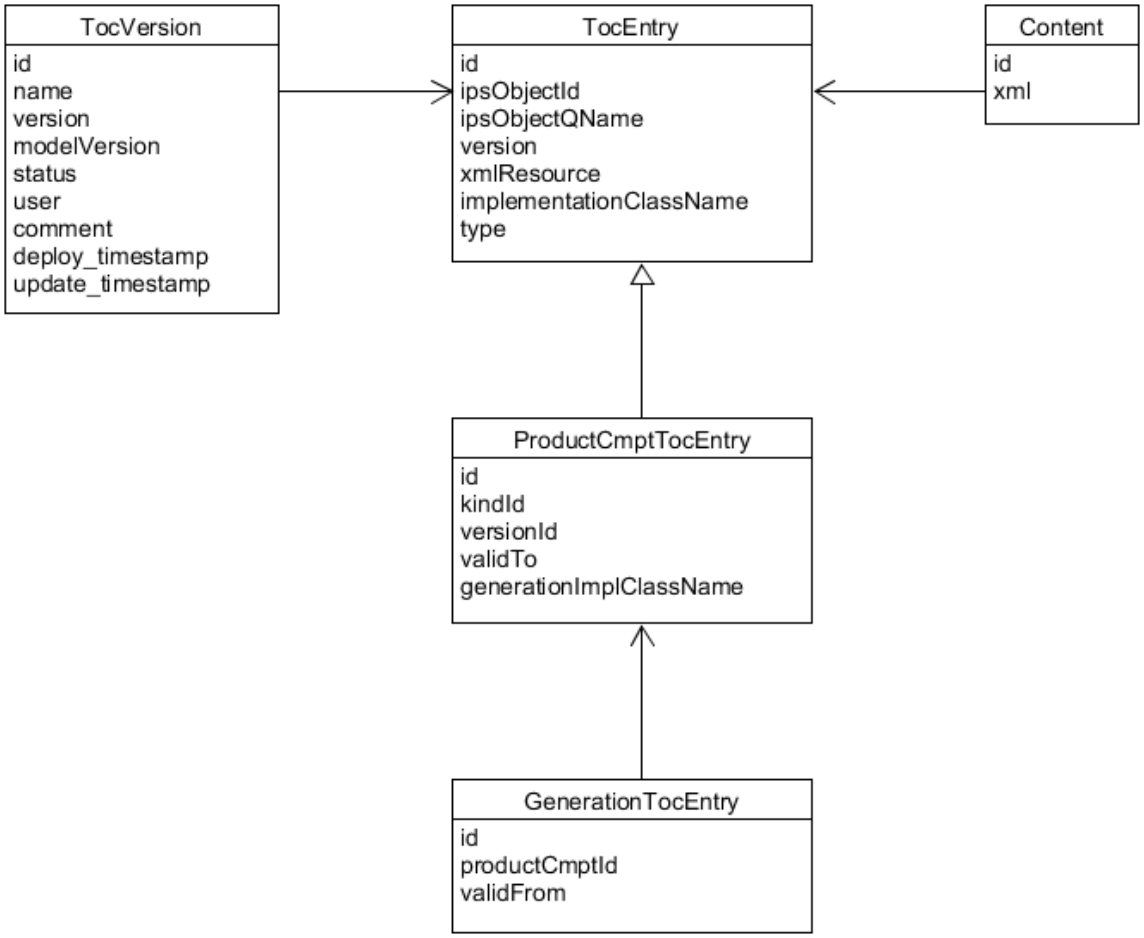
Die Produktdaten in der Datenbank bringen insbesondere folgende Vorteile:

- Deployment unabhängig von EJB (ClassLoader Probleme etc.)
- Paralleles Deployment mehrerer Versionen (nur eine aktive Version je Modellversion)
- Trennen von Test-/Produktivumgebungen durch Verwendung unterschiedlicher Datenbanken
- Ermöglichen von zusätzlichen Überprüfungen (z.B. kein Wegfall von RuntimeIDs, Produktdaten passen zu Modell)

Die Java Persistence API (JPA) ist eine Schnittstelle, die den Zugriff auf unterschiedliche Datenbanken vereinfacht. Da sie für das Schreiben und Lesen der Produktdaten in der Datenbank verwendet wird, ist sie Teil des Namens dieser Erweiterung.

