

---

**www.openbib.org • OpenBib Rechercheportal**

---

# **OpenBib Recherche-Portal**

## **Technische Dokumentation**

Oliver Flimm <flimm@openbib.de>  
Stand: 6.8.2006

---

**www.openbib.org • OpenBib Rechercheportal**

---



# Inhaltsverzeichnis

<b>1. Generelle Features</b>	<b>1</b>
1.1. OpenBib ist OpenSource . . . . .	1
1.2. OpenBib basiert auf Standardkomponenten und bibliothekarischen Standards . .	1
1.3. OpenBib ist über verschiedene Views bzw. Sichten individualisierbar . . . . .	2
1.4. OpenBib ist modular aufgebaut . . . . .	3
1.5. OpenBib integriert Suchmaschinen-Technologie . . . . .	3
1.6. OpenBib kann auf entfernte Datenbanken über das Z39.50-Protokoll zugreifen . .	4
1.7. OpenBib ist flexibel im internen Katalog- bzw. Metadaten-Format . . . . .	4
1.8. OpenBib bietet viel Flexibilität für den Datenimport . . . . .	4
1.8.1. Parametrisierbare Schnittstelle . . . . .	4
1.8.2. Plugin-Mechanismus für verschiedene Quell-Datenbanktypen . . . . .	5
1.8.3. Inkrementelles Live-Update über Open Library WebServices . . . . .	5
1.9. OpenBib läßt sich über WebServices an das Lokalsystem koppeln . . . . .	5
1.10. OpenBib bietet externe Zugriffsschnittstellen . . . . .	6
1.11. OpenBib verfügt über eine intelligente Lastverteilung . . . . .	6
1.12. OpenBib realisiert Content- bzw. Catalogue-Enrichment . . . . .	6
1.13. OpenBib bietet Neuzuganglisten als RSS-Feeds . . . . .	7
1.14. OpenBib bietet ein web-basiertes Administrationsinterface . . . . .	7
1.15. OpenBib respektiert die Sicherheit seiner Nutzer . . . . .	7
<b>2. Praxisbeispiel: OpenBib an der USB Köln</b>	<b>9</b>
2.1. Kataloge und Zahlen . . . . .	9
2.2. Technik . . . . .	9
2.3. Betrieb . . . . .	10
2.3.1. Nächtlicher Aufbau der Datenbanken . . . . .	10
2.3.2. Datenquellen und -formate . . . . .	10
2.3.3. Flexibler Einsatz in Projekten . . . . .	10
2.4. Fazit . . . . .	12
<b>3. Grundlegende Architektur</b>	<b>13</b>
3.1. Die datenbankabhängige Schicht 1 – <code>OpenBib::Search</code> . . . . .	13
3.2. Die datenbankübergreifende Recherche-Schicht 2 – <code>OpenBib::VirtualSearch</code> . .	13
3.3. Die Portal-Schicht 3 mit weiteren externen Zugriffsmechanismen . . . . .	15
3.3.1. Das Portal mit Session- und Benutzerverwaltung . . . . .	15
3.3.2. Externe Anbindung an die DigiBib . . . . .	16
3.3.3. Externe Anbindung an UK-Online . . . . .	16
3.4. Die Lastverteilungs-Schicht 4 . . . . .	16

<b>4. Konzepte und Interna von OpenBib</b>	<b>19</b>
4.1. Datenbankstruktur	19
4.1.1. Normdaten	19
4.1.2. Verknüpfungen	20
4.1.3. Zwischengespeicherte Kurztreffer	21
4.1.4. Anfangssuche	21
4.2. Kategorienformat	21
4.2.1. Allgemeine Informationen	21
4.2.2. Internationalisierung	22
4.2.3. Beispiel	24
4.3. Kaskadierungsmechanismus bei Templates und Stylesheets	27
4.3.1. Templates	28
4.4. Internationalisierung	29
4.4.1. Allgemeine Informationen zu GNU gettext	29
4.4.2. Technische Realisierung	30
4.5. Content-Enrichment	31
4.5.1. Hintergrund	31
4.5.2. Konzept von OpenBib	31
4.5.3. Technische Umsetzung	32
4.6. Suchmaschinen-Technologie von Xapian	33
4.6.1. Allgemeine Informationen zu Xapian	33
4.6.2. Strukturelle Integration in OpenBib	34
4.6.3. Technische Informationen	35
4.7. Konfiguration, Sessions und Benutzer	36
4.7.1. Konfiguration	36
4.7.2. Session-Management	37
4.7.3. Benutzer	38
<b>A. Xapian Perl-API</b>	<b>41</b>
A.1. Genereller Aufbau eines Index	41
A.2. Erzeugung oder Änderung eines Index	41
A.2.1. Allgemeiner Ablauf einer Indexierung	41
A.2.2. Weitere Hinweise	43
A.3. Recherche in einem Index	44
A.3.1. Drill-Downs - RSets und ESets	46
<b>B. Tabellen in config.mysql</b>	<b>49</b>
<b>C. Tabellen in session.mysql</b>	<b>51</b>
<b>D. Tabellen in pool.mysql</b>	<b>53</b>
D.1. Titel-Normdaten	53
D.2. Verfasser-Normdaten	53
D.3. Körperschafts-Normdaten	54
D.4. Schlagwort-Normdaten	54
D.5. Notations-Normdaten	55
D.6. Exemplar-Daten	55
D.7. Verknüpfungen zwischen den Normdaten	56

---

D.8. Recherche-Tabelle <code>search</code> . . . . .	56
<b>E. Objekt-Referenz</b> . . . . .	<b>57</b>
E.1. <code>OpenBib::Config</code> . . . . .	57
E.1.1. Methoden . . . . .	57
E.2. <code>OpenBib::Session</code> . . . . .	60
E.2.1. <code>new</code> . . . . .	60
E.2.2. <code>_init_new_session</code> . . . . .	60
E.2.3. <code>is_valid</code> . . . . .	60
E.2.4. <code>load_queryoptions</code> . . . . .	60
E.2.5. <code>dump_queryoptions</code> . . . . .	60
E.2.6. <code>merge_queryoptions</code> . . . . .	61
E.2.7. <code>get_queryoptions</code> . . . . .	61
E.2.8. <code>get_viewname</code> . . . . .	61
E.2.9. <code>get_profile</code> . . . . .	61
E.2.10. <code>set_profile</code> . . . . .	61
E.2.11. <code>get_resultlists_offsets</code> . . . . .	61
E.2.12. <code>get_mask</code> . . . . .	62
E.2.13. <code>set_mask</code> . . . . .	62
E.2.14. <code>set_view</code> . . . . .	62
E.2.15. <code>set_dbchoice</code> . . . . .	62
E.2.16. <code>get_dbchoice</code> . . . . .	62
E.2.17. <code>clear_dbchoice</code> . . . . .	62
E.2.18. <code>get_number_of_dbchoice</code> . . . . .	63
E.2.19. <code>get_number_of_items_in_resultlist</code> . . . . .	63
E.2.20. <code>get_items_in_resultlist_per_db</code> . . . . .	63
E.2.21. <code>get_all_items_in_resultlist</code> . . . . .	63
E.2.22. <code>get_max_queryid</code> . . . . .	63
E.2.23. <code>get_searchquery</code> . . . . .	63
E.2.24. <code>get_number_of_items_in_collection</code> . . . . .	64
E.2.25. <code>get_items_in_collection</code> . . . . .	64
E.2.26. <code>set_item_in_collection</code> . . . . .	64
E.2.27. <code>clear_item_in_collection</code> . . . . .	64
E.2.28. <code>updatelastresultset</code> . . . . .	64
E.2.29. <code>clear_data</code> . . . . .	64
E.2.30. <code>DESTROY</code> . . . . .	65
E.3. <code>OpenBib::User</code> . . . . .	65
E.3.1. <code>new</code> . . . . .	65
E.3.2. <code>get_cred_for_userid</code> . . . . .	65
E.3.3. <code>get_username_for_userid</code> . . . . .	65
E.3.4. <code>get_userid_for_username</code> . . . . .	65
E.3.5. <code>get_userid_of_session</code> . . . . .	66
E.3.6. <code>get_targetdb_of_session</code> . . . . .	66
E.3.7. <code>get_targettype_of_session</code> . . . . .	66
E.3.8. <code>get_profilename_of_profileid</code> . . . . .	66
E.3.9. <code>get_profiledb_of_profileid</code> . . . . .	66
E.3.10. <code>get_number_of_items_in_collection</code> . . . . .	66
E.3.11. <code>get_all_profiles</code> . . . . .	67

E.3.12. authenticate_self_user . . . . .	67
E.3.13. DESTROY . . . . .	67
<b>F. Module und Objekte geordnet nach Funktion bzw. Schicht</b>	<b>69</b>
F.1. Schichtübergreifend . . . . .	69
F.1.1. Handler . . . . .	69
F.1.2. Objekte . . . . .	69
F.2. Schicht 1 - Zugriff auf Katalogdatenbank . . . . .	69
F.2.1. Handler . . . . .	69
F.3. Schicht 2 - Zugriff auf versch. Datenbanken der Schicht 1 . . . . .	69
F.3.1. Handler . . . . .	69
F.4. Schicht 3 - Rechercheportal . . . . .	69
F.4.1. Handler . . . . .	69
F.5. Schicht 4 - Lastverteilung . . . . .	70
F.5.1. Handler . . . . .	70
<b>Literaturverzeichnis</b>	<b>71</b>

# 1. Generelle Features

Um einen schnellen Überblick von den Fähigkeiten und Möglichkeiten des OpenBib Recherche-Portals in der Version 2.0 und höher zu bekommen sowie eine Entscheidungshilfe für einen möglichen Einsatz zu geben, beginnt dieses technische Handbuch mit einer Aufzählung der generellen Features des Portals.

## 1.1. OpenBib ist OpenSource

Bei dem OpenBib Recherche-Portal handelt es sich um OpenSource-Software. Damit ist eine maximale Freiheit bei der Anpassung an die individuellen Anforderungen gewährleistet. Darüber hinaus ist die Eintrittsschwelle in die Änderung oder Erweiterung des Portals durch die Verwendung des DBMS Mysql (Model), des Template Toolkits (View), der Programmiersprache Perl (Controller) sowie von GNU gettext (I18N/L10N) äußerst niedrig. Schließlich fallen selbstverständlich keine (Lizenz-)Kosten an.

## 1.2. OpenBib basiert auf Standardkomponenten und bibliothekarischen Standards

OpenBib verwendet durchgängig Standardkomponenten. Es sind dies

**Apache Webserver mit mod\_perl** Durch die Verwendung des meistgenutzten OpenSource Webservers der Welt mit mod\_perl ist OpenBib als Webanwendung direkt in den Webserver integriert. Damit wird eine schnellstmögliche Reaktionszeit für die Webanwendung erreicht.

**MySQL DBMS** MySQL ist eines der meist genutzten OpenSource Datenbanksysteme. Durch die Verwendung der in MySQL integrierten Volltextsuche ist eine sehr schnelle Indizierung und Recherche möglich.

**Perl** Die Programmiersprache Perl zeichnet sich durch eine weite Verbreitung und eine niedrige Eingangsschwelle aus (Motto: There's more than one way to do it).

**Perl Template Toolkit** Durch das Perl Template Toolkit wird die gesamte Darstellung im Recherche-Portal realisiert. Durch seine Einfachheit aber auch seiner Mächtigkeit kann durch die Abspaltung der Ansichts-Komponente an Webdesigner oder Nicht-Programmierer sehr gut eine Arbeitsteilung erreicht werden. Im Vergleich zu XSLT ist die Learning Curve des Template Toolkits sehr flach, so daß z.B. eine Auslagerung an einen Bibliothekar problemlos möglich ist.

**CSS** Für die Darstellung werden in den Templates vielfältige CSS-Klassen verwendet, die zentral angepasst werden können.

**GNU gettext** Für die Anpassung des Portals an andere Sprachen hat sich in vielen Projekten (z.B. bei KDE mit mehr als 40 Sprachversionen) GNU gettext mit seinen Werkzeugen bewährt. Während andere Systeme z.B. numerische Message-Idendifier verwenden und dadurch etwaig verwendete Templates unlesbar machen, wird bei GNU gettext die jeweilige Meldung in einer Basissprache – bei OpenBib ist dies Deutsch – mit einem Methodenaufruf gekapselt (inkl. weiterer möglicher Parameter), so daß die Templates durchschaubar bleiben.

**SOAP** Für die Kommunikation mit externen Systemen wird das Standard-Protokoll SOAP verwendet.

**RSS** Über RSS-Feeds können verschiedene Informationen für das Gesamtportal bzw. View-basierte Unter-Portale angeboten werden.

**UTF-8** Die Darstellung und interne Verarbeitung der Daten geschieht im Encoding UTF-8. Damit sind auch Datenbestände, die nicht auf ISO-8859-1 basieren, vollständig integrierbar.

**log4Perl** Durch die Verwendung des Logging-Frameworks log4Perl (Perl-Pendant zum bekannten log4j aus der Java-Welt) kann zu jeder Zeit an jeder Stelle des Portals ein Debugging erfolgen bzw. die genaue Abarbeitung verfolgt werden.

**Xapian** Mit der Verwendung von Suchmaschinen-Technologie aus dem OpenSource Projekt Xapian (<http://www.xapian.org/>) kann alternativ zur herkömmlichen Suche via SQL weitere Funktionalität, wie z.B. Drill-Downs für große Treffermengen realisiert werden.

Das Datenmodell und das zugrundeliegende Meta-Datenformat basieren auf dem bibliothekarischen Standard MAB2. Darüber hinaus ist seit der Ur-Version des Recherche-Portals im Jahr 1997 sehr viel bibliothekarisches Know-How in die Entwicklung mit eingeflossen.

### 1.3. OpenBib ist über verschiedene Views bzw. Sichten individualisierbar

Mit OpenBib können über das Merkmal *View* bzw. *Sicht* ausgehend von einer zentralen Installation mit geringstem Aufwand individuell gestaltete andere Portale erstellt werden. Hierzu wird ein einfacher Mechanismus kaskadierender Templates und Stylesheets verwendet. Dieser Mechanismus funktioniert sowohl view- wie auch datenbankabhängig.

Wird für eine Datenbank oder einen View ein anderer Aufbau eines Templates gewünscht, so reicht es, ein entsprechendes Unterverzeichnis anzulegen, das zugehörige Standard-Template in das Unterverzeichnis zu kopieren und dort zu verändern. Damit müssen nur die Templates kopiert werden, die effektiv auch anzupassen sind. Dies gewährleistet eine sehr gute Übersicht der getätigten Änderungen zu verschiedenen Sichten und Datenbanken im Gegensatz zu dem häufig verwendeten Ansatz in anderen Portalen grundsätzlich alle Templates zu kopieren.

Neben den Templates werden ebenfalls die Stylesheets über diesen Mechanismus individualisierbar gemacht.

Über den Mechanismus kaskadierender Templates hinaus können datenbankabhängige (internationalisierbare bzw. lokalisierbare) Bezeichner für einzelne Kategorien definiert werden. Damit lassen sich einzelne Kategorien unter Beibehaltung ihres Datenbankbezeichners für spezielle thematische Datenbanken (siehe unten z.B. Digitale Einbandsammlung) 'umetikettieren' – und dies mehrsprachig.

Konkrete Beispiele sind

- Das KUG Recherche-Portal  
(<http://kug.ub.uni-koeln.de/>)
- Die Digitale Einbandsammlung der USB Köln  
(<http://einbandsammlung.ub.uni-koeln.de/>)
- Die Virtuelle Bibliothek Elise und Helene Richter  
(<http://richterbibliothek.ub.uni-koeln.de/>)
- Die Virtuelle Bibliothek Historische Bestände im Rheinland  
(<http://rheinlandbib.ub.uni-koeln.de/>)

## 1.4. OpenBib ist modular aufgebaut

OpenBib ist in seiner derzeitigen Version sehr modular konzipiert worden. Dies zeigt sich zu allererst in der Verwendung des MVC-Design-Patterns (Trennung von Modell, View, Controller) sowie der Auslagerung aller verwendeten Texte in internationalisierbare bzw. lokalisierbare GNU gettext Message-Kataloge.

Damit können – ganz praktisch gesehen – entsprechende Arbeiten wie das visuelle Design des Portals (Templates, Stylesheets) oder die Übersetzung der Texte (GNU gettext) an verschiedene – nicht im Programmierumfeld zu suchende – Mitarbeiter ausgelagert werden.

