HUMBOLDT-UNIVERSITÄT ZU BERLIN

Filter für Titeldaten

- Dokumentation -

Ein Projekt der Universitätsbibliothek

der Humboldt-Universität zu Berlin

Entwicklung & Konzeption:

Dr. Michael Voß,

Heiko Miersch,

Lorenz Fichte

Stand: 3. September 2015

Version: 1.0.2

Inhaltsverzeichnis

1 Zielstellung

Im Zuge der Umstellung der Sondersammelgebiete in FIDs, sollte die Möglichkeit geschaffen werden, nationale Fachausschnitte für diese zu generieren. Hierfür sollten die Titeldaten der deutschen Bibliotheks-Verbünde nach vorgegebenen Kriterien durchsucht und die Ergebnismenge in einem einheitlichen Format für den Import in Discovery-Systeme bereitgestellt werden. Zusätzlich sollten die Aktualisierungen der Stammdaten, welche über diverse Schnittstellen bereitgestellt werden, bei der Filterung berücksichtigt werden.

Die Definition von Filtern sollte möglichst einfach gestaltet werden und möglichst viele Freiheiten lassen. Hierbei sollte vorrangig auf die klassifikatorische Sacherschließung und die Erfassung gesamter Bibliotheksbestände, bspw. von Spezial-Bibliotheken, Wert gelegt werden. Ebenso sollte eine Bereichssuche über bestimmte Gruppen von Klassifikationen möglich sein.

Um die Nachnutzung der Anwendung zu gewährleisten, sollen die verwendeten Bestandteile unter möglichst vielen Betriebssystemen lauffähig sein.

2 Umsetzung

Um die zuvor genannten Ziele zu erreichen und um das System flexibel zu halten, wurde sich für einen Verbund von mehreren Software-Komponenten entschieden. Diese wären folgende:

- Apache Solr ... Bereitstellung eines Indizes für die Filterung
- SolrMarc ... Indizierung von Marc-Daten nach Apache Solr
- Perl-Skripte ... Steuerung der Indizierungs, Update und Filter-Prozesse
- Kommandozeilen-Skripte ... Administrative Aufgaben und Basis-Konfigurationen

Mit dem aktuellen Stand dieser Suite ist es möglich, sowohl MARC- als auch MARC-XML-Daten in eine Apache Solr Index zu übertragen und diesen für die Filterung der Daten zu benutzen. Als Abfragesprache, in welcher die Filter definiert werden, wird die Solr-Query-Syntax[1] verwendet, welche der Lucene-Syntax[2] ähnlich ist. Diese Filter wie auch die Konfiguration von Pfaden usw. wird über Konfigurationsdateien erledigt.
Die Indizierung der Basis-Daten eines jeden Verbundes bzw. einer jeden Datenquelle erfolgt in einem eigenen Solr-Core. Dies bietet den Vorteil, spezifische Anpassungen beim Indizieren vornehmen zu können oder auch nur eine bestimmte Quelle abzufragen ohne den kompletten Index zu durchsuchen.
Für Updates und Löschungen von Titeldaten können OAI-Schnittstellen abgefragt oder Dateien via SSH bezogen werden. Dies kann automatisch erfolgen und wird über einen Scheduler gesteuert, welcher nur zyklisch getriggert werden muss.
Die verwendeten Bestandteile wurden so gewählt, dass eine Portierung nach MS Windows ebenfalls möglich ist. Allerdings sind bei einer Portierung das Handling von Pfadeangaben und in Skripten genutzte Konsolen-Programme zu prüfen.