# Konzept

Faktor-IPS legt für jedes Produktprojekt (technisch genauer je Source Root) ein Inhaltsverzeichnis (ToC) an. Dieses kann in unterschiedlichen Versionen vorliegen. Die Einträge eines ToC werden in der Datenbanktabelle `TOC_ENTRY` gespeichert, jeder Eintrag in einer Zeile. Hier werden die aktuellen Runtime-IDs, die zugehörigen Implementierungsklassen und eine Referenz auf die XML Content Tabelle verwaltet. Die einzelnen Versionen des ToC werden in der Tabelle `TOC_VERSION` gespeichert.




# Versionierung

Eine ToC Version wird durch drei Aspekte identifiziert:

- **Name**: der Name des ToC. Dieser kann unabhängig vom Namen des Faktor-IPS-Projekts frei nach fachlichen oder technischen Anforderungen gewählt werden.
- **Modell-Version**: die Version des Modell-Projekts, auf dem die Produktdaten beruhen. Dadurch können in einer Datenbank Produktdaten, die auf unterschiedlichen Modellversionen beruhen, parallel liegen. Wie diese Version im Deployment gesetzt

wird, ist im Abschnitt Deployment der Produktdaten - Client beschrieben.

- **Version**: die Version des Produktprojekts.

Die Version des Produktprojekts ist unabhängig von der Modellversion, da die Produktdaten einem anderen Entwicklungszyklus unterliegen. Je Modellversion können mehrere Produktversionen entwickelt werden. Für jede neue Modellversion muss es aber auch eine neue Produktversion geben.

# Status

Jede ToC-Version hat einen Status mit einen der folgenden Werten:

- `PENDING`: der initiale Status aller ToC-Versionen. Die ToC-Version bleibt solang in diesem Status, bis alle Inhalte erfolgreich deployed wurden.
- `DEPLOYED`: eine vollständig deployte ToC-Version. Die Version kann aktiviert werden um die momentan aktive Version zu ersetzen.
- `ACTIVE`: die aktive ToC-Version.
- `HISTORIC`: Wird eine neue Version aktiviert, erhält die bisher aktive Version diesen Status.

Zu jeder Kombination aus Namen und Modell-Version darf immer maximal eine Version als `ACTIVE` markiert werden. Diese wird vom `DbProductDataProvider` zurückgegeben.

# Weitere Eigenschaften

Zu jeder ToC-Version kann in der Datenbank auch ein Kommentar und Benutzername hinterlegt werden, um zusätzliche Informationen zum Deployment festzuhalten. Automatisch wird auch der Zeitpunkt des Deployments gespeichert.

# Auslesen der Produktdaten

Zum Auslesen der Produktdaten zur Laufzeit wird ein `DbProductDataProvider` zur Verfügung gestellt. Der `DbProductDataProvider` ist eine Implementierung von `org.faktorips.runtime.productdataprovider.IProductDataProvider` und bietet Zugriff auf die Produktdaten. Zusätzlich ermöglicht sie Produktdaten auf Aktualität zu prüfen und ggf. neu zu laden, wenn inzwischen neuere Daten zur Verfügung stehen.

## ProductDatabase

Der Zugriff auf die Datenbank ist über die Stateless EJB `ProductDatabaseBean` gekapselt. Sie erhält einen `EntityManager` für die Persistence Unit `faktorips-productdataservice-jpa` per Konstruktor(-Injection). Das Business Interface `ProductDatabase` enthält die Methode `getActiveTocVersion`, die aus der Datenbank die mit dem Status `ACTIVE` markierte Version abruft.

Der Server muss ein Data Source mit dem JNDI Namen `ips-product-data` zur Verfügung stellen.

## DbProductDataProviderFactory

Ein `DbProductDataProvider` wird über eine `DbProductDataProviderFactory` erstellt. Dieser benötigt neben der `ProductDatabase` zusätzlich den Namen und die Modell-Version (siehe Versionierung) des gewünschten Produktdaten-Projekts.