Darüber hinaus können andere Teile modular ausgewechselt werden – wenn dies gewünscht wird. So läßt sich u.a. die anfängliche Suche über einen oder alle Kataloge z.B. von der Realisierung über eine MySQL-Volltext-Datenbank auf andere Volltext-Datenbanken oder gar Suchmaschinen-technologie ändern. Letzteres wurde bereits mit der Suchmaschinen-Technologie des OpenSource Projektes Xapian realisiert.

Das neue Verfahren muß als Minimum lediglich die 'Einsprünge' (Katalogschlüssel) in den bibliographischen Teil der MySQL-Datenbank liefern. Ganz konkret basierten die ersten Versionen in den Jahren 1997-2000 für die anfängliche Suche auf der Volltextdatenbank `freeWALS-sf` und wurde im Jahr 2000 auf die integrierte Volltextsuche von MySQL umgestellt.

## 1.5. OpenBib integriert Suchmaschinen-Technologie

In OpenBib ist die Suchmaschinen-Technologie des OpenSource Projektes Xapian<sup>1</sup> in Form eines alternativen Recherche-Backends integriert. Mit dieser Technologie können dem Nutzer bei

---

<sup>1</sup><http://www.xapian.org/>

großen Treffermengen sog. Drill-Downs angeboten werden. Drill-Downs bestehen aus den in der Treffermenge meistgenutzten Begriffen, die kategorisiert (Verfasser, Titelbegriff, Schlagworte, Erscheinungsjahr) dargestellt werden und dem Benutzer die Möglichkeit bieten,

- sich einen generellen Überblick von der Treffermenge zu machen
- durch Auswahl eines der Begriffe die bisherige Treffermenge weiter zu reduzieren

## **1.6. OpenBib kann auf entfernte Datenbanken über das Z39.50-Protokoll zugreifen**

Die Recherche in Z39.50-Datenbanken ist nun experimentell in OpenBib integriert. Derzeit ist bereits mit der USB Köln ein Beispielkatalog über das Z39.50-Protokoll erfolgreich integriert mit funktionierender Recherche, Trefferlisten- sowie Einzeltrefferanzeige.

## **1.7. OpenBib ist flexibel im internen Katalog- bzw. Metadaten-Format**

Das standardmäßig verwendete Katalog- bzw. Meta-Datenformat basiert auf dem deutschen Bibliotheksstandard MAB2. Dennoch ist die grundsätzliche Datenbankstruktur von OpenBib nicht auf MAB2 statisch festgelegt. Sie ist so universell, daß sich auch andere Datenformate wie z.B. das anglo-amerikanische MARC21 anstelle von MAB2 verarbeiten und in der Datenbank ablegen lassen. Selbstverständlich würde ein solcher Wechsel verschiedene Änderungen – z.B. in den Templates oder der Konfiguration – bedingen, eine Anpassung der Datenbank selbst an ein Format ist jedoch nicht notwendig.

## **1.8. OpenBib bietet viel Flexibilität für den Datenimport**

### **1.8.1. Parametrisierbare Schnittstelle**

OpenBib bietet ausgehend vom auf MAB2 basierten Meta-Datenformat eine für jede Datenbank individualisierbare und parametrisierbare Importschnittstelle.

Es kann definiert werden

- welche Kategorien für die Kurztrefferliste optimierend aufbereitet werden sollen,
- welche Kategorien in den einzelnen Normdatenbereichen für
  - eine Volltextsuche nutzbar sein sollen,
  - eine String- bzw. Wortanfangs-Suche nutzbar sein sollen,
  - die Anfangsrecherche nutzbar sein sollen sowie

- grundsätzlich ignoriert (blacklisted) werden sollen.

Kategorieeitig ist die grundsätzliche Strategie alles zu importieren, was nicht geblacklisted ist.

Wie schon bei den Templates und den Stylesheets gibt es auch hier eine Standardkonfiguration, die für alle Datenbanken gilt, für die nicht explizit eine eigene Konfiguration erstellt wurde.

### 1.8.2. Plugin-Mechanismus für verschiedene Quell-Datenbanktypen

Ausgehend von einer Standardkonvertierung (Sisis-Format) können im Programm für den automatisierten Import an verschiedenen Stellen des Import-Ablaufes datenbankspezifische Plugins eingebracht werden, in die man Spezialanpassungen ausgelagern kann. Grundlegende Phasen, in die man eingreifen kann sind:

**Einsammeln der Daten** An dieser Stelle können alternative Zugriffsmechanismen – wie z.B. OAI – in externe Plugin-Programme ausgelagert werden.

**Vorbereitung der Daten** An dieser Stelle können alternative Vorbereitungsaktionen – wie z.B. Teilkatalogbildung – in externe Plugin-Programme zwischengeschaltet werden.

**Konvertierung der Daten** An dieser Stelle können alternative Konvertierungsroutinen – z.B. in das MAB2 basierte Meta-Datenformat – zwischengeschaltet werden.

**Einladen der Daten** An dieser Stelle können alternative Behandlungen der Daten zwischengeschaltet werden.

### 1.8.3. Inkrementelles Live-Update über Open Library WebServices

Zusätzlich zu einem Komplet-Update eines Kataloges können neue bzw. geänderte Datensätze von einem Target, das über die WebServices OLWS (Open Library WebServices) angesprechbar ist, auch live inkrementell aggregiert und in der zugehörigen OpenBib-Datenbank aktualisiert werden. Damit können gezielt Datensätze aus einem definierten Datumsbereich verarbeitet werden.

## 1.9. OpenBib läßt sich über WebServices an das Lokalsystem koppeln

Über das Teilprojekt Open Library WebServices (OLWS) von OpenBib können verschiedene Informationen aus Lokalsystemen (derzeit nur Sisis SunRise) in das Recherche-Portal eingebunden werden. Es sind dies

- Authentifizierung an OpenBib anhand der Kennung und OPAC-Pin im jeweiligen Lokalsystem
- Anzeige der Nutzerdaten (Name, Anschrift, Sperrung, Sperrgrund usw.)
- Anzeige des Nutzerkontos mit

- Gebühren
  - vorgemerkten Medien
  - bestellte Medien
  - ausgeliehene Medien
  - überzogene Medien
- Anzeige der Exemplardaten (Signatur, Standort, ausgeliehen usw.). Über diesen Service wird sekundeaktuell der Ausleihstatus zu einem Medium in OpenBib bestimmt.
  - Zugriff auf die Katalogdaten ausgehend von Katalogschlüsselwörtern. Über diesen Service können Datenübernahmen in externe Systeme realisiert werden.

Aufgrund der Offenheit der Schnittstelle muss sie nicht auf ein Lokalsystem beschränkt bleiben, sondern kann auf andere ausgedehnt werden.

### **1.10. OpenBib bietet externe Zugriffsschnittstellen**

Für die Recherche von außen werden verschiedenen Zugriffsschnittstellen angeboten. Neben einer HTML-basierten Zugriffsschnittstelle über die die Digitale Bibliothek NRW des hbz sowie das Hochschulmanagement-System UK-Online der Universität zu Köln angebunden sind, wird eine SOAP-basierte Schnittstelle für die Webservice-basierte Anbindung bereitgestellt.

Neben dieser Nutzbarkeit von OpenBib durch andere bestehen weitere Mechanismen, um eine Integration externer Angebote in OpenBib zu realisieren. So können bereits getätigte Suchanfragen z.B. an externe Datenbanken und Portale weitergeleitet werden (Digitale Bibliothek, Fernleihe, Aufsatzbestellung, EZB, DBIS, MedPilot) – im Falle der Digitalen Bibliothek NRW sogar unter Mitnahme einer etwaig in OpenBib erfolgten Authentifizierung.

### **1.11. OpenBib verfügt über eine intelligente Lastverteilung**

Beim Aufruf des Portals wird der Nutzer an einen der verfügbaren OpenBib Portal-Server weitergeleitet. Maßgabe für die Weiterleitung ist die generelle Ansprechbarkeit (connect) und Auslastung des Servers (load) sowie seine grundlegende Funktionsfähigkeit (MySQL).

Die für die Lastverteilung verwendeten Server können zentral konfiguriert werden. Damit kann bei Ausfall oder bei Wartungsarbeiten der jeweilige Server aus der Verteilung herausgenommen werden, um in aller Ruhe entsprechende Arbeiten vorzunehmen.

### **1.12. OpenBib realisiert Content- bzw. Catalogue-Enrichment**

Über eine zentrale Anreicherungsdatenbank können zusätzliche Inhalte katalogübergreifend in die vorhandenen Katalogdaten eingebündelt werden. Grundlage ist eine vorhandene ISBN/ISSN.

Hiermit ist z.B. die Ergebnisanreicherung mit Inhaltsverzeichnissen realisierbar. Die Implementierung der Ergebnisanreicherung in OpenBib ist jedoch so allgemein gehalten, daß sie sich auch mit anderen 'Zusatz'-Informationen nutzen lässt. So können beliebige Informationen wie Abstracts, Stichwortverzeichnisse u.ä. dort zentral abgelegt werden und sind für alle Kataloge des OpenBib-Portals nutzbar.

### **1.13. OpenBib bietet Neuzugangslisten als RSS-Feeds**

Für jeden Katalog können Neuzugangslisten bereitgestellt werden. Hierzu bietet sich die XML-basierte RSS-Technologie an. Im Gegensatz zu einer gewöhnlichen Präsentation über simple Webseiten bieten RSS-Feeds den Nutzern durch die geschickte Verwaltung über spezialisierte Programme deutlich mehr Nutzungsmöglichkeiten. So können solche Programme sich um die Sichtung der Daten kümmern, schon aufgerufene Titel von den noch nicht aufgerufenen farblich trennen, Informationen archivieren, Data-Mining in Verbindung mit spezialisierter Suchtechnologie einsetzen usw.

Grundlage für die in den RSS-Feeds angezeigten Informationen ist das Neuaufnahmedatum im Katalog. Neben einer allgemeinen Neuzugangsliste (unter der Rubrik RSS in OpenBib) stehen in den einzelnen Titeln auf Wunsch Verfasser/Personen-, Körperschafts/Urheber-, Notations- sowie Schlagwortspezifische Neuzugangslisten als RSS-Feed zur Verfügung. Erkennbar sind sie durch ein orangenes RSS-Icon. Damit können sich Nutzer ausgehend von einem gefundenen Titel z.B. über thematisch verwandte – z.B. im Fall von Notation oder Schlagwort – informieren lassen.

Für einzelne Sichten in OpenBib können einzelne Feeds auf Wunsch sowohl im Kontext einer automatischen Browsererkennung angeboten werden, wie sie z.B. der Browser Firefox ab der Version 1.5 besitzt (dynamisches Lesezeichen), oder in Listenform.

### **1.14. OpenBib bietet ein web-basiertes Administrationsinterface**

Die Einrichtung einzelner Datenbanken, Zusammenfassung von Datenbanken in Views, die Konfiguration der RSS-Feeds sowie eine Beobachtung aktiver Sessions (Suchanfragen usw.) kann über ein web-basiertes Administrationsinterface erfolgen. Damit läßt sich auch eine große Anzahl an Katalogen unproblematisch konfigurieren.

### **1.15. OpenBib respektiert die Sicherheit seiner Nutzer**

OpenBib ist auch ohne die Aktivierung der sicherheitskritischen Browser-Sprache JavaScript vollständig nutzbar. Ebenso verwendet es keine Cookies. Damit wird ein sicherheitsbewußter Nutzer nicht dazu gezwungen, für die Verwendung von OpenBib JavaScript in seinem Browser zu aktivieren.

Ebenso werden Nutzerdaten bei einer Authentifizierung in OpenBib an einem Bibliothekssystem nach Ende einer Session selbstverständlich wieder gelöscht.



## 2. Praxisbeispiel: OpenBib an der USB Köln

Anhand des Einsatzes an der USB Köln sollen die Möglichkeiten des OpenBib Recherche-Portals illustriert werden.

OpenBib wird an der USB Köln als KUG Recherche-Portal eingesetzt. Im Projekt 'Kölner UniversitätsGesamtkatalog' (KUG) wird unter Mitwirkung der Universitäts- u. Stadtbibliothek (USB), der Institute und Seminare sowie der Zentralbibliothek für Medizin (ZBMED) seit Anfang 2002 ein universitätsweiter bibliothekarischer Gesamtkatalog aufgebaut.

### 2.1. Kataloge und Zahlen

Mit Ende des Jahres 2005 sind im KUG-Projekt insgesamt 145 Institute und Seminare neben den Katalogen der USB Köln und der ZBMED zusammengefasst, von denen wiederum 139 (in 104 Datenbanken) im KUG Recherche-Portal sichtbar sind.

Neben den Katalogen der Instituts- und Seminarbibliotheken wurden im KUG Recherche-Portal weitere Spezialkataloge erschlossen bzw. erst verwirklicht. Dazu gehören die Digitale Einbandsammlung der USB Köln, die Virtuelle Bibliothek Elise und Helene Richter, der Hochschulschriftenserver, EconBiz, das Graph Drawing Eprint-Archive, die Poetica-Sammlung, die Sammlung Kölner Zeitungsausschnitte, der Katalog der Bibliothek des Oratoriums Kevelaerinense sowie diverse separate Teilkataloge des USB-Bestandes (Lehrbuchsammlung, Lesesaal, Europäisches Dokumentationszentrum).

Insgesamt sind damit im KUG Recherche-Portal 119 verschiedene Katalogdatenbanken mit derzeit insgesamt 4942543 Titelaufnahmen vertreten.

### 2.2. Technik

Das KUG Recherche-Portal wird mit drei Doppel-Pentium-III Servern (1.16 GHz CPU, 4 GB RAM) im RAID-Level 1 betrieben. Alle Rechner werden im Rahmen der Lastverteilung genutzt, wobei einer der drei Rechner zusätzlich die eigentliche Verteilung übernimmt.

Neben diesen drei Produktions-Servern verfügen wir über ein Test- und Entwicklungssystem (300 EUR Standard-Desktop), auf dem vor einem Upgrade unter Beteiligung unserer Kollegen aus dem Haus bzw. aus den dezentralen Bibliotheken eine intensive Testphase durchgeführt wird. Erst wenn keine gravierenden Fehler gemeldet werden erfolgt die Umstellung der Produktionssysteme.

## 2.3. Betrieb

### 2.3.1. Nächtlicher Aufbau der Datenbanken

Alle im Recherche-Portal vorhandenen Daten werden ausgehend von den jeweiligen Quell-Systemen, auf denen die Katalogisierung stattfindet, nächtlich aktualisiert. Dazu werden die Daten auf den Quell-Systemen entladen sowie auf einem geschützten Bereich eines Webservers abgelegt. Von dort sammeln die einzelnen Server des Recherche-Portals die Daten automatisiert ein, wandeln sie um und spielen Sie in ihre lokalen MySQL-Recherchedatenbanken ein.

Unsere Erfahrung mit einem alternativen Recherche-Portal hatte zwischenzeitlich gezeigt, dass Recherchen direkt auf den Quell-Systemen die entsprechenden Systeme massiv belasteten – in den Modulen Katalogisierung, Erwerbung sowie Ausleihe waren ganz erhebliche Performance-Probleme zu verzeichnen. Die Entkopplung von Katalogisierungs- und Recherche-Datenbanken hat sich hier für beide Seiten als sehr vorteilhaft erwiesen.

### 2.3.2. Datenquellen und -formate

Neben den Daten der USB, ZBMED sowie der dezentralen Bibliotheken, die einheitlich aus Sisis SunRise-Systemen stammen, verarbeiten wir für unsere Spezialkataloge auch andere Daten. So speisen sich die Kataloge unseres Hochschulschriftenservers (KUPS) sowie des am Lehrstuhl für Informatik angesiedelten Graph Drawing E-print Archive (GDEA) über das OAI-Synchronisationsprotokoll.

Zusätzlich können wir derzeit automatisiert Quelldaten von Lars, Allegro, Bislok, LIDOS und FileMaker verarbeiten, dazu kommen abgeleitete Datenformate, die z.T. auf einem einfachen Lars-Derivat (Kategorieiname/Kategorieinhalt) beruhen.

Produktiv setzen wir derzeit jedoch nur ein Lars-Derivat für die Einbindung des Kataloges des Italienischen Kulturinstituts sowie unserer Zeitungsausschnitts-Sammlung ein.