[1] siehe `https://cwiki.apache.org/confluence/display/solr/The+Standard+Query+Parser` bzw. `http://wiki.apache.org/solr/SolrQuerySyntax`

[2] `http://lucene.apache.org/core/2_9_4/queryparsersyntax.html`

3 Beschreibung des Systems

3.1 Komponenten

3.1.1 Apache Solr

Solr dient im Projekt als Index auf welchem die Suche vollzogen wird. Da der Index die kompletten MARC-Datensätze enthält, liefert jedes Suchergebnis die entsprechenden Titeldaten mit. Um allerdings die Größe des Indizes nicht zu extrem anwachsen zu lassen, werden nur die Titeldaten gespeichert und die übrigen Suchfelder indiziert. Hierdurch wird der Overhead im Vergleich zu den reinen MARC-Daten minimiert.
Wie bereits erwähnt, erhält jede Datenquelle ihren eigenen Solr-Core, welche aber immer das gleiche Schema verwendet. Lediglich Anpassungen an die Verknüpfungen der MARC-Felder auf Solr-Felder können und sollten hier stattfinden. Dies kann innerhalb der SolrMarc-Konfiguration geschehen.

3.1.2 Solr Marc

Die Indizierung der MARC-Daten erfolgt über die Java-Bibliothek SolrMarc. Diese wurde mit BeanShell-Skripten erweitert um MARC-Feldinhalte bspw. zu verketten. Die eingesetzte Solr-Marc-Version ist eine selbst-kompiliere Version, da die bereitgestellten Distributionen neuere Features noch nicht implementierten.
Da alle versuchten SolrMarc-Versionen nicht mit aktuellen Apache Solr Versionen kompatibel sind, erfolgt die Indizierung nur über die REST-Schnittstelle von Solr. Sollte sich dieses Verhalten von SolrMarc ändern, wäre es auch möglich den Index über die Hadoop-Container von Solr zu schreiben, was das Indizieren stark beschleunigen würde.

BeanShell-Erweiterungen Bei den Skripten handelt es sich um Erweiterungen, die es ermöglichen, die in den Marc-Daten enthaltenen Daten für das Indizieren vorzubereiten. D.h. es werden spezifische Marc-Felder voruntersucht und in spezielle Solr-Felder exportiert. Verwendet werden die Skripte in ./data/core/conf/index.properties.
Der Ordner `index_scripts` in dem sich die Skripte befinden müssen, liegt im SolrMarc Unterverzeichnis `local_build`. Die Skripte sind in Java-Code geschrieben und werden im Folgenden erklärt.

getCategory.bsh Mit dem Skript ist es möglich die entsprechenden Klassifikationen aus den Marc-Datensätzen zu extrahieren. In Marc-Datensätzen befinden sich die Klassifikationen im Feld 084 mit den entsprechenden Unterfeldern. Würde man nur diese Felder in Solr via SolrMarc (ohne getCategory.bsh) indizieren, wäre es nicht mehr möglich, die jeweiligen Klassifikationen zu den Klassifikationsnummern zuzuordnen, da es sehr viele verschiedene Klassifikationen[3] gibt und sich diese teils ähneln (beispielsweise

[3] `http://www.loc.gov/standards/sourcelist/classification.html`

kann es eine Klassifikation BCL 18.00 geben und SDNB 18).
Es beinhaltet vier verschiedene Methoden:

- `getCategoryTypeAndNr(Record)` Exportiert nur Kategorie-Typ zusammen mit Kategorie-Nummer. Diese Variante wird nicht mehr verwendet, da immer nur das erste Vorkommen extrahiert wird.
- `getCertainCategoryNr(Record, String)` Wird hauptsächlich benutzt und exportiert alle Klassifikations-Nummern aus Feld 084 einer bestimmten Klassifikation, die jeweils als Parameter übergeben werden muss (z.B. `getCertainCategoryNr("bcl")`).
- `getAllCategories(Record, String, String)` Nicht mehr in Verwendung. Extrahiert die jeweiligen Klassifikationen aus einem beliebigen Feld (falls Klassifikation nicht in Feld 084 ist).
- `writeFixName(Record, String)` Keine Funktionalität. Gibt nur den Wert von String zurück, um in ein Feld einen fixen String schreiben zu können (z.B. für Feld „verbund“ `writeFixName("b3kat")`)

topicFacet.bsh Dieses Skript ist eine Erweiterung des bereits vorhanden Skripts vom Hebis[4], um Schlagworte zu indizieren.