## Verwendung mit DetachedContentRuntimeRepository

Sollen Produktdaten aus der Datenbank in einer Anwendung verwendet werden, wird ein `DetachedContentRuntimeRepository` benötigt. Dieses wiederum wird von einem `DetachedContentRuntimeRepositoryManager` erzeugt. Dieser kümmert sich außerdem darum, die Aktualität der Produktdaten bei jedem Abruf zu überprüfen und ggf. ein neues `DetachedContentRuntimeRepository` mit neuem `DbProductDataProvider` zur nun aktiven Produktdaten-Version zu erzeugen. Dem Builder des `DetachedContentRuntimeRepositoryManager` wird dazu die `DbProductDataProviderFactory` übergeben.

```
emForRepo = emFactory.createEntityManager(); ①

DbProductDataProviderFactory productDataProviderFactory = new
DbProductDataProviderFactory(tocName, "1.2.3",
        new ProductDatabaseBean(emForRepo)); ②
```

```
IRuntimeRepositoryManager runtimeRepositoryManager = new
DetachedContentRuntimeRepositoryManager.Builder(
        productDataProviderFactory).build();
IRuntimeRepository runtimeRepository =
runtimeRepositoryManager.getCurrentRuntimeRepository();

testDeployedData(runtimeRepository);
```

① Der EntityManager kann in einer JEE (6+) Umgebung auch per Injection gesetzt werden

② Die Modellversion kann in der Anwendung konfiguriert werden oder zur Laufzeit aus dem Modell-TOC gelesen werden

# Deployment der Produktdaten

Für das Einspielen der Produktdaten in die Datenbank ist der `ProductDataDeployment` -Service vorgesehen, der über eine REST-API angesprochen werden kann. Zum einfachen Aufruf dieser API dient der Client, der direkt programmatisch bedient oder über einen Kommandozeilen-Client oder ein Maven-Plugin aufgerufen werden kann.

# ProductDataDeployment-Service

Die Rest-API für das Einspielen der Produktdaten wird über das Deployment des `productdata-jpa-deployment-service.war` auf einen Java-EE-Server (EE6+ oder NetWeaver 7.50+) mit konfigurierter Persistence Unit `faktorips-productdataservice-jpa` zur Verfügung gestellt.

## Data Source

Im Server muss dazu die Datasource mit dem JNDI Namen `ips-product-data` (bzw. je nach Server auch `java:/ips-product-data`) konfiguriert werden.

Um die Datenbanktabellen zu erstellen kann das folgende Skript als Vorlage dienen:

*Oracle*

```
CREATE TABLE IPS_CONTENT (ID VARCHAR2(22) NOT NULL, XML CLOB NOT NULL, PRIMARY
KEY (ID));
CREATE TABLE IPS_TOC_ENTRY (ID VARCHAR2(22) NOT NULL, type VARCHAR2(31) NULL,
IMPLEMENTATIONCLASSNAME VARCHAR2(255) NOT NULL, IPSOBJECTID VARCHAR2(255) NOT
NULL, IPSOBJECTQNAME VARCHAR2(255) NOT NULL, VERSION_ID VARCHAR2(22) NULL,
XMLRESOURCE_ID VARCHAR2(22) NULL, PRIMARY KEY (ID));
CREATE INDEX PSTCENTRYFKPSTOCENTRYVERSIONID ON IPS_TOC_ENTRY (VERSION_ID);
CREATE INDEX PSTCNTRYPSTCENTRYXMLRESOURCEID ON IPS_TOC_ENTRY (XMLRESOURCE_ID);
CREATE UNIQUE INDEX AK_TOCENTRY_OBJECT_VERSION ON IPS_TOC_ENTRY (IPSOBJECTID,
VERSION_ID);
CREATE TABLE IPS_GENERATION_TOC_ENTRY (ID VARCHAR2(22) NOT NULL, VALIDFROM
TIMESTAMP NOT NULL, PRODUCT_CMPT_ID VARCHAR2(22) NULL, XMLRESOURCE_ID
VARCHAR2(22) NULL, PRIMARY KEY (ID));
CREATE INDEX PSGNRTNPSGNRTNTCNTRYPRDCTCMPTD ON IPS_GENERATION_TOC_ENTRY
(PRODUCT_CMPT_ID);
CREATE INDEX PSGNRTNTCPSGNRTNTCNTRYXMLRSRCD ON IPS_GENERATION_TOC_ENTRY
(XMLRESOURCE_ID);
CREATE TABLE IPS_PRODUCT_CMPT_TOC_ENTRY (ID VARCHAR2(22) NOT NULL,
GENERATIONIMPLCLASSNAME VARCHAR2(255) NULL, KINDID VARCHAR2(255) NOT NULL,
VALIDTO TIMESTAMP NULL, VERSIONID VARCHAR2(255) NOT NULL, PRIMARY KEY (ID));
CREATE INDEX PSPRDCTCMPTTPSPRDCTCMPTTCNTRYD ON IPS_PRODUCT_CMPT_TOC_ENTRY
(ID);
```

```
CREATE TABLE IPS_TOC_VERSION (ID VARCHAR2(22) NOT NULL, DEPLOY_COMMENT
VARCHAR2(255) NULL, DEPLOY_TIMESTAMP TIMESTAMP NULL, UPDATE_TIMESTAMP
TIMESTAMP NULL, MODEL_VERSION VARCHAR2(255) NULL, NAME VARCHAR2(255) NULL,
STATUS VARCHAR2(255) NULL, DEPLOY_USER VARCHAR2(255) NULL, VERSION
VARCHAR2(255) NULL, PRIMARY KEY (ID));
CREATE UNIQUE INDEX AK_TOC_VERSION_NAME_VER_MVER ON IPS_TOC_VERSION (NAME,
VERSION, MODEL_VERSION);
CREATE INDEX IX_TOC_VERSION_NAME_VER_STATUS ON IPS_TOC_VERSION (NAME, VERSION,
STATUS);
ALTER TABLE IPS_TOC_ENTRY ADD CONSTRAINT FK_IPS_TOC_ENTRY_VERSION_ID FOREIGN
KEY (VERSION_ID) REFERENCES IPS_TOC_VERSION (ID);
ALTER TABLE IPS_TOC_ENTRY ADD CONSTRAINT IPS_TOC_ENTRY_XMLRESOURCE_ID FOREIGN
KEY (XMLRESOURCE_ID) REFERENCES IPS_CONTENT (ID);
ALTER TABLE IPS_GENERATION_TOC_ENTRY ADD CONSTRAINT
PSGNERATIONTOCENTRYPRDCTCMPTID FOREIGN KEY (PRODUCT_CMPT_ID) REFERENCES
IPS_TOC_ENTRY (ID);
ALTER TABLE IPS_GENERATION_TOC_ENTRY ADD CONSTRAINT
PSGNRATIONTOCENTRYXMLRSOURCEID FOREIGN KEY (XMLRESOURCE_ID) REFERENCES
IPS_CONTENT (ID);
ALTER TABLE IPS_PRODUCT_CMPT_TOC_ENTRY ADD CONSTRAINT
IPS_PRODUCT_CMPT_TOC_ENTRY_ID FOREIGN KEY (ID) REFERENCES IPS_TOC_ENTRY (ID);
```

## Dokumentation Rest-API

Die Dokumentation der Rest-API wird mit dem Service zusammen deployt und kann unter dem Deployment-URL erreicht werden.

*Deployment auf JBoss Server*

Wird JBoss als Server verwendet müssen zwei Properties gesetzt werden:

- `org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true`, damit die in den IDs von Tabellen vorkommenden Slashs korrekt verarbeitet werden
- `eclipselink.target-server=JBoss` damit EclipseLink die Transaktionen korrekt abschließt.

Die API ist mit Basic Authentication im Authentication Realm `ips-deploy-realm` gesichert und erwartet einen Benutzer mit der Rolle `ipsdeploy`. Dieser ist im Application Server, wie im Kapitel Deploymnet mit Wildfly einrichten beschrieben, einzurichten. Der Benutzer selbst muss dem Realm `ips-deploy-realm` nicht zugeordnet sein, entscheidend ist hier die Rolle `ipsdeploy`.

# Client

Der `ProductDataDeploymentClient` wird über einen Builder mit folgenden Informationen erstellt:

- `ReadonlyTableOfContents`
- Name
- Modell-Version
- Produkt-Version (siehe Versionierung)

Er bietet Methoden an, die die REST-API aufrufen um zum Beispiel eine neue ToC Version anzulegen und alle darin aufgelistete Produkte, Tabellen und Enums einzuspielen.