Ausgehend von den USB-Katalogdaten bilden wir verschiedene abgeschlossene Teilkataloge. Diese basieren entweder auf einem speziellen Standort (Lehrbuchsammlung, Lesesaal) oder einem speziellen Ausleih-Nutzer (EDZ, realisiert über einen entsprechenden OLWS-Dienst für das Lokalsystem). Zusätzlich werden im Falle des EDZ-Katalogs dessen Daten mit einer speziellen Sachgruppenkategorie angereichert.

Schließlich verwenden wir für die Internetquellen der Virtuellen Fachbibliothek Wirtschaftswissenschaften (EconBiz) eine SQL-Schnittstelle für den Datenabzug.

Durch die Verwendung eines einheitlichen Meta-Datenformats als Grundlage können neue Datenbestände sehr einfach in das Portal integriert werden.

### 2.3.3. Flexibler Einsatz in Projekten

Durch die vielen Möglichkeiten der Anpassung war die Software des Recherche-Portals predestiniert für den Einsatz in weiteren Projekten[FliHof06].

Hier sind insbesondere drei Kataloge zu nennen: Die digitale Einbandsammlung der USB Köln, die Virtuelle Bibliothek Elise und Helene Richter sowie die Virtuelle Bibliothek Historische Bestände im Rheinland.

### **Digitale Einbandsammlung der USB Köln**

Im Projekt 'Digitale Einbandsammlung' wurde an der USB eine Einbanddatenbank, bestehend aus Einbandbeschreibungen und Bildern der jeweiligen Einbände, konzipiert und realisiert. Dazu wurde ein Scan-Workflow erarbeitet und programmtechnisch in einer eigenentwickelten Komponente OpenDIA – die ein weiteres Teilprojekt von OpenBib ist – umgesetzt, mit dem die eingescannten Einbandbilder erfasst, mit Meta-Daten angereichert und präsentiert werden.

Ausgehend vom KUG Recherche-Portal (OpenBib) wurden diese Abbildungen samt Daten bei Wahrung der Eigenständigkeit beider Komponenten – OpenBib und OpenDIA – visuell zu einem Ganzen zusammen gefügt. Diese visuelle Integration wurde durch die Flexibilität beider Komponenten erst ermöglicht – speziell durch die individualisierbaren Portal-Sichten in OpenBib.

Durch den pragmatischen Einsatz weitgehend bereits vorhandener, bekannter sowie beherrschbarer Techniken und Programme bei diesem Projekt konnte die USB innerhalb von nur 3 Monaten mit einem neuen interessanten Dienst an die Öffentlichkeit treten. Die feierliche Präsentation der 'Digitalen Einbandsammlung' geschah dabei im Rahmen der 10. Jahrestagung des Arbeitskreises für die Erfassung, Erschließung und Erhaltung historischer Bucheinbände, die an der USB Köln vom 22.-24. September 2005 stattgefunden hat. Eine ausführliche Zusammenfassung der Konzeption und der technischen Realisierung findet sich in [BoeFli:06].

### **Virtuelle Bibliothek Elise und Helene Richter**

Im Projekt 'Virtuelle Bibliothek Elise und Helene Richter', das Teil der NS-Provenienzforschung in der USB Köln ist, wurde für die in die von der USB aufgespurten Medien aus der ehemaligen Bibliothek der Schwestern Richter ein eigenständiges Recherche-Portal auf Basis individualisierbarer Portal-Sichten erstellt.

### **Virtuelle Bibliothek Historische Bestände im Rheinland**

Für die Arbeitsstelle *Historische Bestände im Rheinland* wurde ein Virtuelle Katalog konzipiert, in dem die Bestände kleiner historischer Bibliotheken des Rheinlandes zusammengetragen werden sollen.

Zum gegenwärtigen Zeitpunkt umfasst dieser virtuelle Katalog die Bestände folgender Bibliotheken:

- Bibliotheca Domus Presbyterorum Gaesdonck
- Heimatverein Königswinter
- Alte Pfarrbibliothek St. Dionysius Hürth-Gleuel

- Kreis- und Stadtbibliothek Kempen
- Bibliothek des Oratoriums Kevelaeriense

Darüber hinaus ist der Bestand der Rheinischen Abteilung der USB mit mehr als 117.000 Titeln integriert.

## 2.4. Fazit

Die Einsatz der eigenentwickelten Portal-Software OpenBib, die vollständig aus OpenSource-Komponenten aufgebaut ist, hat sich in der praktischen Arbeit als sehr vorteilhaft erwiesen:

- Erweiterungen werden umgehend selbständig vorgenommen sowie Probleme schneller gelöst.
- Von unseren Benutzern an uns herangetragene Wünsche werden schneller umgesetzt.
- Release-Zyklen der Software werden selbst vorgegeben.
- Die Integration mit anderen Software-Produkten über standardisierte Schnittstellen ist nun mit wenig Aufwand möglich.

## 3. Grundlegende Architektur

Die OpenBib-Portalsoftware-Suite ist in logischen Schichten organisiert. Ein Schaubild der generellen Architektur zeigt Abb. 3.1. Dieses Kapitel soll einen kurzen Abriß dieser Architektur geben, damit die größeren Zusammenhänge deutlich werden. Die Schichten von 'unten' nach 'oben' sind: Die datenbankabhängige Schicht, die datenbankübergreifende Schicht, die Portal-Schicht mit weiteren externen Zugriffsmechanismen (DigiBib, UK-Online) und schließlich die Lastverteilungsschicht. Diese Schichten werden bei der Interaktion mit einem Benutzer bei der Verwendung des OpenBib-Portals in entgegengesetzter Richtung von 'oben' nach 'unten' durchlaufen.

### 3.1. Die datenbankabhängige Schicht 1 – `OpenBib::Search`

In der untersten, der datenbankabhängigen Schicht greift das Perl-Modul `OpenBib::Search` auf eine feste Datenbank zu. Welche Datenbank dies ist, kann als Aufrufparameter dem Programm mitgegeben werden. Dadurch ist es möglich, das gleiche Programm für Recherchen über verschiedene Datenbanken zu nutzen. Das Modul selbst – wie auch alle anderen unter

`openbib/portal/perl/modules`

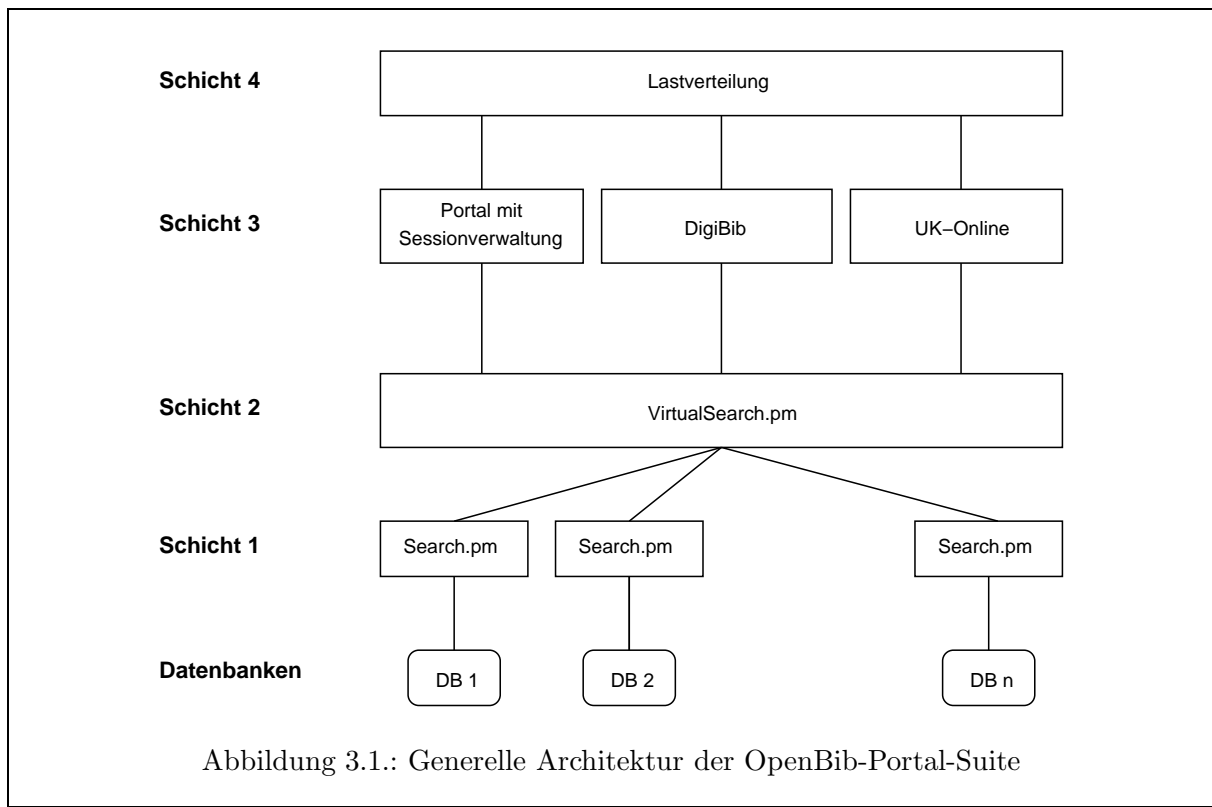
angesiedelten Module – ist als Erweiterung des Apache-Webservers via `mod_perl` realisiert. Dadurch arbeitet es außerordentlich schnell.

### 3.2. Die datenbankübergreifende Recherche-Schicht 2 – `OpenBib::VirtualSearch`

Die datenbankübergreifende Recherche geschieht in der über `OpenBib::Search` liegenden Schicht 2 (vgl. Abb. 3.1) mit dem Perl-Modul `OpenBib::VirtualSearch`. Aufgabe dieses Modules ist es, eine Suchanfrage und eine Anzahl an Recherche-Datenbanken vom Benutzer entgegenzunehmen und genau diese eine Suchanfrage sequentiell an jede der übergebenen Datenbanken zu schicken. Zu diesem Zweck werden entsprechende Utility-Routinen aus dem im `OpenBib::Search`-Kontext angesiedelten Modul `OpenBib::Search::Util` verwendet.

Die Ergebnisse der einzelnen Recherchen werden entgegengenommen und in Listenform aufbereitet. Die jeweiligen Titel selbst sind in dieser Gesamttrefferliste via URL mit einer Suchanfrage für genau diesen Einzeltreffer mit `OpenBib::Search` verknüpft. Bei der Auswahl eines einzelnen Treffers wird dadurch direkt in die zugehörige Datenbank via `OpenBib::Search` gesprungen.

Alle 'normalen' Verknüpfungs-Links (Normdaten, Hierarchien, etc.) beziehen sich – wir befinden uns nach der Auswahl eines Treffer wieder in Schicht 1 – auf genau diese eine Datenbank.



Über das große stilisierte 'G' bei den Normdaten kann jedoch auch an dieser Stelle wieder eine datenbankübergreifende Recherche nach genau dem zugehörigen Begriff gestartet werden. Der Benutzer hat somit bei einem Einzeltreffer immer die Möglichkeit selbst zu entscheiden, ob er sich bzgl. der Normdaten weiter im zugehörigen Katalog aufhalten möchte, oder katalogübergreifend Recherchieren möchte.

Da in jedem Katalog die Verknüpfungen bereits in der Datenbank existieren, ist ein 'sich treiben lassen' in einem festen Katalog außerordentlich schnell, während bei einer katalogübergreifenden Suche ein kleiner Tribut zu zahlen ist, da nun effektiv wieder alle Datenbanken mit einer 'neuen Suche' abgefragt werden.

Da `OpenBib::VirtualSearch` letztlich für die datenbankübergreifende Gewinnung der Rechercheergebnisse zuständig ist, werden nach erfolgreicher Recherche die Ergebnisse aufgeteilt nach den einzelnen Katalogen in der Session-Datenbank `session` 'gecached'. Durch das Caching der Rechercheergebnisse kann im Kontext von der Portal-Schicht 3 tieber das dort angesiedelte Modul `OpenBib::ResultLists` auch eine Trefferlistenfunktionalität geleistet werden, in der man über die Ergebnisse der letzten Recherche hinaus auch noch direkt auf die Ergebnisse jeder zuvor eingegebenen Suche in der aktuellen Session zugreifen kann.

## 3.3. Die Portal-Schicht 3 mit weiteren externen Zugriffsmechanismen

### 3.3.1. Das Portal mit Session- und Benutzerverwaltung

In der Portalschicht wird dem Benutzer sessionbasiert in einer aus zwei Frames bestehenden Web-Seite eine Arbeits- und Recherche-Oberfläche geliefert.

Der Einsprung-URL in das Portal wird durch das Modul `OpenBib::StartOpac` realisiert.

Das Modul `OpenBib::StartOpac` kann über verschiedene Parameter gesteuert werden, die z.B. für den direkten Sprung zu einem einzelnen Treffer einer Datenbank oder der Anzeige einer katalogeigenen Sicht des Portals Verwendung finden und gegebenenfalls an nachgelagerte Module durchgereicht werden. Seine wesentliche Aufgabe ist jedoch, eine eindeutige Session-ID zu generieren sowie ein Frameset mit zwei Frames bereitzustellen. Das obere Frame für die Navigation wird durch das Modul `OpenBib::HeaderFrame` gefüllt, das untere Frame standardmäßig zuerst durch das Modul `OpenBib::SearchFrame`, das dem Navigations-Element 'Einfache Recherche' bzw. 'Komplexe Recherche' entspricht.

Das Modul `OpenBib::HeaderFrame` ist für die Navigation im Portal inklusive Counter für die Merkliste zuständig.

In der Navigation werden folgende Funktionalitäten angeboten:

**Katalogauswahl** Die Auswahl der zu durchsuchenden Kataloge geschieht über eine entsprechende Auswahlseite, die durch das Modul `OpenBib::DatabaseChoice` generiert wird.

**Recherche** Die Recherche-Maske wird über einen Aufruf von `OpenBib::SearchFrame` dargestellt.

**Trefferliste** Ein Zugriff auf Trefferlisten aus vorangegangenen Recherchen geschieht über das Modul `OpenBib::ResultLists`, welches neben der verteilten Recherche auch für ein Caching der Ergebnis-Daten zuständig ist.

**Merkliste** Die Anzeige, Organisation sowie der Mailversand der Merkliste wird durch die Module `OpenBib::ManageCollection` und `OpenBib::MailCollection` übernommen. Bei Auswahl eines Treffers für die Merkliste wird `OpenBib::HeaderFrame` aktualisiert, und zeigt die neue Zahl gemerkter Treffer an.

**RSS** Über das Modul `OpenBib::RSSFrame` wird eine zu diesem View zugeordnete Liste an RSS-Feeds für die letzten 50 Neuaufnahmen der entsprechenden Kataloge ausgegeben.

**Mein OpenBib** Über das Modul `OpenBib::Login` kann sich der Benutzer am Portal authentifizieren und so von weiteren personalisierten Funktionalitäten profitieren. Dazu gehören neben generellen Benutzereinstellungen (`OpenBib::UserPrefs`) u.a. persistente Merklisten sowie Datenbankprofile (`OpenBib::DatabaseProfile`).

Das Modul gewährt dabei den Zugang auf Grundlage seiner eigenen Datenbank, die über das Modul `OpenBib::SelfReg` via Selbstregistrierung gefüllt wird, bzw. auf Grundlage von lokalen Bibliothekssystemen – hier wird konkret das Bibliothekssystem Sisis SunRise unterstützt –, die über eine selbst entwickelte WeBservices-Schnittstelle OLWS (Open Library WeBservices) angesprochen werden können. Falls ein Benutzer sein selbstregistriertes

Passwort vergessen hat, so kann er es sich über das Modul `OpenBib::MailPassword` per Mail zusenden lassen.

**Hilfe** Die Hilfeseite wird direkt als URL `/suchhilfe.html` angesprochen.

**Sitzung beenden** Beim Ende der Sitzung wird zum Modul `OpenBib::Leave` gesprungen, welches die sessionrelevanten Daten entfernt sowie einen URL in die Schicht 4 zur Lastverteilungsinstanz `OpenBib::LoadBalancer` mit Angabe eines eventuell aktiven Views (Institutssicht) für einen erneuten Einstieg in das Such-Portal bereitstellt.

### 3.3.2. Externe Anbindung an die DigiBib

Neben der Bereitstellung einer Recherche- und Arbeitsoberfläche für den Endbenutzer, besteht auch die Möglichkeit über eine externe Zugriffs-Schnittstelle automatisiert auf den Gesamtkatalog zuzugreifen.