- `getTopic(Record)` Bereits vorhandene Methode im Ursprungsskript zum Exportieren von Sachschlagworten. Diese Methode wird nicht angewendet, da es zweifelhaft war, dass die korrekten Felder exportiert wurden, sodass eine eigene Methode geschrieben wurde.
- `getKeywords(Record, String)` Dies ist eine Methode ähnlich zu getTopic zum Extrahieren von beliebigen Schlagworten. Welche Art von Schlagwort extrahiert werden soll (beispielsweise Formschlagwort oder Sachschlagwort), wird als Parameter `typeOfKeyword` übergeben. Mögliche Werte für `typeOfKeyword`:
  - 'c' Körperschaft, deren Ansetzungsform mit einem Geographikum beginnt
  - 'f' Formschlagwort
  - 'g' Geographisches/ethnographisches Schlagwort, Sprachbezeichnung
  - 'k' Körperschaft (soweit nicht c)
  - 'p' Personenschlagwort (in der PND durch die Satzart tp ersetzt)
  - 's' Sachschlagwort
  - 't' Titel eines Werkes
  - 'z' Zeitschlagwort

Hinweis: die Bean-Shell Erweiterungen müssen aktuell noch händisch in den SolrMarc Ordner hineinkopiert werden.

[4] `trac.hebis.de/svn/verbuendeindex/trunk/usr/SolrMarc_Generic_Binary_Unix-2.5.1/index_scripts/topicFacet.bsh`

3.1.3 Perl

Die entwickelten Perl-Skripte dienen zur Abwicklung der administrativen Prozesse. Sie lesen die Konfigurationen ein, indizieren einen neuen Solr-Core oder steuern die Update-Prozesse über einen Scheduler. Sie erzeugen ebenso die Anfragen an Solr, welche die gewünschten Filter enthält und legen die Ergebnismenge als Dateien ab. Teilweise werden Prozesse auch über mehrere Threads verteilt um eine optimale Performance zu erreichen.

3.1.4 Kommandozeilen-Skripte

Diese Skripte befinden sich im Verzeichnis `./bin` und sind zu meist Aliase für die Perl-Skripte oder aber sie nehmen administrative Aufgaben ab, wie bspw. das Anlegen eines neuen Solr-Cores inkl. der entsprechenden Verzeichnisse, des Schemas und der Registrierung bei Solr. Die folgenden Skripte sind in der Suite enthalten:

`createCore.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was einen neuen leeren Solr-Core erstellt. Die entsprechenden Dateien werden automatisch generiert und die Konfiguration für die Solr-Cores durch die Anweisungsroutine aktualisiert. Es kann ausgewählt werden ob die Initialdaten als XML- oder Marc-Format vorliegen. Es werden drei verschiedene Update Typen angeboten (siehe Abschnitt 3.1.4 runUpdates). Je nach Auswahl müssen dann anschließend die jeweiligen Pfade und das Update Intervall angegeben werden. Alle vorgenommenen Einstellungen können nachträglich in der Datei `./etc/config.ini` eingesehen und verändert werden.
  Im Wesentlichen werden hier drei Schritte durchgeführt: Erstenes wird die Konfigurationsdatei (`./etc/config.ini`) um die jeweiligen Angaben des Benutzers erweitert. Zweitens wird die entsprechende Ordnerstruktur für den neuen Core hergestellt, sodass in `./data` ein neuer Core mit den jeweiligen Core-spezifischen Konfigurationen (`./data/core/conf`) angelegt wird. Die Konfigurationen sind Standardvorlagen und stammen aus dem Ordner `./etc/templates`. Und drittens wird der Core der laufenden Solr-Instanz mitgeteilt und hinzugefügt.
- Aufruf: `$ ./createCore.sh nameOfNewCore`

`removeCore.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was einen vorhandenen Solr-Core löscht. Es werden alle indizierten Daten, sowie initialen Daten und Updates und sonstigen Konfigurationseinträge für den entsprechenden Core gelöscht. Dies sollte nur ausgeführt werden, wenn es absolut notwendig ist.
- Aufruf: `$ ./removeCore.sh nameOfCoreToRemove`