## Builder

Der Builder für den `ProductDataDeploymentClient` wird über die Methode `ProductDataDeploymentClient.Builder.forApi(WebTarget)` erstellt. Mittels diverser `with~` -Methoden kann die zukünftige Client-Instanz konfiguriert werden. Durch den Aufruf von `build()` wird die Client-Instanz erzeugt und kann dann verwendet werden. Im Folgenden werden die einzelnen `with~`-Methoden erläutert.

Dieses Muster wird "Builder-Pattern" genannt. Die `with~`-Methoden geben jeweils ein neues Builder-Objekt zurück, um den Aufruf im Stil einer Fluent API zu erlauben und teilweise konfigurierte Builder-Instanzen wiederverwenden zu können.

### withTocPath

Konfiguriert den Pfad zu einer Table-of-Contents-Datei auf dem Classpath, mit dem der Builder gestartet wurde. Dieser Parameter muss konfiguriert werden, wenn der Client zum Einspielen von Produktdaten verwendet wird. Aus dem angegebenen ToC werden dann die einzuspielenden Produkte, Tabellen und Enums gelesen. Falls der Client zum Ändern des Status oder Löschen einer Produktdaten-Version eingesetzt wird, muss dieser nicht angegeben werden.

Wenn `tocPath` konfiguriert ist, kann die Produktversion automatisch aus dem ToC ausgelesen werden. Auch das Auslesen der Modell-Version ist möglich. Dazu muss einerseits zur `<toc-name>.xml`-Datei eine `<toc-name>.model.properties`-Datei vorhanden sein, andererseits die darin benannten Modell-ToCs auf dem Classpath verfügbar sein (siehe Modell-Versionen).

### withTocName

Der Name für das Produktdatenprojekt. Muss immer konfiguriert werden. Der `tocName`

entspricht dem Namen der `xml`-Datei, die das `ToC` enthält.

#### withModelVersion

Die Modell-Version. Kann weggelassen werden, wenn der `tocPath` angegeben ist und der Versionsstring damit automatisch erstellt wird (siehe Modell-Versionen).

Für das Löschen sowie Statusupdate von mehreren ToC Versionen kann ein Leerstring oder * als Wildcard angegeben werden.

#### withVersion

Die Produktdaten-Version. Kann weggelassen werden, wenn der `tocPath` angegeben ist und damit die im Table of Contents gespeicherte Versionsnummer verwendet wird.

Für das Löschen sowie Statusupdate von mehreren ToC Versionen kann ein Leerstring oder * als Wildcard angegeben werden.

#### withAuthentication

Benutzername und Passwort für den Zugriff auf die REST-API. Im Application Server muss der Benutzer mit der Rolle `ipsdeploy` eingetragen sein.

#### Logging und Fehlerbehandlung

Über die Methoden `withSuccessHandler` und `withFailureHandler` können einem Client Handler für das Logging von Deployment-Fortschritt und -Fehlern mitgegeben werden. Werden keine Handler explizit gesetzt, wird auf `System.out` geloggt und Fehlermeldungen auf `System.err` ausgegeben.

Ist einer der benötigten Parameter nicht gesetzt, wirft der Builder eine `MissingArgumentException`.

## Modell-Versionen

Wird dem Client-Builder nicht manuell eine Modell-Version mitgegeben, versucht er diese automatisch zu erstellen. Dazu sucht er parallel zum Produkt-ToC in `<toc-path>.xml` eine Datei `<toc-path>.model.properties`. In dieser sind Pfade zu ebenfalls auf dem Classpath liegenden Modell-ToCs abgelegt, aus denen wiederum die Versionsnummern ausgelesen werden.

Beruht das Produktprojekt direkt auf einem einzelnen Modellprojekt, hat die Property das folgende Format:

```
modelToc=org/faktorips/sample/model/internal/faktorips-repository-toc.xml
```

Die Modellversion wird dann direkt aus dem Attribut `productDataVersion` des Tags `FaktorIps-TableOfContents` ausgelesen, wohin sie beim Bauen des Modellprojekts aus dem Tag `Version` der `.ipsproject`-Datei kopiert wurde.