Eine solche Möglichkeit besteht in der Einbindung in das Such-Portal **Digitale Bibliothek NRW** (DigiBib) des Hochschulbibliothekszentrum NRW (hbz). Das Suchinterface besteht aus einem fest definierten Such-URL und einem fest definierten HTML-basierten Antwortformat für die Treffer. Grundsätzlich wird auf eine Rechercheanfrage, z.B. unter Angabe eines Suchbegriffes für den Titel, mit einer Kurztitelliste geantwortet. Es kann bestimmt werden, welcher Teil der Kurztitelliste von den OpenBib-Datenbanken angefordert werden darf (Blätterfunktion mit frei wählbarem Offset und Schrittweite). Darüber hinaus wird neben der Kurztitelliste auch die Anzeige eines Einzeltreffers mit weitergehenden Kategorien angeboten. In der Kurztitelliste sind alle Informationen vorhanden, um den Einzeltreffer aufzurufen.

### 3.3.3. Externe Anbindung an UK-Online

Eine weitere Anbindung besteht in das Universitätsportal UK-Online von Herrn Prof. Lohnstein aus der Philosophischen Fakultät der Universität zu Köln. Hier wird in UK-Online die Möglichkeit angeboten eigenen Bibliographie-Listen zu verwalten. Der Aufbau einer solchen Bibliographie-Liste geschieht über die Recherche in OpenBib und anschließende Umwandlung und Abspeicherung in ein Bibliographie-Listen-Format.

Nach dem derzeitigen Stand sind beide externen Anbindung noch nicht in einer lastverteilten Variante verfügbar. Diese ist jedoch kurz vor der Fertigstellung.

## 3.4. Die Lastverteilungs-Schicht 4

Die Lastverteilungs-Schicht ist der erste Anlaufpunkt für die Benutzung von OpenBib. Diese Schicht wird für das Portal durch das Modul `OpenBib::LoadBalancer` und `OpenBib::ServerLoad` gebildet. Entscheidungsgrundlage für die Verteilung ist die Auslastung der betroffenen Recher, deren generelle Ansprechbarkeit sowie die generelle Funktionsfähigkeit des MySQL-Datenbanksystems.

Durch diese vorgeschaltete Verteilung der Recherche-Sitzungen auf mehrere Rechner ergeben sich mehrere Vorteile:

1. Es wird eine Lastverteilung durchgeführt. Wenn also ein Nutzer einen dieser URL's aufruft, so wird er auf den Rechner weitergeleitet, der am wenigsten belastet ist. Dies ist primärer Zweck dieser Schicht.
2. Ist ein Rechner defekt – im Sinne von 'nicht ansprechbar' bzw. 'DBMS-seitig nicht funktionsfähig' –, dann wird er bei der Verteilung/Weiterleitung nicht mehr berücksichtigt und die Benutzer werden nur noch auf die verbliebenen Rechner verteilt. In diesem Fall ist bei der Weiterleitung mit einer kurzen Wartezeit von maximal 5 Sekunden (Timeout) zu rechnen. Automatisch wird zusätzlich an den Administrator eine Mail generiert, die ihn auf den defekten Rechner hinweist. Ein Sonderfall ist natürlich der 'Verteilungsrechner' selbst, bei dessen Ableben temporär ein anderer Rechner unter seiner Identität einspringen muß. Letzteres kann selbstverständlich nicht automatisch geschehen und muss manuell konfiguriert werden. In diesem Sinn besteht mit dem Lastverteilungsrechner ein Single-Point-Of-Failure, der nur über HA-Mechanismen ausgeräumt werden kann.
3. Werden Wartungsarbeiten auf einem der Rechner ausgeführt, so kann man ihn selbst temporär aus der Verteilung/Weiterleitung herausnehmen. Auf diese Weise ändert sich für die Nutzer nichts, es werden dann die verbliebenen Rechner verwendet.



## 4. Konzepte und Interna von OpenBib

### 4.1. Datenbankstruktur

Die bibliographischen Daten verteilen sich auf eine festgelegte Weise auf verschiedene Tabellen. Es sind dies die Tabellen der **Normdaten**, der **Verknüpfungen**, der **Anfangssuche** sowie der **zwischen gespeichertem Kurztrefe**.

#### 4.1.1. Normdaten

Es gibt 6 Normdatenbereiche. Es sind dies

1. Reine Titeldaten
2. Verfasser bzw. Personen
3. Körperschaften bzw. Urheber
4. Schlagworte
5. Notationen bzw. Systematik
6. Exemplare

Intern ist jedem dieser Bereiche ein alphabetisches sowie ein numerisches Kürzel zugeordnet. Für die Mehrsprachigkeit (Internationalisierung / I18N) wird darüber hinaus vor der vierstelligen Kategoriennummer ein Prefix verwendet. Das alphabetische Kürzel ist Teil des Tabellennamens, das numerische Kürzel wird zur Kennzeichnung des Normdatentyps in der Verknüpfungstabelle **conn** (s.u.) verwendet.

Jedem Normdatentyp sind drei Tabellen zugeordnet, deren Grundname dem alphabetischen Kürzel entspricht.

**Darstellung** In der Tabelle, die dem alphabet. Kürzel entspricht, werden die Kategorien mit ihren Inhalten abgelegt. Die Inhalte sind UTF8-kodiert und werden so 1:1 ausgegeben. Diese Tabellen heißen **tit**, **aut**, **kor**, **notation**, **swt** sowie **mex**.

**Wortanfangssuche** Der ersten, primär für die Darstellung zuständigen Tabelle, ist eine zweite zugeordnet, über die String- oder Wortanfangssuchen realisiert werden können. Der Tabellenname besteht aus dem alphabet. Kürzer, an das **\_string** angehängt ist. Der Inhalt in diesen Tabellen wurde bereits auf seine Grundform reduziert. Welche Kategorien in diese Tabellen

Normdatentyp	alphabet. Kürzel	numer. Kürzel	I18N-Prefix
Titel	tit	1	T
Verfasser	aut	2	P
Körperschaften	kor	3	C
Schlagworte	swt	4	S
Notationen	notation	5	N
Exemplare	mex	6	X

Tabelle 4.1.: Normdatentypen mit Kürzeln und I18N-Prefix

wandern, lässt sich generell sowie für einzelne Katalog konfigurieren. Die Namen dieser Tabellen heissen `tit_string`, `aut_string`, `kor_string`, `notation_string`, `swt_string` sowie `mex_string`.

**Volltextsuche** Der ersten, primär für die Darstellung zuständigen Tabelle, ist eine weitere dritte zugeordnet, über die Volltextsuchen realisiert werden können. Der Tabellename besteht aus dem alphabet. Kürzer, an das `_ft` – wie `fulltext` – angehängt ist. Der Inhalt in diesen Tabellen wurde bereits auf seine Grundform reduziert. Welche Kategorien in diese Tabellen wandern, lässt sich generell sowie für einzelne Katalog konfigurieren. Die Namen dieser Tabellen heissen `tit_ft`, `aut_ft`, `kor_ft`, `notation_ft`, `swt_ft` sowie `mex_ft`.

#### 4.1.2. Verknüpfungen

Um einen vollständigen bibliographischen Titelsatz zu erhalten müssen die jeweiligen Kategorien aus den einzelnen Normdaten-Tabellen über entsprechende Verknüpfungen in der Tabelle `conn` einander zugeordnet werden. Ausgehend von einer ID in einer Normdaten-Tabelle – typischerweise `tit` – wird eine ID aus einer anderen (bei hierarchischen Titelverknüpfungen auch der gleichen) Normdaten-Tabelle zugeordnet.

Über die *Zusammenführung zu einem vollständigen bibliographischen Titelsatz* hinaus wird über diese Tabelle die *Navigation zwischen einzelnen Titeln* über die jeweiligen Normdaten-ID's realisiert.

Im Falle der Zusammenführung zu einem bibliographisch vollständigen Titelsatz kann in `conn` zusätzlich definiert werden, welche Kategorie der Inhalt aus der jeweilig anderen Normdatei im Titelsatz bekommen soll (`category`) und ob eine Zusatzinformation (`supplement`) – z.B. Hrsg. – bezogen auf den speziellen Titel zu dem verknüpften Normdateninhalt (z.B. `aut`) hinzugefügt werden soll.

Im Falle hierarchischer Werke – damit sind sowohl `sourcetype` wie auch `targettype` gleich 1 (Titel), ist sofort ersichtlich, dass

- wenn die lokale Titelsatz-ID im `targettype` auftritt, die Titelsatz-ID im `sourcetype` demnach die ID des **untergeordneten** Titelsatzes ist, während

- wenn die lokale Titelsatz-ID im `sourcetype` auftritt, die Titelsatz-ID im `targettype` demnach die ID des **übergeordneten** Titelsatzes ist.

### 4.1.3. Zwischengespeicherte Kurztreffer

Zur Beschleunigung der Informationsbestimmung für einen Kurztreffer werden in der Tabelle `titlistitem` die jeweiligen Kategorie-Inhalte bereits in der Form eines Caches abgelegt. Damit müssen die Informationen nicht aus allen Normdaten-Tabellen unter Zuhilfenahme der Verknüpfungstabelle bestimmt werden, sondern stehen direkt zur Verfügung.

### 4.1.4. Anfangssuche

Grundsätzlich werden zwei Suchtypen genutzt. Wenn ein Nutzer seine Suchanfrage eingibt, dann wird für diese Anfangssuche zunächst in einem von zwei Recherche-Backends recherchiert. Die Backends lassen sich aufgrund der modularen Struktur von OpenBib sehr einfach erweitern.

Mögliche Recherche-Backends sind

**SQL** In der Tabelle `search` wird entsprechend der *einggegebenen Suchanfrage* recherchiert und eine Liste zutreffender Titel-ID's zurückgeliefert. Mit diesen Titel-ID's wird dann entweder aus den Normdatentabellen oder der Tabelle `titlistitem` der zwischengespeicherten Kurztreffer eine Liste von Kurztreffern für die Anzeige bestimmt.

**Xapian** Alternativ zur Suche via SQL kann auf die Suchmaschinen-Technologie von Xapian zurückgegriffen werden. Es wird dann die eingegebene Suchanfrage in dem entsprechenden Xapian-Index recherchiert. Zurückgeliefert wird eine Liste der direkt im Xapian-Index abgespeicherter Kurztreffereinträge, die für die Anzeige bestimmt sind

Nach diesen zwei Varianten der Anfangssuche wurde dem Nutzer eine Liste von Kurztitelaufnahmen präsentiert, von denen dann direkt in die SQL-Datenbank zu einem vollständigen Treffer in einem spezifischen Katalog gesprungen werden kann.

Ist man erst einmal in einem Treffer eines einzelnen Katalogs, dann wird dort primär mit der Verknüpfungstabelle `conn` über die Normdaten navigiert. Darüber hinaus kann von hier aber auch wieder eine globale Anfangssuche über eines der beiden Recherche-Backends angestoßen werden.

## 4.2. Kategorienformat

### 4.2.1. Allgemeine Informationen

Die bibliographische Datenhaltung in der MySQL-Datenbank ist grundsätzlich unabhängig vom verwendeten Kategorienschema. Damit können dort – ohne Änderung oder Erweiterung der Datenbank – beliebige andere Kategorien, als die aus dem an den MAB2-Standard angelehnte Standardkategorienschema, abgelegt und verarbeitet werden.

Eine Kategorie in der bibliographischen Datenbank zeichnet sich durch folgendes aus:

- Eine Kategorie gehört zu einem Normdatentyp. Mögliche Normdatentypen sind Titel (tit, **T**, 1), Verfasser bzw. Person (aut, **P**, 2), Körperschaft bzw. Urheber (kor, **C**, 3), Schlagworte (swt, **S**, 4), Systematik bzw. Notation (notaton, **N**, 5) und Exemplare (mex, **X**, 6).
- Eine Kategorie besteht aus der numerischen ID bezogen auf einen Normdatentyp, einer Kategorienummer, einem numerischen Indikator<sup>1</sup>.
- Jeder Kategorie ist ein abstrakter Bezeichner zugeordnet, dem wiederum pro Sprache ein Beschreibungstext zugeordnet werden kann. Pro Datenbank können bei Bedarf andere Beschreibungstexte zugeordnet werden

### 4.2.2. Internationalisierung

#### Standardmechanismus

Um Kategoriebezeichnungen für die Ausgabe mehrsprachig aufbereiten zu können, werden aus den Kategoriennummern internationalisierbare Kategoriebezeichner für GNU `gettext` gebildet. Dazu wird der auf vier Stellen vorgegüllten Kategoriennummer der Normdatenprefix vorangestellt (vgl. Abb. 4.1).

Das Bezeichner-Format ist damit:

```
<Normdaten I18N-Prefix><4-stellige Kategorienr.>
```

Der Bezeichner für die Kategorie 331 (HST) in den Titelnormdaten lautet demnach T0331 und der Bezeichner für die Kategorie 1 in den Verfassernormdaten (Ansetzungsform) entsprechend P0001.

Damit ergibt sich folgende Abhängigkeit:

```
Kategorie-Nr. (331) -> Kategorie-Bezeichner (T0331)
                    -> Kategorieausgabebezeichnung ('HST')
```

Damit die einzelnen Kategorie-Bezeichner automatisiert mit dem Programm `xgettext.pl` erkannt und in die I18N-Message-Kataloge übertragen werden können, werden sie in den jeweiligen Templates an den Stellen als Kommentar aufgeführt, an denen definiert wird, welche Kategorien in welcher Reihenfolge angezeigt werden sollen. In diesen Kommentaren sind sie in einem `maketext`-Aufruf gekapselt.

Als Beispiel wird das Template `visible_categories_titset` betrachtet:

---

<sup>1</sup>Mit dem numerischen Indikator können entweder alphabetische Indikatoren gemappt werden (a gleich 1, b gleich 2 usw.), eine Multiplizität der Kategorie ausgedrückt werden (1. Exemplar gleich 1, 2. Exemplar gleich 2) oder auch beides zusammen über Modulos (a des 1. Exemplars gleich 1, b des 1. Exemplars gleich 2, a des 2. Exemplars gleich 3, b des 2. Exemplars gleich 4)

```
[%# Ausgabekategorien %]
[% categories = [
    'T0100', # msg.maketext("T0100") Verfasser
    'T0101', # msg.maketext("T0101") Person
    'T0103', # msg.maketext("T0103") Gefeierte Person
    'T0200', # msg.maketext("T0200") Urheber
    'T0201', # msg.maketext("T0201") Koerperschaft
    'T0089', # msg.maketext("T0089") Bandangabe
    ...

```

Der für die Internationalisierung wesentliche Teil – der benötigt wird – ist

```
# msg.maketext("T1234")
```

### Katalogweise Umbenennung der Kategoriebezeichnungen

Es gibt Situationen in denen es zwingend notwendig wird für eine Kategorie-Nr. in einer ganz speziellen Datenbank eine andere Kategorieausgabebezeichnung zu verwenden – und diese selbstverständlich mehrsprachig.

Ausgehend von einer Kategorie in der Datenbank – z.B. 331 – wird dafür ein *Virtueller Kategorie-Bezeichner* gebildet, der zusätzlich zum bereits bekannten Bezeichner – mit Bindestrich getrennt – den Datenbanknamen angehängt hat. Das Bezeicher-Format ist dann

```
<Normdaten I18N-Prefix><4-stellige Kategorienr.>-<DB-Name>
```

Soll die Kategorieausgabebezeichnung z.B. für die Kategorie 331 in der Datenbank `test` auf `Titel` geändert werden, so wird hierfür der virtuelle Bezeichner `T0331-test` verwendet, der so in die Message-Kataloge mit der Ausgabebezeichnung `Titel` Eingang findet. Dieses Beispiel wäre in der Realität selbstverständlich recht sinnlos.

Vorteil dieses Mechanismus ist, daß unter Beibehaltung des internen – sinnvollerweise einheitlichen – Kategorienschemas für einzelne Kataloge die Kategorieausgabebezeichnung individuell angepasst werden kann.

Damit dieser Mechanismus greifen kann müssen jedoch folgende Vorarbeiten erfolgen:

**Kennzeichnung der Kategorien** Die Kategorien, die für eine Datenbank 'umetikettiert' werden sollen, müssen für die entsprechende Datenbank in dem Modul `OpenBib::Config.pm` unter `categorymapping` definiert werden.