`prepareSolrMarc.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was ausschließlich zum Herunterladen und konfigurieren von SolrMarc dient und nur einmalig zu Beginn beim Einrichten angewendet werden sollte. Alternativ können die Einstellungen händisch ohne Skript durchgeführt werden (beschrieben in Abschnitt 4).
- Aufruf: `$ ./prepareSolrMarc.sh`

`buildIndex.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was den Solr Index mit Daten des jeweiligen Verbundes füllt und die Basisdaten Indizierung (siehe Abschnitt 3.4.2) durchführt. Nach dem Aufruf wird zunächst eine Liste der existierenden Solr Cores ausgegeben. Zum Auswählen eines bestimmten Cores muss die dazugehörige Zahl eingetippt werden. Nach erneuter Bestätigung wird der gewählte Solr Core mit den Initialdaten indexiert. Die Initialdaten müssen dabei in der Ordnerstruktur in `./data/coreName/initialData` vorliegen. Es ist durch Setzen eines symbolischen Links auf dieses Verzeichnis möglich, die Daten beispielsweise auf Grund ihrer Gesamtgröße an einem anderen Platz zu hinterlegen.
  Das Skript sollte nur einmalig beim initialen Indizieren des jewiligen Cores aufgerufen werden. Dieser Prozess kann je nach Größe der Initialdaten bis zu mehreren Tagen dauern. Wenn weitere Daten hinzukommen, die den bestehenden Index erweitern sollen, wird `runUpdates.sh` gerufen.
- Aufruf: `$ ./buildIndex.sh`

`runUpdates.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was parameterlos aufgerufen wird und nach Updates für die jeweiligen Cores an den verschiedenen Schnittstellen sucht. Bei genügend freiem Speicherplatz wird nach jedem erfolgreichen Update eine Optimierung des Solr-Cores durchgeführt. Das Skript selbst sollte nicht händisch gerufen werden, sondern durch einen Cron-Job verwaltet und täglich automatisch aufgerufen werden. (siehe auch Abschnitt 3.4.3)

`removeUpdates.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was parameterlos aufgerufen wird und alle bereits geladenen Updates der Verbünde im Verzeichnis `./data/core/updates` löscht, da Updates sehr groß werden können und auf Dauer nicht gespeichert werden sollen, da die Änderungen i.d.R. im Solr-Index bereits angepasst sind.
- Aufruf: `$ ./removeUpdates.sh`

`getResults.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was parameterlos die vordefinierten Filter im Verzeichnis `./etc/solrQuery.ini` auf den jeweiligen Solr-Core anwendet und die Ergebnisse in das Verzeichnes speichert, welches in `config.ini` unter der Variable `pathResults` angegeben wurde (default: `./results`).
- Aufruf: `$ ./getResults.sh`

`startSshAgent.sh`

- Es handelt sich um ein ausführbares Bash-Skript, was parameterlos gerufen wird und dazu dient, den ssh-agent zu starten, sodass bei erstmaligem Aufruf nach Neustart, nach dem Private-Key-Passphrase gefragt wird. Es speichert die Umgebungsvariablen, die bei einer Public-Private-Key Authentifizierung notwendig sind in die Datei `./etc/env` ab, sodass für zukünftige Verbindungsanfragen kein Kennwort mehr eingegeben werden muss. Wichtig hierbei ist, dass cron-basierte Prozesse, die eine Public-Private-Key Authentifizierung zum Server benötigen (wie beispielsweise die Updates via SCP) nur automatisiert funktionieren, wenn zuvor dieses Skript ausgeführt wurde.
- Aufruf: `$ ./startSshAgent.sh`

3.2 System-Anforderungen

3.2.1 Hardware (derzeitiger Testbetrieb)

- Intel Core i7
- 64 Bit Architektur
- 4 GiB RAM
- Speicherplatzbedarf >4 bis 5-fache[5] der Initialdaten