Beruht das Produktprojekt direkt auf einem mehreren Modellprojekten können diese mit Namen und einer Reihenfolge versehen werden:

```
modelToc.1.base=org/faktorips/sample/model/internal/faktorips-repository-
toc.xml
modelToc.2.lob=org/faktorips/sample/lob/model/internal/faktorips-lob-
repository-toc.xml
```

## Verwendung

Der Client bietet mehrere Methoden, die nacheinander aufgerufen werden können, um Produktdaten einzuspielen, aktiv zu schalten und ggf. wieder zu löschen. Jede Methode gibt zurück, ob das anlegen geklappt hat und loggt zusätzlich über die Success- und FailureHandler weitere von der REST-API zurückgegebene Informationen.

### createTocVersion

Legt eine neue Table-of-Contents-Version in der Datenbank an. Dabei werden die bei der Erstellung des Clients angegebenen Versionsparameter genutzt. Optional können die Parameter `user` und `comment` angegeben werden: `user`, um in der Datenbank sehen zu können, wer eine bestimmte Version angelegt hat; `comment`, um die Version mit einem Kommentar zu versehen..

### deployProducts

Spielt alle Produkte aus dem Table of Contents ein. Schlägt das Einspielen eines Produktbausteins fehl, bricht die Methode sofort ab.

### deployTables

Spielt alle Tabelleninhalte aus dem Table of Contents ein. Schlägt das Einspielen eines Tabelleninhalts fehl, bricht die Methode sofort ab.

### deployEnums

Spielt alle Aufzählungsinhalte aus dem Table of Contents ein. Schlägt das Einspielen eines Aufzählungsinhalts fehl, bricht die Methode sofort ab.

### updateStatus

Ändert den Status einer Table-of-Contents-Version. Dazu sind vier Status-Übergänge

definiert, von denen einer als Parameter übergeben werden muss:

- `COMPLETE` setzt eine Version vom Status `PENDING` auf `DEPLOYED`. Dieser Status-Übergang wird intern am Ende eines Deployments durchgeführt.
- `ACTIVATE` setzt eine Version vom Status `DEPLOYED` auf `ACTIVE`. Eine eventuell zuvor aktive Version (mit gleicher Modell-Version und ToC-Namen) erhält den Status `HISTORIC`.
- `DEACTIVATE` setzt eine Version vom Status `ACTIVE` auf `HISTORIC`. Dadurch ist dann keine Version mehr aktiv, was zu Fehlern zur Laufzeit führen kann.
- `REACTIVATE` setzt eine Version vom Status `HISTORIC` auf `ACTIVE`. Eine eventuell zuvor aktive Version (mit gleicher Modell-Version und ToC-Namen) erhält den Status `HISTORIC`.

Der neue Status wird zurückgegeben, wenn der Statuswechsel funktioniert hat. Ging etwas schief, weil z.B. die Version nicht den passenden Ausgangszustand hat, wird ein leerer Optional-Wert zurückgegeben.

Wird eine Wildcard ("*") als Modellversion oder Produktdatenversion angegeben, so werden alle ToC Versionen selektiert, die die angegebenen Kriterien erfüllen und einen zum Statusübergang passenden Status haben. Das Statusupdate wird nur dann ausgeführt, wenn es genau eine solche ToC Version gibt.

**delete**

Löscht Table-of-Contents-Versionen mit allen zugehörigen Produkten, Tabellen- und Aufzählungsinhalten. Falls eine Wildcard als Modell-/Produktversion angegeben wird, werden alle passende Table-of-Contents-Versionen, die nicht den Status `ACTIVE` haben, gelöscht. Optional kann ein Status als Parameter angegeben werden. In dem Fall werden alle Table-of-Contents-Versionen, die sich in diesem Status befinden, gelöscht.

Wird eine Wildcard ("*") als Modellversion oder Produktdatenversion angegeben, so werden alle ToC Versionen gelöscht, die die angegebenen Kriterien erfüllen. Aktive ToC Versionen können nicht gelöscht werden.