Im `Titel`-Beispiel würde dieser Eintrag wie folgt aussehen:

```
# Katalogabhaengige Kategorie-Mappings

categorymapping => {
```

```
test => {
    'T0331' => 1,
},
},
```

**Anpassung des Templates** Für die Nutzung der neuen virtuellen Kategorie-Bezeichner im Template muß zunächst ein entsprechendes Fragment hinzugefügt werden:

```
# Nachgeschaltete datenbankspezifische Kategorienamen-Mappings sind
# in Config.pm fuer die entsprechende Datenbank definiert
# und werden im Message-Katalog als <Kategorie>-<Datenbankname>
# kodiert.
IF config.categorymapping.$database.$category ;
    thiscategory = "${category}-${normset.database}" ;
ELSE ;
    thiscategory = ${category} ;
END;
```

Darüber hinaus sollten für die automatische Extraktion der I18N-Bezeichner über das Hilfsprogramm `xgettext.pl` in den Kommentaren des Templates `visible_categories_titset` die neuen virtuellen Kategorie-Bezeichner genannt werden:

```
'T0331', # msg.maketext("T0331-test") Titel
```

Auch hier ist der wesentliche Teil des Kommentars:

```
msg.maketext("T0331-test")
```

### 4.2.3. Beispiel

Als Beispiel verwenden wir die Titelaufnahme des Hauptaufnahme des mehrbändigen Werkes *TCP/IP Illustrated* von *Richard W. Stevens*. Diese Titelaufnahme sieht wie folgt aus:

#### Verknüpfungen

Dieser Titel verteilt sich nun mit seinen Daten auf mehrere Normdaten-Tabellen. Es sind dies hier konkret die Titledaten-Tabelle `tit`, die Verfasser-Tabelle `aut`, die Schlagwort-Tabelle `swt` sowie für die Verknüpfungen des Titelsatzes mit den anderen Normdaten und die Abhängigkeiten zwischen Titeln (Unterordnungen) die Verknüpfungstabelle `conn`.

Zunächst werden für diesen Titel die Informationen in der Titelnormdaten-Tabelle dargestellt mit

```
mysql> select * from tit where id=5899;
```

id	5899
Verfasser	Stevens, W. R.
HST	TCP/IP Illustrated
Vorl.Verfasser	W. Richard Stevens
Verlagsort	Reading, Mass. [u.a.]
Verlag	Addison-Wesley
Schlagwort	TCP / IP
Schlagwort	Internet
Schlagwort	Kommunikationsprotokoll
Unterordnungen	2

Tabelle 4.2.: Beispiel-Titel

dargestellt.

```

+-----+-----+-----+-----+
| id   | category | indicator | content          |
+-----+-----+-----+-----+
| 5899 | 1        | 0        | 05899           |
| 5899 | 2        | 0        | 27.05.2003     |
| 5899 | 331     | 1        | TCP/IP Illustrated |
| 5899 | 359     | 1        | W. Richard Stevens |
| 5899 | 410     | 1        | Reading, Mass. [u.a.] |
| 5899 | 412     | 1        | Addison-Wesley   |
+-----+-----+-----+-----+
6 rows in set (0.30 sec)

```

Dieser Titelsatz ist mit den anderen Normdaten verknüpft. Die ID's der verknüpften Sätze erhält man mit

```
mysql> select * from conn where (sourceid=5899 and sourcetype=1)
or (targetid=5899 and targettype=1);
```

zu:

```

+-----+-----+-----+-----+-----+-----+
| category | sourceid | sourcetype | targetid | targettype | supplement |
+-----+-----+-----+-----+-----+-----+
| 100     | 5899    | 1         | 684     | 2         |            |
| 710     | 5899    | 1         | 133     | 4         |            |
| 710     | 5899    | 1         | 604     | 4         |            |
| 710     | 5899    | 1         | 571     | 4         |            |
| 0       | 5900    | 1         | 5899    | 1         |            |

```

```

|      0 | 5901 |      1 | 5899 |      1 |      |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.11 sec)

```

Es ist damit ersichtlich, dass

- der Verfasser des Titels (`category:100, targettype:2`) die ID 684 hat
- die drei Schlagworte des Titels (`category:710`) die ID's 133, 604 sowie 571 haben
- Die Unterordnungen des Titels (`sourceid`) die Titel-ID's 5900 und 5901 haben.

Damit ergibt sich der Verfasser mit

```
mysql> select * from aut where id=684;
```

zu

```

+-----+-----+-----+-----+-----+
| id  | category | indicator | content          |
+-----+-----+-----+-----+-----+
| 684 |      1  |      0  | Stevens, W. R.   |
| 684 |     102 |      1  | Stevens, W. Richard |
+-----+-----+-----+-----+-----+
2 rows in set (0.04 sec)

```

sowie die Schlagworte mit

```
mysql> select * from swt where id in (133,604,571);
```

zu

```

+-----+-----+-----+-----+-----+
| id  | category | indicator | content          |
+-----+-----+-----+-----+-----+
| 133 |      1  |      1  | TCP / IP         |
| 133 |     113 |      1  | Kommunikationsprotokoll |
| 133 |     108 |      1  | 30m              |
| 133 |     102 |      1  | Transmission control protocol / Internet pr... |

| 571 |      1  |      1  | Kommunikationsprotokoll |
| 571 |     108 |      1  | 30                |
| 571 |     102 |      3  | Kommunikationstechnik |
| 571 |     102 |      2  | Datenübertragung   |

```



Aus diversen Erfahrungen mit dieser Art an Systemen kann ich konstatieren, daß der Aufbau, Betrieb und insbesondere die Updates bei einem Versionswechsel in solchen Systemen ein Administrations-Albtraum ist.

Aus diesen Erfahrungen ist ein alternatives Konzept entstanden, bei dem ebenfalls ein Standard-Set definiert wird, dann jedoch durch Klonen *einzelner* abweichender Templates und Stylesheets – und nicht aller – in fest definierte Verzeichnisse im Dateibaum die Individualisierung für das jeweilige neue Portal erfolgt.

*Grundsätzlich gilt: Existiert ein spezifischeres Template oder Stylesheet gleichen Namens, so wird dieses verwendet und nicht das Standard-Template.*

### 4.3.1. Templates

In OpenBib befindet sich der Standard-Set an Templates in dem Verzeichnis, das in dem Modul `OpenBib::Config.pm` unter `tt_include_path` definiert ist. Default ist das Verzeichnis

```
/opt/openbib/templates
```

In diesem Verzeichnis existieren zwei spezielle Unterverzeichnisse, die für die Kaskadierung der Templates wesentlich sind. Sie heißen `views` und `database`.

Eigenständige Portale werden in OpenBib als Views (Sichten) definiert. Jeder View erhält einen Namen, unter dem er angesprochen werden kann. Jedem View zugeordnet sind ferner verschiedene Datenbanken, die beim Aufruf des Views vorausgewählt sind.

Anhand eines Beispiels soll nun demonstriert werden, wie für einen neuen View `rheinlandbib` ein einziges Template – `corporate_banner` angepasst wird. In diesem Template werden Logo-Texte und Graphiken des Views definiert.

Zunächst muß der entsprechende View in der Web-Administration erzeugt werden. Dazu wird ihm dort ein Name – hier `rheinlandbib` –, eine Beschreibung, eine Aktivität (aktiv oder nicht) sowie verschiedene Datenbanken zugeordnet.

Um nun unter Beibehaltung aller anderen Templates lediglich das Template `corporate_banner` für den neuen View zu individualisieren, geht man wie folgt vor:

1. Man erzeugt im Unterverzeichnis `views` von `tt_include_path` ein neues Verzeichnis mit dem Namen des neuen Views – hier `rheinlandbib`.
2. Man kopiert die zu ändernden Templates aus `tt_include_path` in das neue Verzeichnis – hier `corporate_banner`.
3. Man ändert das Template in dem neuen Verzeichnis.

Damit ist der neue View bis auf das Template `corporate_banner` identisch mit der Standard-Darstellung des Portals.

Diese Vorgehensweise hat neben der Einfachheit den entscheidenden Vorteil, daß View-spezifische Template-Änderungen sehr geordnet in den einzelnen View-spezifischen Verzeichnissen abgelebt

sind. Dies schafft Übersicht, Ordnung - und vereinfacht die weitergehende Administration – u.a. bei Updates – erheblich.

Manchmal kann es auch notwendig werden, dass für eine spezielle Datenbank eine vom Standard abweichende Einzeltitel-Darstellung benötigt wird – z.B. wenn Kategorie-Inhalte anders gruppiert werden sollen<sup>2</sup>.

Die Vorgehensweise entspricht weitestgehend den bei den Views, nur daß das datenbank-spezifische Unterverzeichnis nun `database` und nicht `views` heißt, und dort die anzulegenden Verzeichnisse den Namen der Datenbank tragen, für die entsprechende Templates anzupassen waren, also

1. Man erzeugt im Unterverzeichnis `database` von `tt_include_path` ein neues Verzeichnis mit dem Namen der relevanten Datenbank – hier `einbaende`.
2. Man kopiert die zu ändernden Templates aus `tt_include_path` in das neue Verzeichnis – hier u.a. `search_showtitset`.
3. Man ändert das Template in dem neuen Verzeichnis.

## 4.4. Internationalisierung

### 4.4.1. Allgemeine Informationen zu GNU `gettext`

Zur Unterstützung von verschiedenen Sprachen durch das Portal wurde auf GNU `gettext` zurück gegriffen, das eine *best practice* in diesem Bereich darstellt und von unzähligen Projekten und Programmen verwendet wird, wie z.B. den KDE-Projekt mit mehr als 40 unterstützten Sprachen.

GNU `gettext` zeichnet sich im Vergleich zu anderen Mechanismen dadurch u.a. dadurch aus, daß in den Templates die einzelnen zu übersetzenden Textbezeichner nicht numerisch sind, sondern aus gewöhnlichem Text in einer Basis-Sprache – typischerweise Englisch, bei OpenBib bisher Deutsch – bestehen. Damit ergeben sich verschiedene Vorteile:

- Die Templates bleiben lesbar und beherrschbar. Bei numerischen Bezeichnern hingegen sieht man in einem Template gar nicht mehr, wo genau welcher Text steht.
- Die eigentliche Übersetzung lässt sich an entsprechende Mitarbeiter auslagern, die weder um die Programmierung noch die Templates wissen müssen.

Weitere Vorteile von GNU `gettext` sind:

- Die Möglichkeit der Verwendung von Platzhaltern
- Es existiert ein ganzes Toolset, um die Internationalisierung so automatisiert wie möglich durchzuführen. Gerade dies stellt einen hohen Wert dar.

---

<sup>2</sup>An der USB Köln war dies z.B. insbesondere für die Titeldarstellung der Digitalen Einbandsammlung wesentlich, da in dieser speziellen Sicht die Informationen nach Titel und Einband gruppiert werden mussten.

#### 4.4.2. Technische Realisierung

Während GNU `gettext` seine Wurzeln im Wesentlichen in der C- und C++-Welt hat, wird in Perl sehr gerne stattdessen `Maketext` verwendet, welches jedoch zwar zusätzlich die gezielte Bildung von Pluralen mit `quant` unterstützt, jedoch nicht über ein vergleichbares Toolset wie GNU `gettext` verfügt.

OpenBib setzt zur Internationalisierung das Modul `Locale::Maketext::Lexicon` von Audrey Tang ein, das sich auf Perl-Seit wie `Maketext` verhält, im Backend jedoch speziell GNU `gettext` einsetzt. Damit verwendet OpenBib das *Beste aus beiden Welten*.

Damit wird konkret in den Templates die `Maketext`-Syntax verwendet, während die automatische Message-Tag-Extraktion aus den Templates GNU `gettext`-Syntax liefert.

Um in einem Perl-Programm oder einem Template die Text zu Internationalisieren geht man wie folgt vor:

**Perl-Programm** Der Text wird mit einem Methodenaufruf `maketext` des Message-Objektes `$msg` gekapselt. Letzteres wird in jedem `mod_perl`-Handler über `OpenBib::L10N` instanziiert.

Beispiel:

```
$msg->maketext("Ungültige Session")
```

**Template** Der Text wird ebenfalls mit einem entsprechenden Methoden-Aufruf gekapselt – diesmal jedoch in der Template-Toolkit-Syntax mit dem übergebenen Message-Objekt `$msg`.

Beispiel:

```
[% msg.maketext("Suchhistorie") %]
```

Nachdem neue Begriffe so gekennzeichnet wurden oder andere geändert, muß der Message-Katalog aktualisiert werden.

Dazu sind folgende Schritte notwendig:

1. Aktualisierung der Datei `portal/perl/locales/FILES`. Dort sind alle relevanten Dateinamen aufgeführt, die übersetzt werden sollen.
2. Aufruf des mitgelieferten Hilfsprogramms `xgettext.pl`. Es werden damit aus allen in `FILES` genannten Dateien die Message-Tags extrahiert und in einer Datei `message.po` abgelegt.
3. Abgleich von `message.po` mit dem bereits existierenden Message-Katalog – z.B. `de/LC_MESSAGES/openbib.po`:

```
msgmerge -U de/LC_MESSAGES/openbib.po message.po
```

4. Übersetzen der Sprachdatei `openbib.po` mit einem geeigneten Programm. Sehr zu empfehlen ist hier das KDE-Programm `kbabel`.

Nach dem Restart des Apache-Daemons sind die Änderungen aktiv.

## 4.5. Content-Enrichment

### 4.5.1. Hintergrund

In NRW werden derzeit Scan-Projekte durchgeführt, in denen die Inhaltsverzeichnisse von Büchern digitalisiert werden. Ergebnis der Digitalisierung sind einerseits OCR-Text-Daten zur Anreicherung der Suche sowie Abbilder der Inhaltsverzeichnisse in Form von (HTML-,) TIFF- und PDF-Dateien zur Präsentation für den Benutzer.

Geplant ist die Anreicherung der jeweiligen Lokalsystem-Kataloge mit diesen Informationen. Hierbei wird im Wesentlichen ein Paket bestehend aus OCR-Text-Daten sowie Verweis-URL's auf die (HTML-,) TIFF- und PDF-Dateien gebildet und einem einzelnen Katalogsatz entsprechend seines Katalogschlüssels zugeordnet.

Bei der Benutzerrecherche werden dann zusätzlich diese OCR-Daten konsultiert und idealerweise weitere, diesen zugeordnete Katalogsätze gefunden. Beim Aufruf eines solchen Katalogsatzes im WebOPAC des Lokalsystems werden dem Benutzer dann Verweise zu den Abbildern der Inhaltsverzeichnisse geliefert.

### 4.5.2. Konzept von OpenBib

Vor dem Hintergrund solcher Scan-Projekte wurde ein ein allgemeineres, über deren Ziele hinausgehendes Konzept für den Einsatz in OpenBib und dem Kontext eines viele voneinander unabhängige Kataloge umfassenden Portals erstellt. Neben der Nutzbarmachung solcher Digitalisate waren die Ziele u.a.

- Einfachheit (Verstehen, Umsetzen, Warten),
- Kostenneutralität,
- geringer Aufwand bei der Umsetzung,
- insbesondere die Ablage beliebiger Informationen – nicht nur digitalisierter Inhaltsverzeichnisse - zur Kataloganreicherung sowie
- eine *automatische* Nutzung durch alle im Portal zusammengefassten Kataloge.

Entsprechend des letzten Zieles kann eine Bindung der Digitalisat-Informationen an einen einzelnen Kataloges, wie dies typischerweise über eindeutige Katalogschlüssel geschehen ist, nicht zum Erfolg führen. Aus diesem Grund bietet sich als katalogübergreifender Zuordnungsschlüssel zu einem Digitalisat sinnvollerweise nur die ISBN an.

Aufbauend auf dieser Zuordnung können nun alle Anreicherungsinformationen – auch jenseits der digitalisierten Inhaltsverzeichnisse (z.B. speziell Stichwortverzeichnisse, aber auch Rezensionen, Verlagsinformationen usw.) – zentral in einer separaten *Anreicherungsdatenbank* verwaltet werden. Diese Datenbank beinhaltet

- einen Bereich für die Recherche – hier sind z.B. die OCR-Volltexte recherchierbar und liefern eine ISBN als Ergebnis zurück – sowie

- einen Bereich für entsprechende Zusatzinformationen, die in einem Titel mit angezeigt werden sollen – wie z.B. die URL's zu den Abbildern der Inhaltsverzeichnisse – und ebenfalls über die ISBN als Zuordnungskriterium zugreifbar sind.