# Kommandozeilen-Client

Der Kommandozeilen-Client für das Produktdaten-Deployment wird mit dem Client im `productdata-jpa-deployment-client-<version>.jar` ausgeliefert. Mit einem einfachen Aufruf mit `java -jar` werden die verfügbaren Befehle angezeigt:

```
Usage: <command> <serviceURL> -n=<tocName> [Parameter...]
Commands:
  deploy  Deploys all product data referenced in a table of contents.
  status  Updates the status of a product data version.
```

```
  delete  Deletes a product data version.
```

Der Kommandozeilen-Client verwendet zum Auffinden der notewendigen Dateien den Java Classpath. Damit wird sichergestellt, dass alle benötigten Ressourcen gefunden werden. Außerdem können dadurch die Ressourcen sowohl in JAR/ZIP oder als Dateien vorliegen. Um einen spezifischen Classpath in Kombination mit dem Client-JAR zu verwenden, muss der Befehl wie folgt aussehen:

```
java -cp "./path/to/produkt.jar:./productdata-jpa-sample-product.jar"
org.faktorips.runtime.productdata.jpa.deployment.client.ProductDataDeploymentC
lientCLI [...]
```

Die Befehle werden im folgenden genauer erläutert. Die Service-URL muss zur REST-API führen. Der ToC-Name kann frei gewählt werden und dient, zusammen mit Modell- und Produkt-Version, zur Identifikation der ToC-Version für weitere Befehle und den DbProductDataProvider.

## deploy

Mit dem Befehl `deploy` werden alle in einem Table of Contents verzeichneten Produktdaten (Produktbausteine, Tabellen- und Aufzählungsinhalte) eingespielt. Die Parameter dazu müssen wie folgt angegeben werden:

```
Missing required options [-n=<tocName>, params[0]=<serviceURL>]
Usage: deploy [-c=<comment>] [-m=<modelVersion>] -n=<tocName>
              [-P=<deployPassword>] [-t=<tocPath>] [-u=<userName>]
              [-U=<deployUser>] [-v=<version>] <serviceURL>
Deploys all product data referenced in a table of contents.
      <serviceURL>            The URL to the REST API
  -u, --user=<userName>       [OPTIONAL] The name of the user deploying the
                                product data. Uses the current user as
default.
                                Default: <current user>
  -c, --comment=<comment>     [OPTIONAL] A comment to distinguish the deployed
                                product data.
  -t, --tocPath=<tocPath>     The path (on the classpath) to the table of
                                contents xml file.
                              [REQUIRED] for deploy and when either
                                -m/--modelVersion or -v/--version are missing.
  -n, --name=<tocName>        The name of the product project to be used in
the
                                database.
  -m, --modelVersion=<modelVersion>
                              The version of the model(s) the product data is
```

```
                                based on.
                              If this parameter is omitted, the model version
                                is read from the table of contents files of
the
                                model(s), which must be on the classpath and
                                referenced in the <tocPath-without-xml>.model.
                                properties file.
  -v, --version=<version>     The version of the product data.
                              If this parameter is omitted, the version is
read
                                from the table of contents file referenced
with
                                the <tocPath>.
  -U, --deployUser=<deployUser>
                              The user used to authenticate against the
                                deployment service. Do not set this parameter
                                if there is no authentication configured.
  -P, --deployPassword=<deployPassword>
                              The password used to authenticate against the
                                deployment service.
```

Die Parameter entsprechen den für den Client-Builder definierten, sowie dem User und Kommentar des Client-Befehls createTocVersion. Wird der User nicht angegeben, verwendet das Kommandozeilen-Tool den Namen des am System angemeldeten Benutzers.

### status

Der Befehl `status` entspricht dem Client-Befehl updateStatus und benötigt den Namen eines Status-Übergangs als Parameter:

```
Missing required options [-x=<statusTransition>, -n=<tocName>,
params[0]=<serviceURL>]
Usage: status [-m=<modelVersion>] -n=<tocName> [-P=<deployPassword>]
              [-t=<tocPath>] [-U=<deployUser>] [-v=<version>]
              -x=<statusTransition> <serviceURL>
Updates the status of a product data version.
      <serviceURL>            The URL to the REST API
  -x, --transition=<statusTransition>
                              The change for a product version's status.
  -t, --tocPath=<tocPath>     The path (on the classpath) to the table of
                                contents xml file.
                              [REQUIRED] for deploy and when either
                                -m/--modelVersion or -v/--version are missing.
  -n, --name=<tocName>        The name of the product project to be used in
```

```
the
                                database.
  -m, --modelVersion=<modelVersion>
                              The version of the model(s) the product data is
                                based on.
                              If this parameter is omitted, the model version
                                is read from the table of contents files of
the
                                model(s), which must be on the classpath and
                                referenced in the <tocPath-without-xml>.model.
                                properties file.
                              May be wildcard '*' for status update or delete
  -v, --version=<version>     The version of the product data.
                              If this parameter is omitted, the version is
read
                                from the table of contents file referenced
with
                                the <tocPath>.
                              May be wildcard '*' for status update or delete
  -U, --deployUser=<deployUser>
                              The user used to authenticate against the
                                deployment service. Do not set this parameter
                                if there is no authentication configured.
  -P, --deployPassword=<deployPassword>
                              The password used to authenticate against the
                                deployment service.
```

## delete

Der Befehl `delete` entspricht dem Client-Befehl delete:

```
Missing required options [-n=<tocName>, params[0]=<serviceURL>]
Usage: delete [-m=<modelVersion>] -n=<tocName> [-P=<deployPassword>]
              [-s=<status>] [-t=<tocPath>] [-U=<deployUser>] [-v=<version>]
              <serviceURL>
Deletes a product data version.
      <serviceURL>            The URL to the REST API
  -s, --status=<status>       [OPTIONAL] The status of the version(s) that
                                should be deleted.
  -t, --tocPath=<tocPath>     The path (on the classpath) to the table of
                                contents xml file.
                              [REQUIRED] for deploy and when either
                                -m/--modelVersion or -v/--version are missing.
  -n, --name=<tocName>        The name of the product project to be used in
the
```

```
                                database.
  -m, --modelVersion=<modelVersion>
                              The version of the model(s) the product data is
                                based on.
                              If this parameter is omitted, the model version
                                is read from the table of contents files of
the
                                model(s), which must be on the classpath and
                                referenced in the <tocPath-without-xml>.model.
                                properties file.
                              May be wildcard '*' for status update or delete
  -v, --version=<version>     The version of the product data.
                              If this parameter is omitted, the version is
read
                                from the table of contents file referenced
with
                                the <tocPath>.
                              May be wildcard '*' for status update or delete
  -U, --deployUser=<deployUser>
                              The user used to authenticate against the
                                deployment service. Do not set this parameter
                                if there is no authentication configured.
  -P, --deployPassword=<deployPassword>
                              The password used to authenticate against the
                                deployment service.
```

## Exit Codes

*Table 1. Table Exit Codes*

| Code | Beschreibung |
|---|---|
| 0 | Alle Befehle wurden korrekt übertragen |
| 1 | Der Service ist unter der angegebenen URL nicht erreichbar |

| Code | Beschreibung |
| --- | --- |
| 2 | Beim Übertragen der Befehle ist ein Fehler aufgetreten. Das Log gibt weitere Informationen aus<br><br>:jbake-title: Maven-Plugin :jbake-type: section :jbake-status: published<br><br>:sample-dir: ../../../../../productdata-jpa-sample :icons: font<br><br>=== Maven-Plugin<br><br>Wird ein Produktprojekt mit Maven gebaut, kann auch das Deployment in die Datenbank darüber erfolgen. Dazu muss das `productdata-jpa-deployment-maven-plugin` in der `pom.xml` eingebunden werden.<br><br>[source,xml,indent=0] ---- <pluginManagement> <!--1-→ <plugins> <plugin> <groupId>org.faktorips.runtimejpa</groupId> <artifactId>productdata-jpa-deployment-maven-plugin</artifactId> <version>${project.version}</version> <executions> <execution> <goals> <goal>deploy</goal> </goals> </execution> </executions> <configuration> <serviceURL>${product.deployment.service.url}</serviceURL> </configuration> </plugin> </plugins> </pluginManagement> ---- <1> Das Plugin-Management kann für alle Projekte im Parent POM eingestellt werden. Alternativ kann diese Konfiguration mit der folgenden direkt im Produkt-Projekt erfolgen.<br><br>[source,xml,indent=0] ---- <plugins> <plugin> <groupId>org.faktorips.runtimejpa</groupId> <artifactId>productdata-jpa-deployment-maven-plugin</artifactId> <configuration> <tocName>sample-products</tocName> <comment>Deployed by Maven</comment> |