Damit ist die *automatische* Nutzbarkeit der Informationen nunmehr lediglich auf die Existenz von ISBN's in den einzelnen Katalogen des Portals beschränkt. Alle Kataloge des Portals, die einen Titel mit einer ISBN besitzen, die in der Digitalisatliste (s.o.) enthalten ist, können diesen über Informationen aus dem zugehörigen Inhaltsverzeichnis finden und bei der Ausgabe Verlinkungen zu den entsprechenden Abbildern der Inhaltsverzeichnisse präsentieren.

Wesentliche Vorteile dieser Lösung sind damit auch

- Die Anreicherungsdaten müssen nur an einer Stelle – der Anreicherungsdatenbank – und nicht in allen Katalogen verwaltet werden.
- Alle Titelaufnahmen mit entsprechender ISBN profitieren katalogübergreifend *automatisch* von Such- und Inhaltsanreicherung.
- Es ist problemlos möglich die Lösung um Inhaltsverzeichnisse aus anderen Scan-Projekten innerhalb und außerhalb des Landes NRW für die Kataloge des OpenBib Recherche-Portals nutzbar zu machen.

### 4.5.3. Technische Umsetzung

In OpenBib besteht diese zentrale Anreicherungs-Datenbank – standardmäßig *enrichmnt* genannt – aus folgenden Tabellen:

**search** Diese Tabelle soll für die Anreicherung der Recherche durch entsprechende in ihr abgelegte Informationen genutzt werden. Obwohl der Programmcode zur Nutzung dieser Informationen in OpenBib bereits existiert, wird er solange noch nicht genutzt, bis ausreichende Tests die Skalierbarkeit dieser Art der Suchanreicherung untermauert haben.

**normdata** Diese Tabelle wird für die Anreicherung der Ergebnisse verwendet. Sie hat den Rang und den Aufbau einer katalogübergreifenden Norm-Datei. Damit besitzt sie ebenfalls die Spalten *category*, *indicator* sowie *content*. An die Stelle einer katalogspezifischen Id tritt hier jedoch die katalogübergreifend nutzbare ISBN. Darüber hinaus wird zur Organisation der Anreicherungsinformationen der Ursprung der Informationen numerisch in der Spalte *origin* kodiert. Damit können Informationen verschiedener Herkunft unterschieden und bei Updates entsprechend darauf zugegriffen werden.

Entsprechend dieser Tabellen wird bei OpenBib unterschieden zwischen der *Suchanreicherung* und der *Ergebnisanreicherung* durch externe Daten.

#### Suchanreicherung

Der programmtechnische Ablauf bei der Suchanreicherung ist wie folgt:

- Mit den vom Benutzer eingegebenen Begriffen werden zunächst – **zentral** – in der Tabelle `search` der Anreicherungsdatenbank nach ISBN's als Treffer recherchiert.
- Diese ISBN's werden dann für die herkömmliche Recherche – **pro einzeltem ausgewähltem Katalog** – hinzugenommen und liefern katalogspezifische Titel-ID's.

Derzeitig wird diese Suchanreicherung nicht produktiv verwendet. Ebenso ist noch keine Integration mit der Xapian Suchmaschinen-Technologie erfolgt.

### Ergebnisanreicherung

Der programmtechnische Ablauf bei der Ergebnisanreicherung ist wie folgt:

- Bei der Einzeltrefferanzeige wird dessen ISBN bestimmt
- Mit der ISBN wird in der Tabelle `normdata` der Anreicherungsdatenbank nach Treffern recherchiert.
- Die Treffer aus der Anreicherungsdatenbank haben die Form von zusätzlichen Kategorien. Diese Kategorien werden mit den *herkömmlichen* Kategorien der Katalogaufnahme verschmolzen und bilden so einen angereicherten Titelsatz.

## 4.6. Suchmaschinen-Technologie von Xapian

### 4.6.1. Allgemeine Informationen zu Xapian

OpenBib beherrscht neben dem bereits länger vorhandenen SQL-Such-Backend zusätzlich auch ein Suchmaschinen-Backend, das auf der OpenSource Suchmaschinen-Technologie von Xapian ([www.xapian.org](http://www.xapian.org)) beruht. Xapian wird u.a. auch von der Webpräsenz der Zeitung 'Die Zeit' eingesetzt.

Xapian zeichnet sich durch verschiedene Vorteile aus.

- Xapian ist schnell
- Obwohl Xapian selbst in C++ programmiert ist, kann mit Perl darauf zugegriffen werden. Indexer und Suche sind zusammen weniger als 100 Zeilen Code.
- Xapian bietet Relevance Feedback. Damit können Drill-Downs realisiert werden
- Xapian skaliert vernünftig mit grossen Datenmengen. Bei einer Web-Suchmaschine mit 500 Millionen Webseiten (etwa 1.5 Terrabytes an Datenbank Dateien), bei der Xapian im Einsatz war, brauchte eine Suche trotzdem weniger als 1 Sekunde - natürlich hängt das auch massgeblich von der verwendeten Hardware ab. Eine Suche nach 'Deutschland' im USB-Katalog mit knapp 77000 Treffer brauchte knapp 1 Sekunde...

Neben Xapian ließe sich alternativ auch eine andere Suchmaschinen-Technologie, wie speziell die von Lucene verwenden. Es müsste lediglich der Zugriff von OpenBib durch eine entsprechende Schnittstelle gewährleistet werden. Diese könnte z.B. realisiert werden über

- ein Servlet, das einen Such-URL entgegennimmt und das Ergebnis als XML-Dokument zurückliefert
- einen SOAP-basierten Webservice (z.B. Lucene mit Axis)

Der Ausschlag bei der Entscheidung für Xapian gaben letztlich die bereits vorhandenen Funktionen out-of-the-box, das Perl-API und der fast nicht existente Aufwand bei der Integration in OpenBib.

### 4.6.2. Strukturelle Integration in OpenBib

In OpenBib kann derzeit wahlweise die *Einfache Suche* Suchmaschinen-Technologie verwenden und ist somit *einfach* und treffermengenmässig auch für ungeübte Nutzer – die typischerweise unpräzise suchen und damit sehr große Treffermengen produzieren – geeignet.

Geübte Such-Experten können in der *Komplexen Suche* wie bisher alle bibliothekarisch anspruchsvollen Anfragen über das bereits vorhandene SQL-Such-Backend ausführen – hier wird die Suchmaschinen-Technologie derzeit noch nicht eingesetzt. Bei der Komplexen Suche ist im Gegensatz zur einfachen damit zu rechnen, daß der Nutzer schon sinnvolle Anfragen formuliert – und dafür ist das erprobte SQL Such-Backend weiteraus besser geeignet. Ebenso ist dort ein Drill-Down nicht notwendig.

Diese *Einfache Suche* mit Suchmaschinen-Technologie zeichnet sich durch folgendes aus:

- Es sind – in einem gewissen Umfang – keine komplexen Suchanfragen möglich. Alle eingegebenen Begriffe werden implizit mit UND verknüpft<sup>3</sup>. Damit wird die *Einfache Suche* sehr Google-artig, lässt sich also so bedienen, wie fast jeder Google verwendet.
- Bei den Ergebnissen wird die Zahl aller gefundenen Titel ausgegeben. Nach wie vor ist die Treffermenge begrenzt (derzeit auf 50), allerdings sind diese Treffer – im Gegensatz zur bisherigen Suche über SQL – bereits gewichtet - die *wichtigen* Treffer – in Sinne von: die Suchbegriffe sind häufiger enthalten – *schwimmen* oben, bei den *unwichtigen* kann man verschmerzen, daß sie abgeschnitten und verworfen werden. Ideal ist dies jedoch immer noch nicht – aber auf jeden Fall besser als bei unstrukturierten Suchen über das bisherige SQL-Backend.
- Mit dem Relevance Feedback werden aus der Treffermenge die Wörter bestimmt, die besonders häufig vorkommen – geordnet nach dem Kategoriekreis ihres Vorkommens. Mit diesen wird dem Benutzer eine Drill-Down-Funktion in der bisherigen Treffermenge angeboten. Derzeit werden folgende Drill-Downs angeboten: Verfasser, Wörter aus dem Titel, Wörter

---

<sup>3</sup>Dies kann ohne viel Aufwand auch erweitert werden, da Xapian auch sehr komplexe Abfragen verarbeiten kann – insofern ist das also ein Entscheidung und keine technische Begrenzung

aus den Schlagworten sowie Jahreszahlen. Hat der Benutzer also eine Treffermenge bekommen, so kann er auf einen der Drill-Down-Begriffe klicken und er erhält als neue Trefferliste die Untermenge der bisherigen, die zusätzlich den Drill-Down-Begriff enthält. Damit reduziert sich die Treffermenge und wird übersichtlicher.

- Generell werden die Kurztitel in den Trefferlisten sehr schnell geliefert.

### 4.6.3. Technische Informationen

Die Suchmaschinen-Technologie von Xapian ist an zwei Stellen in OpenBib verankert:

**Indexierung** bei dem automatischen Aufbau eines Katalog sowie

**Recherche** bei der *Virtuellen Recherche* bezogen auf einen Katalog.

#### Indexierung

Für die Indexierung wird über den Xapian-Datenbank-Konnektor `db2xapian.pl`

- eine komplette – bereits gefüllte – Katalog-Datenbank angesprochen,
- die Inhalte nach Kategorien aufgeteilt,
- tokenisiert sowie
- in den Xapian-Index geladen.

Wesentlich sind hierbei zwei Tabellen aus der MySQL-Datenbank. Es sind dies `search` sowie `titlistitem`.

Konkret wird im Xapian-Index entsprechend der Terme aus `search` gesucht und via `Storable` serialisierte Kurztreffer aus dem Data-Teil zurückgeliefert, die effektiv aus `titlistitem` stammen.

Die Indizes werden unter dem in `OpenBib::Config` eingetragenen `xapian_index_base_path` – standardmäßig ist das `/opt/openbib/xapian/index` – in einem Unterverzeichnis abgelegt, welches den Namen der indexierten Datenbank trägt.

Besteht bei der Indexierung bereits ein Index, so wird dieser überschrieben.

Der Datenbank-Konnektor `db2xapian.pl` ist vollständig in die automatische Konvertierung mit `autoconv.pl` integriert. Damit können automatisiert auch die Suchmaschinen-Indexe aufgebaut werden.

#### Recherche

Bei der Recherche kann zwischen den beiden Such-Backends `xapian` und `sql` über den übergebenen Parameter `sb` (Search Backend) ausgewählt werden. Standard ist `sql`.

Zusätzlich kann über den Parameter `drilldown` gesteuert werden, ob beim Xapian-Backend Drill-Downs gewünscht werden.

Die Drill-Downs berücksichtigen aus den `maxhits`-Treffern jeweils die 50 häufigsten Wörter. Entsprechend der Kategorisierung dieser Wörter in die Bereiche

- Verfasser,
- Begriffe aus dem Titel,
- Schlagworte sowie
- Erscheinungsjahre

werden aus den einzelnen Wörtern komplette Verfasser bzw. Schlagworte gebildet, die entsprechend ausgegeben werden.

Grundsätzlich werden Drill-Down-Informationen nur dann ausgegeben, wenn die Treffermenge `maxhits` überschreitet, da erst in diesem Fall eine etwaige Unübersichtlichkeit gegeben ist und Drill-Downs überhaupt Sinn machen.

### 4.7. Konfiguration, Sessions und Benutzer

Für die Konfiguration des Portals, seiner Datenbanken und Views, für das Session-Management sowie für authentifizierte Portal-Nutzer werden drei entsprechende Objekte `OpenBib::Config`, `OpenBib::Session` und `OpenBib::User` verwendet.

#### 4.7.1. Konfiguration

Über das Objekt `OpenBib::Config` kann überall im Portal auf die Konfiguration des Portals als solches sowie der im Portal über die Administration eingerichteten Datenbank-View-Informationen zugegriffen werden.

Dazu wird üblicherweise in einem `mod_perl`-Handler das entsprechende Objekt instanziiert:

```
my $config = new OpenBib::Config();
```

Danach kann auf die Variablen

```
my $dbh = DBI->connect("DBI:$config->{dbmodule}:dbname=$database;  
                      host=$config->{dbhost};port=$config->{dbport}",  
                      $config->{dbuser}, $config->{dbpasswd});
```

oder Methoden

```
my @view_databases = $config->get_dbs_of_view($view);  
my @active_databases = $config->get_active_databases();
```

des Objektes zugegriffen werden.

Intern werden die Informationen für die Variablen der Portalkonfiguration in einem Hash abgelegt, auf den das Objekt direkt zugreifen kann. Demgegenüber werden die Informationen über die Kataloge und Sichten in einer eigenständigen Konfigurations-Datenbank<sup>4</sup> abgelegt, auf die über entsprechende Methoden des Objektes zugegriffen werden kann.

Ein instanziiertes Objekt besitzt immer einen Datenbank-Handle, über den die Kommunikation mit der Datenbank abläuft.

### 4.7.2. Session-Management

Beim Aufruf des Portals wird eine eindeutige Session-ID generiert, mit deren die verbindungslose (stateless) Natur des http-Protokoll umgangen wird. Mit dieser ID sind alle Anfragen einer Session einander zugeordnet.

Alle Session-Informationen werden mit der Session-ID als zentralem Zugriffsschlüssel in der Session-Datenbank gespeichert.

Über die Verwendung der Methoden des Objektes `OpenBib::Session` kann strukturiert auf alle relevanten Informationen einer spezifischen Session zugegriffen werden.

Ein Session-Objekt läßt sich auf zwei verschiedene Arten instanziiieren:

**Erzeugung einer neuen Session** Beim ersten Aufruf des Portals über den `mod_perl`-Handler `OpenBib::StartOpac` wird zwingend eine neue Session mit zugehöriger ID erzeugt. Dazu wird der Konstruktor **ohne** Argument aufgerufen:

```
my $session = new OpenBib::Session();
```

**Zugriff auf eine existierende Session** Existiert bereits eine Session, das ist der Fall bei dem Aufruf aller anderen Handler, so kann über die dann schon vorhandene Session-ID auf die Daten der Session zugegriffen werden. Dazu wird der Konstruktor **mit** der jeweiligen Session-ID als Argument aufgerufen:

```
my $query = Apache::Request->new($r);  
  
my $status = $query->parse;  
  
if ($status) {  
    $logger->error("Cannot parse Arguments - ".$query->notes("error-notes"));  
}
```

---

<sup>4</sup>Die Definition befindet sich in `.../db/mysql/config.mysql`.

```
my $session = new OpenBib::Session({
    sessionID => $query->param('sessionID'),
});
```

Danach kann über verschiedene Methoden auf die entsprechenden Session-Daten abrufend und setzend zugegriffen werden.

```
my $view      = $session->get_viewname();
my $prevprofile = $session->get_profile();
$session->set_mask($setmask);
$session->updatelastresultset(\@resultset);
```

Auf die sehr wichtige Session-ID kann über die Variable ID zugegriffen werden:

```
...
$ldnresult->execute($session->{ID}) or $logger->error($DBI::errstr);
...
```

### 4.7.3. Benutzer

Wenn sich ein Benutzer am Portal authentifiziert, dann werden ihm weitere Funktionen angeboten:

- Seine Merkliste wird dauerhaft gespeichert.
- Er kann sich verschiedene Suchprofile anlegen, verwalten und dauerhaft speichern.
- Er kann für sich dauerhaft einstellen, welche Recherchekategorien ihm in der Recherchemaske angezeigt werden.
- Bei der Authentifizierung an einem Sisis-System werden ihm seine Stammdaten angezeigt – ebenso eine mögliche Sperre samt Begründung.

Für die Authentifizierung und diese personalisierten Funktionen wird intern das Objekt `OpenBib::User` verwendet.

Dieses Objekt wird ohne Argumente im Konstruktor instanziiert:

```
my $user = new OpenBib::User();
```

Danach können über dieses Objekt mit seinen Methoden verschiedene Informationen abgerufen werden:

```
# Authorisierte Session?
my $userid = $user->get_userid_of_session($session->{ID});

foreach my $profile_ref ($user->get_all_profiles()){
    ...
}

my $result = $user->authenticate_self_user({
    username => $loginname,
    pin      => $password,
});
```



# A. Xapian Perl-API

An dieser Stelle werden weitergehende Informationen zum Perl-API für den Zugriff auf die Xapian Suchmaschinen-Technologie gegeben.

## A.1. Genereller Aufbau eines Index

Ein Xapian-Index wird über ein entsprechend instanziiertes Datenbank-Objekt angesprochen. Je nachdem, ob der Index neu erzeugt, geändert oder lediglich für eine Recherche lesend geöffnet werden soll, werden verschiedene Objekte verwendet.

Grundsätzliche Verwaltungseinheit innerhalb eines Xapian-Index ist ein *Dokument*, welches sich wiederum aus Untereinheiten wie *Termen*, *Postings* oder *Data* zusammensetzt.

Xapian bietet in einem Dokument *keine unterschiedlichen Suchfelder*, denen verschiedene Inhalte wie Verfasser, Titel usw. zugeordnet werden können.

Xapian kennt primär nur Text-Token, die indexiert werden sollen.

Dies stellt keine Einschränkung dar, da über die Token-Bildung wiederum die Zuordnung zu inhaltlichen Bereichen durch geeignetes **Prefixen** gesteuert werden kann. Alle Tokens aus dem Bereich Verfasser können so z.B. den Prefix X1, alle aus dem Bereich Titel den Prefix X2 usw. bekommen. Bei der Suche müssen die Such-Tokens entsprechend geprefixed werden. Xapian stellt für solche Prefixe systemseitig *Aliase* bereit, mit denen automatisch in Suchanfragen die Recherche in entsprechenden Inhaltsbereichen gesteuert werden kann.

## A.2. Erzeugung oder Änderung eines Index

### A.2.1. Allgemeiner Ablauf einer Indexierung

Für die Änderung oder die Erzeugung eines neuen Index – speziell also, wenn indexiert werden soll – wird das Datenbank-Objekt

```
Search::Xapian::WritableDatabase
```

verwendet. Diesem Objekt wird bei der Erzeugung der Pfad des Index-Verzeichnisses übergeben sowie Flags, die den Zugriff genauer definieren.

Typische Flags sind

- `Search::Xapian::DB_CREATE_OR_OVERWRITE` für das Anlegen eines **neuen** Index und
- `Search::Xapian::DB_CREATE_OR_OPEN` für die Änderung eines bestehenden Index.

Beispiel:

```
my $db = new Search::Xapian::WritableDatabase ( $dbpath,  
        Search::Xapian::DB_CREATE_OR_OVERWRITE )  
        || die "Couldn't open/create Xapian DB $!\n";
```

In diesen Index können nun Dokumente abgelegt werden. Dazu wird ein Dokument-Objekt über die Klasse

`Search::Xapian::Document`

erzeugt, in das entsprechende Daten Eingang finden können.

Beispiel:

```
my $doc = new Search::Xapian::Document();
```

Xapian unterscheidet verschiedene Arten von Daten, die einem Dokument zugeordnet werden können:

**term** Terme sind einzelne Token, bei denen **keine** Informationen über deren Reihenfolge vorliegt.

Terme werden über die Methode `add_term` dem Dokument hinzugefügt, deren Argument der zu indexierende Term ist.

Beispiel:

```
$doc->add_term("xapian");
```

**postings** Postings sind einzelne Token, bei denen – im Gegensatz zu Termen – eine Reihenfolge definiert werden kann. Damit sind über Postings Recherchen nach Phrasen oder genereller Nachbarschaft einzelner Worte bzw. Tokens möglich.

Postings werden über die Methode `add_postings` dem Dokument hinzugefügt, deren erstes Argument das zu indexierende Token und deren zweites Argument die numerische Stelle des Token in einem Zusammenhang beschreibt.

Beispiel:

```
$doc->add_posting("die",1);  
$doc->add_posting("xapian",2);  
$doc->add_posting("suchmaschine",3);
```

**data** Data stellt einen selbst definierbaren *Ablagekorb* für eigene Daten innerhalb eines Dokuments dar. Bei der Recherche kann in den Ergebnissen direkt darauf zugegriffen werden. Damit bietet es sich an, hier z.B. Informationen abzulegen, die mit keinem oder geringem Aufwand direkt für die Trefferanzeige genutzt werden können, wie z.B. mit **Storable** serialisierte und **unpack** hexadezimal kodierte komplexe Datenstrukturen (Hashes von Hashes von Arrays).

Data wird dem Dokument über die Methode `set_data` hinzugefügt, deren Argument das hinzuzufügende Datenpaket enthält.

Beispiel:

```
$doc->set_data(encode_utf8($listitem));
```

Schließlich wird das Dokument dem Index mit der Methode `add_document` hinzugefügt. Diese Methode liefert eine Xapian-interne *Document-Id* zurück.

Beispiel:

```
my $docid=$db->add_document($doc);
```

### A.2.2. Weitere Hinweise

Bei der Indexierung haben sich einige Vorgehensweisen als sinnvoll erwiesen.

#### Verwendung eines Tokenizers

Um einen Satz in zu indexierende Bestandteile aufzubrechen bietet sich die Verwendung eines Tokenizers an. Ein Beispiel für einen solchen Tokenizer stellt das CPAN-Modul `String::Tokenizer` dar, das sehr einfach zu handhaben ist.

Beispiel:

```
my $tokenizer = String::Tokenizer->new();

$tokenizer->tokenize("die xapian suchmaschine");

my $k = 0;
my $i = $tokenizer->iterator();

while ($i->hasNextToken()) {
    my $next = $i->nextToken();
    next if (!$next);
    next if (length($next) < 3);

    # Token als Posting einfüegen
    $doc->add_posting($next,$k);
    $k++;
}
```

## Normalisierung der Tokens

Xapian indexiert unter Beachtung der Groß-/Kleinschreibung. Aus diesem Grund – und damit z.B. Sonderzeichen geeignet gefunden werden können – empfiehlt sich

- eine Normalisierung der Tokens auf Kleinschreibung sowie
- eine Normalisierung der Tokens bzgl. Sonderzeichen (ue anstelle ü usw.).

Diese Normalisierung muß sowohl bei der Indexierung wie auch bei der späteren Suchanfrage geschehen.

## Inhaltliche Zuordnung durch Prefixen

Wie bereits dargestellt wird in Xapian eine inhaltliche Zuordnung nicht durch separate Suchfelder – wie z.B. in einer SQL-Datenbank – realisiert, sondern durch Prefixen der Tokens.

Reichen 10 Felder aus, so bieten sich die Prefixe X0 bis X9 an, bis 100 Felder ansonsten X00 bis X99.

Bei der Wahl der Prefixe sollte darauf geachtet werden, dass

- diese nicht irrtümlich Teil eines regulären Wortes sind. Dies kann durch großgeschriebene Prefixe bei ansonsten kleingeschriebenen angehängten Token jedoch generell vermieden werden.
- diese nicht übermäßig lang sind, da sonst der Index unnötigerweise vergrößert wird.

Damit von allen Begriffen auch ein allumfassender Suchraum der Token gebildet werden kann – auch Freie Suche genannt – sollte jedes Token auch unprefixiert Eingang in den Index finden.

## A.3. Recherche in einem Index

Für die Recherche in einem Index wird das Datenbank-Objekt

```
Search::Xapian::Database
```

verwendet. Diesem Objekt wird bei der Erzeugung lediglich der Pfad des Index-Verzeichnisses übergeben.

Beispiel:

```
my $db = new Search::Xapian::Database (
    $config{xapian_index_base_path}."/".$database)
    || $logger->fatal("Couldn't open/create Xapian DB $!\n");
```

Für die Formulierung der Anfrage bietet sich die Verwendung des QueryParsers an. Dazu wird ein QueryParser-Objekt

```
Search::Xapian::QueryParser
```

erzeugt mit

```
my $qp = new Search::Xapian::QueryParser()
        || $logger->fatal("Couldn't create QueryParser $!\n");
```

Das Verhalten dieses Objektes kann durch verschiedene Methoden beeinflusst werden.

**Definition von Prefix-Aliases** Prefix-Aliases werden mit der Methode `add_prefix` definiert, deren erstes Argument der Alias und deren zweites Argument der Prefix ist.

Beispiel:

```
$qp->add_prefix('inauth' , 'X1');
$qp->add_prefix('intitle' , 'X2');
$qp->add_prefix('incorp' , 'X3');
$qp->add_prefix('insubj' , 'X4');
$qp->add_prefix('insys' , 'X5');
$qp->add_prefix('inyear' , 'X7');
$qp->add_prefix('inisbn' , 'X8');
$qp->add_prefix('inissn' , 'X9');
```

Damit braucht in der Suchanfrage dann für die Recherche im Titelfeld nach *perl* lediglich

```
intitle:perl
```

einggegeben werden.

**Definition der Standardverknüpfung** Die Standardverknüpfung zwischen den eingegebenen Suchbegriffen kann über die Methode `set_default_op` definiert werden.

Beispiel:

```
$qp->set_default_op(Search::Xapian::OP_AND);
```

Ausgehend von diesem QueryParser-Objekt kann nun eine Suchanfrage (Enquiry) an die Datenbank gestellt werden. Dazu werden zunächst mit der Methode `parse_query` des QueryParsers die Suchbegriffe geparkt, um dann über die Methode `enquire` der Datenbank die eigentliche Anfrage zu stellen.

Beispiel:

```
my $enq = $db->enquire($qp->parse_query($querystring));
```

Mit der Methode `get_description` der Methode `get_query` kann die genaue Suchanfrage aus `$enq` bestimmt werden.

Beispiel:

```
my $thisquery = $enq->get_query()->get_description();
```

Als Ergebnis der Anfrage erhält man einen sog. *MSet* (Match-Set). In diesem sind die Suchergebnisse zusammengefasst.

Mit der Methode `matches` kann dieser *MSet* aus `$enq` gebildet werden. Als Parameter dieser Methode kann die Größe des *MSet* definiert werden.

Beispiel:

```
my @matches = $enq->matches(0,99999);
```

In diesem Beispiel erhält man Maximal 100000 Treffer.

Die wirkliche Trefferzahl liefert die Auswertung von `@matches` in einem skalaren Kontext.

Beispiel:

```
my $fullresultcount = scalar(@matches);
```

Mit einer Iteration über `@matches` kann nun über die Methoden `get_document` sowie `get_docid` auf den entsprechenden Treffer zugegriffen werden.

Beispiel:

```
my $docid = $matches[$i]->get_docid;  
my $document = $matches[$i]->get_document();
```

Aus diesen Dokument lassen sich nun wiederum die dort als Data gespeicherten Informationen mit der Methode `get_data` bestimmen.

Beispiel:

```
my $titlistitem_raw=pack "H*", decode_utf8($document->get_data());
```

Damit sind die wesentlichen Aspekte der reinen Suche umrissen. Darüber hinaus lassen sich Treffermengen weiter verarbeiten für sog. Drill-Downs.

### A.3.1. Drill-Downs - RSets und ESets

Aus den *MSets* können über die *DocId* relevante Titel definiert werden, die dann einen sog. *RSet* (Relevant Set) bilden.

Beispiel:

```
my $rset=Search::Xapian::RSet->new()

...Anfang Schleife über MSet...
  $rset->add_document($matches[$i]->get_docid)
...Ende Schleife über MSet...
```

Mit dem *RSet* werden diejenigen Treffer zusammengefasst, die für eine weitere Auswertung im Rahmen der sog. *ESets* (Enhanced Set) interessant sind.

Ausgehend von einem *RSet*-Objekt kann nun ein *ESet*-Objekt auf Grundlage der Suchanfrage `$enq` mit deren Methode `get_eset` gebildet werden. Als Parameter der Methode wird die Anzahl der relevantesten Token übergeben. Als Ergebnis erhält das *ESet*-Objekt die entsprechenden Token.

Beispiel:

```
my $eterms=$enq->get_eset(50,$rset);
```

Über diese Terme ist nun eine Iteration mit einem entsprechenden Iterator möglich, bei der sowohl die Terme wie auch deren Gewichtung bestimmt werden könne.

Beispiel:

```
my $iter=$eterms->begin();

while ($iter != $eterms->end()){
  my $term    = $iter->get_termname();
  my $weight  = $iter->get_weight();
  ...
}
```

Die Terme/Token können nun in Zusammenhang zu verschiedenen Kategorien der Datenbank gesetzt werden, um *sprechendere* Begriffsketten, wie z.B. einen vollständigen Verfasser zu erhalten. Dies wird von *Xapian* jedoch nicht abgedeckt und muß selbst realisiert werden.



## B. Tabellen in config.mysql

In diesem Anhang werden die einzelnen Tabellen in der Konfigurations-Datenbank beschrieben.

**dbinfo** In dieser Tabelle werden Informationen zu den einzelnen Katalogdatenbanken abgespeichert. Dazu gehört die Fakultät (**faculty**, derzeit fest in den Modulen einprogrammiert), eine Beschreibung (**description**), das Katalogisierungssystem (**system** anhand Kürzel), der Datenbankname (**dbname**), das Bibliothekssigel (**sigel**) und schließlich die wesentliche Information, ob die Datenbank im Portal aktiv ist oder nicht (**active**). Über die letzte Einstellung lassen sich im laufenden Betrieb Datenbanken ein- und ausblenden.

**dboptions** Zu jeder Datenbank werden hier Informationen abgelegt, wo die zum Aufbau der Datenbank notwendigen Daten liegen und alle zum Heranschaffen der Daten benötigten Informationen wie Rechnername (**host**), Pfade, Dateinamen, Zugriffsprotokoll (**local**, **http**, **ftp**) und Authentifizierungsinformationen wie Benutzernamen oder Passworte. Derzeit sind diese Informationen schon im Wesentlichen vorhanden, sie werden durch die automatischen Konvertierungsskripte jedoch noch nicht verwendet.

**titcount** In dieser Tabelle wird die Anzahl der Titel pro Datenbank festgehalten. Diese Information wird über das Modul `OpenBib::SearchFrame` für den Benutzer ausgegeben. Das Setzen dieser Werte geschieht über die automatischen Konvertierungsskripte.

**rssfeeds** In dieser Tabelle werden alle verfügbaren Typen an RSS-Feeds zu den einzelnen Datenbanken aus **dbinfo** definiert. Die RSS-Feeds lassen sich hier nach deren Konfiguration generell aktivieren sowie deaktivieren.

**rsscache** Um die Antwortzeiten bei der Generierung eines RSS-Feeds zu minimieren, wird ein RSS-Cache mit einer Aktivitätsdauer von 12 Stunden verwendet.

**viewinfo** In dieser Tabelle werden zu einer Recherche-Sicht (**view**) neben dem den View definierenden Viewnamen (**viewname**) auch eine Beschreibung **description** abgelegt. Die Information, ob der View aktiv ist oder nicht (**active**) wird derzeit nicht genutzt.

**viewdbs** In dieser Tabelle werden zu dem Viewnamen die Datenbanken festgelegt, die mit der konkreten Recherche-Sicht assoziiert sind. Hier können auch mehrere Datenbanken eingetragen werden.

**viewrssfeeds** In dieser Tabellen werden zu einem View die RSS-Feeds konfiguriert, die in diesem View über den Menu-Punkt RSS angezeigt werden sollen



## C. Tabellen in session.mysql

In diesem Anhang werden die einzelnen Tabellen in der Session-Datenbank beschrieben.

**session** Dies ist die primäre Tabelle in der die `sessionid` für eine Nutzersitzung abgespeichert wird. Zusätzlich werden folgende Informationen gespeichert:

**createtime** Anfangszeit der Session.

**lastresultset** Geordnete Liste der Katalog-ID's zur letzten Rechercheanfrage. Mit dieser Liste wird die katalogübergreifende Titelnavigation in der Einzeltrefferanzeige realisiert.

**queryoptions** Allgemeine Suchoptionen. Durch die zentrale Speicherung der Suchoptionen brauchen diese nicht bei den internen URL-Verweisen verwendet werden.

die Anfangszeit der Session `createtime` gespeichert.

**treffer** Diese Tabelle realisiert sessionabhängig die **Merkliste**. Es werden zu jeder Session der Datenbankname sowie die Identifikations-Nr des gemerkten Titels abgespeichert. Anhand dieser Informationen wird die Merkliste bei ihrem Aufruf sekundenaktuell mit dem etwaig vorhandenen Ausleihstatus der entsprechenden Titelsätze aufgebaut.

**sessionlog** Diese Tabelle soll im Betrieb mit relevanten Informationen gefüllt werden, die dann statistisch ausgewertet werden können. Derzeit wird sie nicht verwendet.

**queries** In dieser Tabelle werden sessionabhängig die Suchanfragen inkl. gefundener Treffer abgespeichert. Dadurch ist es möglich, sich ältere Suchanfragen wieder in die Recherchemaske eintragen zu lassen.

**dbchoice** In dieser Tabelle wird die vom Benutzer vorgenommene Datenbankauswahl sessionabhängig abgespeichert.

**searchresults** In dieser Tabelle wird sessionabhängig zu jeder Suchanfrage (`queryid` der `queries`-Tabelle) pro Datenbank das Ergebnis gecached. Zusätzlich wird jeweils auch noch die Anzahl der Treffer in der jeweiligen Datenbank abgespeichert. Durch diese Tabelle in Verbindung mit der Tabelle `queries` wird die **Trefferliste** realisiert.

**sessionview** Über diese Tabelle wird die Nutzung eines Views durch eine Session festgehalten. Diese Information wird benötigt, damit im oberen Frame (`OpenBib::HeaderFrame`) wie auch in der Recherchemaske (`OpenBib::SearchFrame`) die View-Informationen zusätzlich angezeigt werden. Derzeit wird in `OpenBib::HeaderFrame` auf ein Bild mit dem Namen des Views

`HTDOCS/images/openbib/views/<viewname>.png`

verwiesen.

**sessionmask** In dieser Tabelle wird festgehalten, welche Recherchemaske – Einfache Recherche oder Komplexe Recherche – der Nutzer zuletzt aktiviert hat. Bei einer Aktivierung des allgemeinen Ober-Menueintrags 'Recherche' wird dann diese konkrete Recherchemaske ausgegeben.

**sessionprofile** ...

## D. Tabellen in `pool.mysql`

Die Tabellen in `pool.mysql` werden nach Norm- bzw. Stamm-Dateien geordnet dargestellt. Es sind dies *Titel*, *Verfasser*, *Körperschaften*, *Notationen*, *Schlagworte* sowie *Exemplardaten*. Darüber hinaus gibt es eine zusätzliche Tabelle `search` in der in Form einer Volltextsuche recherchiert werden kann. Verknüpfungen zwischen den Norm-Dateien werden über die Tabelle `conn` abgebildet.

### D.1. Titel-Normdaten

**tit** In dieser Tabelle befinden sich die für die Anzeige bestimmten Titeldaten. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – sie basiert auf auf MAB2-Kategorienummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**tit\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – sie basiert auf auf MAB2-Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**tit\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – sie basiert auf auf MAB2-Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

### D.2. Verfasser-Normdaten

**aut** In dieser Tabelle befinden sich die für die Anzeige bestimmten Verfasserinformationen für die Kategorien **Verfasser** und **Person**. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**aut\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**aut\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

### D.3. Körperschafts-Normdaten

**kor** In dieser Tabelle befinden sich die für die Anzeige bestimmten Körperschaftsinformationen für die Kategorien **Körperschaft** und **Urheber**. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**kor\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**kor\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategorienummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

### D.4. Schlagwort-Normdaten

**swt** In dieser Tabelle befinden sich die für die Anzeige bestimmten Körperschaftsinformationen für die Kategorie **Schlagwort**. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**swt\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**swt\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

## D.5. Notations-Normdaten

**notation** In dieser Tabelle befinden sich die für die Anzeige bestimmten Notationsinformationen für die Kategorie **Notation**. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**notation\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**notation\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

## D.6. Exemplar-Daten

**mex** In dieser Tabelle befinden sich die für die Anzeige bestimmten Exemplarinformationen aus dem Titeldatenbereich. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**indicator** Indikator – transformiert von alphabetisch nach numerisch

**content** Eigentlicher Kategorieinhalt

**mex\_ft** Kategorien, die über einen Volltextindex recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

**mex\_string** Kategorien, die über eine Wortanfangssuche recherchierbar gemacht werden.

**id** Identifikationsnummer

**category** Kategorienname – eigentlich eine interne Kategoriennummer

**content** Eigentlicher Kategorieinhalt, der suchbar gemacht wird.

## D.7. Verknüpfungen zwischen den Normdaten

**conn** In dieser Tabelle befinden sich Verknüpfungsinformationen für die Normdaten. Grundsätzlich werden in dieser Tabelle folgende Informationen in einem Metadaten-unabhängigen Format abgelegt.

**category** Kategorie(nummer), der die Verknüpfung zugeordnet ist (z.B. Verfasser vs. Person)

**sourceid** Id des zu verknüpfenden Satzes

**sourcetype** Normdatei des zu verknüpfenden Satzes

**targetid** Id des verknüpften Satzes

**targettype** Normdatei des verknüpften Satzes

**supplement** Zusatzinformation, die der Verknüpfung zugeordnet ist (z.B. [Hrsg.]

## D.8. Recherche-Tabelle `search`

Sämtliche vom Benutzer eingegebenen Suchbegriffe werden in der Tabelle `search` recherchiert. Als Ergebnis der Suche werden Identifikationsnummern (IDN's, Katkeys) der gefundenen Titel-Sätze zurückgeliefert. Mit diesen IDN's wird dann in die Norm- bzw. Stammdateien gesprungen, um alle Informationen zu diesen Titel-Sätzen zu gewinnen. Diese Funktionalität wird allein von dem Modul `OpenBib::Search` geleistet.

# E. Objekt-Referenz

## E.1. OpenBib::Config

### E.1.1. Methoden

#### **new**

Konstruktor	<code>new</code>
Argumente	keine
Rückgabe bei Erfolg	Objekt mit Instanzvariablen und Handle zur Config-Datenbank

#### **get\_number\_of\_dbs**

Konstruktor	<code>get_number_of_dbs</code>
Argumente	keine
Rückgabe bei Erfolg	Gesamtzahl aller aktiven Datenbanken

#### **get\_number\_of\_titles**

Konstruktor	<code>get_number_of_titles</code>
Argumente	keine
Rückgabe bei Erfolg	Gesamtzahl der Titel aller aktiven Datenbanken

#### **get\_viewdesc\_from\_viewname**

Konstruktor	<code>get_viewdesc_from_viewname</code>
Argumente	Name des Views
Rückgabe bei Erfolg	Beschreibungstext des Views

### **view\_exists**

Konstruktor	<code>view_exists</code>
Argumente	Name des Views
Rückgabe bei Erfolg	Zahl größer Null, wenn der View existiert

### **get\_valid\_rsscache\_entry**

Konstruktor	<code>get_valid_rsscache_entry</code>
Argumente	Datenbankname, RSS-Typ, RSS-Untertyp, Verfallsdatum
Rückgabe bei Erfolg	Gecacheter RSS-Feed-Inhalt, sonst undef

### **get\_dbs\_of\_view**

Konstruktor	<code>get_dbs_of_view</code>
Argumente	Name des Views
Rückgabe bei Erfolg	Array aller aktiver Datenbanken, die einem View zugeordnet sind

### **get\_primary\_rssfeed\_of\_view**

Konstruktor	<code>get_primary_rssfeed_of_view</code>
Argumente	Name des Views
Rückgabe bei Erfolg	URL des als primär definierten RSS-Feed zu diesem View

### **get\_targetdbinfo**

Konstruktor	<code>get_targetdbinfo</code>
Argumente	keine
Rückgabe bei Erfolg	HoH verschiedener Datenbankinformationen

### **get\_targetcircinfo**

Konstruktor	<code>get_targetcircinfo</code>
Argumente	keine
Rückgabe bei Erfolg	HoH aller OLWS-Zugriffsinformationen

**get\_active\_databases**

Konstruktor	<code>get_active_databases</code>
Argumente	keine
Rückgabe bei Erfolg	Array aller aktiven Datenbanknamen (sortiert nach OrgUnit und Namen)

**get\_active\_databases\_of\_orgunit**

Konstruktor	<code>get_active_databases_of_orgunit</code>
Argumente	Name der Organisationseinheit
Rückgabe bei Erfolg	Sortiertes Array aller der Organisationseinheit zugeordneten aktiven Datenbanknamen

**get\_system\_of\_db**

Konstruktor	<code>get_system_of_db</code>
Argumente	Datenbankname
Rückgabe bei Erfolg	Der Datenbank zugeordneter Systemtyp

**get\_infomatrix\_of\_active\_databases**

Konstruktor	<code>get_infomatrix_of_active_databases</code>
Argumente	keine
Rückgabe bei Erfolg	Gefüllte Matrix von Informationen, die ideal formatiert als Datenbankauswahl genutzt werden kann

**DESTROY**

Konstruktor	<code>DESTROY</code>
Argumente	keine
Rückgabe bei Erfolg	Disconnect von der Config-Datenbank

## E.2. OpenBib::Session

### E.2.1. new

Konstruktor	<code>new</code>
Argumente	keines oder Session-ID
Rückgabe bei Erfolg	Bei Übergabe der Session-ID: Handle auf die Session-Datenbank, Session-ID als Instanz-Variable. Bei Übergabe keiner Session-ID: Initialisierung einer neuen Session-ID in der Session-Datenbank. Dann Rückgabe des Handles auf die Session-Datenbank und der neuen Session-ID als Instanz-Variablen

### E.2.2. \_init\_new\_session

Konstruktor	<code>_init_new_session</code>
Argumente	keine
Rückgabe bei Erfolg	Initialisierung einer neuen Session-ID in der Session-Datenbank
Hinweis	Dies ist eine private Methode, die <b>nur</b> innerhalb der Klasse verwendet werden darf!

### E.2.3. is\_valid

Konstruktor	<code>is_valid</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.4. load\_queryoptions

Konstruktor	<code>load_queryoptions</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.5. dump\_queryoptions

Konstruktor	<code>dump_queryoptions</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.6. merge\_queryoptions**

Konstruktor	<code>merge_queryoptions</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.7. get\_queryoptions**

Konstruktor	<code>get_queryoptions</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.8. get\_viewname**

Konstruktor	<code>get_viewname</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.9. get\_profile**

Konstruktor	<code>get_profile</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.10. set\_profile**

Konstruktor	<code>set_profile</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.11. get\_resultlists\_offsets**

Konstruktor	<code>get_resultlists_offsets</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.12. get\_mask

Konstruktor	get_mask
Argumente	
Rückgabe bei Erfolg	

### E.2.13. set\_mask

Konstruktor	set_mask
Argumente	
Rückgabe bei Erfolg	

### E.2.14. set\_view

Konstruktor	set_view
Argumente	
Rückgabe bei Erfolg	

### E.2.15. set\_dbchoice

Konstruktor	set_dbchoice
Argumente	
Rückgabe bei Erfolg	

### E.2.16. get\_dbchoice

Konstruktor	get_dbchoice
Argumente	
Rückgabe bei Erfolg	

### E.2.17. clear\_dbchoice

Konstruktor	clear_dbchoice
Argumente	
Rückgabe bei Erfolg	

**E.2.18. get\_number\_of\_dbchoice**

Konstruktor	<code>get_number_of_dbchoice</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.19. get\_number\_of\_items\_in\_resultlist**

Konstruktor	<code>get_number_of_items_in_resultlist</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.20. get\_items\_in\_resultlist\_per\_db**

Konstruktor	<code>get_items_in_resultlist_per_db</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.21. get\_all\_items\_in\_resultlist**

Konstruktor	<code>get_all_items_in_resultlist</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.22. get\_max\_queryid**

Konstruktor	<code>get_max_queryid</code>
Argumente	
Rückgabe bei Erfolg	

**E.2.23. get\_searchquery**

Konstruktor	<code>get_searchquery</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.24. `get_number_of_items_in_collection`

Konstruktor	<code>get_number_of_items_in_collection</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.25. `get_items_in_collection`

Konstruktor	<code>get_items_in_collection</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.26. `set_item_in_collection`

Konstruktor	<code>set_item_in_collection</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.27. `clear_item_in_collection`

Konstruktor	<code>clear_item_in_collection</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.28. `updatelastresultset`

Konstruktor	<code>updatelastresultset</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.29. `clear_data`

Konstruktor	<code>clear_data</code>
Argumente	
Rückgabe bei Erfolg	

### E.2.30. DESTROY

Konstruktor	DESTROY
Argumente	
Rückgabe bei Erfolg	

## E.3. OpenBib::User

### E.3.1. new

Konstruktor	new
Argumente	
Rückgabe bei Erfolg	

### E.3.2. get\_cred\_for\_userid

Konstruktor	get_cred_for_userid
Argumente	
Rückgabe bei Erfolg	

### E.3.3. get\_username\_for\_userid

Konstruktor	get_username_for_userid
Argumente	
Rückgabe bei Erfolg	

### E.3.4. get\_userid\_for\_username

Konstruktor	get_userid_for_username
Argumente	
Rückgabe bei Erfolg	

### E.3.5. `get_userid_of_session`

Konstruktor	<code>get_userid_of_session</code>
Argumente	
Rückgabe bei Erfolg	

### E.3.6. `get_targetdb_of_session`

Konstruktor	<code>get_targetdb_of_session</code>
Argumente	
Rückgabe bei Erfolg	

### E.3.7. `get_targettype_of_session`

Konstruktor	<code>get_targettype_of_session</code>
Argumente	
Rückgabe bei Erfolg	

### E.3.8. `get_filename_of_profileid`

Konstruktor	<code>get_filename_of_profileid</code>
Argumente	
Rückgabe bei Erfolg	

### E.3.9. `get_profiledb_of_profileid`

Konstruktor	<code>get_profiledb_of_profileid</code>
Argumente	
Rückgabe bei Erfolg	

### E.3.10. `get_number_of_items_in_collection`

Konstruktor	<code>get_number_of_items_in_collection</code>
Argumente	
Rückgabe bei Erfolg	

**E.3.11. get\_all\_profiles**

Konstruktor	get_all_profiles
Argumente	
Rückgabe bei Erfolg	

**E.3.12. authenticate\_self\_user**

Konstruktor	authenticate_self_user
Argumente	
Rückgabe bei Erfolg	

**E.3.13. DESTROY**

Konstruktor	DESTROY
Argumente	
Rückgabe bei Erfolg	



## **F. Module und Objekte geordnet nach Funktion bzw. Schicht**

### **F.1. Schichtübergreifend**

#### **F.1.1. Handler**

- `OpenBib::Admin`

#### **F.1.2. Objekte**

- `OpenBib::Config`
- `OpenBib::L10N`
- `OpenBib::Session`
- `OpenBib::User`

### **F.2. Schicht 1 - Zugriff auf Katalogdatenbank**

#### **F.2.1. Handler**

- `OpenBib::Search`

### **F.3. Schicht 2 - Zugriff auf versch. Datenbanken der Schicht 1**

#### **F.3.1. Handler**

- `OpenBib::VirtualSearch`

### **F.4. Schicht 3 - Rechercheportal**

#### **F.4.1. Handler**

- `OpenBib::Circulation`

- `OpenBib::Connector::DigiBib`
- `OpenBib::Connector::OLWS`
- `OpenBib::Connector::RSS`
- `OpenBib::DatabaseChoice`
- `OpenBib::DatabaseProfile`
- `OpenBib::DispatchQuery`
- `OpenBib::ExternalJump`
- `OpenBib::FrameSet`
- `OpenBib::HeaderFrame`
- `OpenBib::Leave`
- `OpenBib::Login`
- `OpenBib::MailCollection`
- `OpenBib::MailPassword`
- `OpenBib::ManageCollection`
- `OpenBib::RSSFrame`
- `OpenBib::SearchFrame`
- `OpenBib::SelfReg`
- `OpenBib::StartOpac`
- `OpenBib::Template::Provider`
- `OpenBib::UserPrefs`

## **F.5. Schicht 4 - Lastverteilung**

### **F.5.1. Handler**

- `OpenBib::LoadBalancer`
- `OpenBib::ServerLoad`

# Literaturverzeichnis

- [FliHof05] O. Flimm: *Veränderungen und Erweiterungen im KUG-Projekt*. – in: Kooperative Informationsverarbeitung an der Universität zu Köln - Bericht für das Jahr 2004: Kölner UniversitätsGesamtkatalog (KUG-Projekt), S. 111-115 (2005)
- [FliHof06] O. Flimm, Chr. Hoffrath: *Der Weg des KUG zum umfassenden Recherche-Portal*; erscheint in: Kooperative Informationsverarbeitung an der Universität zu Köln - Bericht für das Jahr 2005 (2006).
- [BoeFli:06] R. Boeff, O. Flimm: *Von der traditionellen zur digitalen Sammlung historischer und künstlerischer Bucheinbände der USB Köln mit einem Einblick in die technische Konzeption der Datenbank*. – in: Bibliothek. Forschung und Praxis. Jahrgang 30 (2006) Nr. 1, S.63