3.2.2 Software (derzeitig eingesetzte Versionen)

- Ubuntu 12.04 mit folgenden Zusatzmodulen
  - curl
  - Oracle (Sun) Java
- Apache Solr 4.8.1 [6]
- SolrMarc SVN Revision r17461[7]

[5] ergibt sich aus: Speicherplatzbedarf Initialdaten + 2x Speicherplatzbedarf Index + maximale Ergebnisgröße

[6] `https://archive.apache.org/dist/lucene/solr/4.8.1/solr-4.8.1.tgz`

[7] `https://code.google.com/p/solrmarc/source/checkout`

- Perl 5.14.2 mit folgenden Modulen (nachinstallierbar via CPAN[8])
  - Apache::Solr
  - Archive::Extract
  - Config::INI
  - HTTP::OAI
  - Date::Simple
  - (JSON::Parse)
  - LWP::Simple
  - LWP::UserAgent
  - MARC::Batch
  - MARC::Field
  - MARC::File
  - Net::SCP
  - Net::SSH
  - Sys::Info
  - Text::Unidecode
  - Time::Piece
  - Try::Tiny
  - XML::Simple

3.3 Verzeichnis-Struktur

Das Projekt ist so angelegt, dass ein spezifische Ordnerstruktur eingehalten wurde, um die jeweiligen Module je nach Funktionalität und Zweck voneinander zu trennen.
Ausgehend vom Wurzelverzeichnis gibt es folgende Ordner und wichtige Unterverzeichnisse:

`./bin`

- beinhaltet alle ausführbaren Kommandozeilen-Skripte (siehe Abschnitt 3.1.4)

`./data`

- Das Verzeichnis enthält alle Cores (Verbünde), die in der Solr Instanz angezeigt, verwaltet und geupdatet werden.

  `./data/core/conf/config.properties`

[8] `http://www.cpan.org/`

- enthält Core-spezifische Einstellungen und Pfadangaben, die bei der Erstellung eines neuen Cores (Verbundes) automatisch angelegt werden
- nachträgliche Änderung i.d.R. nicht mehr notwendig

`./data/core/conf/index.properties`

- enthält alle Core-spezifischen Angaben, welche Felder der Daten indiziert werden sollen
- nachträgliche Änderung ist möglich und notwendig, z.B. wenn bibliotheksspezifische Angaben wie (Bibliotheks-)Sigel im Index gelistet werden soll
- alle Änderungen müssen erfolgen bevor `./bin/buildInitialIndex.sh` ausgeführt wird, da sonst die jeweiligen Felder nicht mehr indiziert werden

`./data/core/initialData/`

- enthält die intialen Daten vom jeweiligen Verbund

`./data/core/updates/`

- enthält die Core-spezifischen Updates

`./data/core/updates/lastUpdates.txt`

- enthält abhängig von der Update API entweder den letzten Zeitstempel des korrekten Updates (bei OAI) oder die Dateinamen von den bereits durchgeführten Updates (bei HTTP- und SCP-Updates)
- sollte nicht verändert werden

`./etc`

– enthält wichtige Dateien wie die Konfigurationen der jeweiligen Cores und eine Datei mit der die jeweiligen Filter geschrieben werden können

`./etc/config.ini`

- enthält die allgemeinen und Core-spezifischen Konfigurationen für die internen Skripte

`./etc/solrQuery.ini`

- enthält die jeweiligen Filter in Form von Solr-Queries

`./etc/env`

- enthält die jeweiligen Umgebungsvariablen für die SSH Public-Private-Key Authentifizierung
- sollte die Prozess-ID des Ssh-Agent nicht mehr existieren, muss ein neuer über das Skript `./bin/startSshAgent.sh` gestartet werden

`./lib`

– enthält die jeweiligen Perl Skripte, die über die Bash-Skripte aufgerufen werden

`./log`

– enthält eine globale log.txt in denen alle Änderungen, Updates usw. verzeichnet werden

– enthält außerdem die jeweiligen Core-spezifischen Log-Dokumente, die angelegt werden, wenn die Core Daten initial in den Solr Index initialisiert werden

`./results`

– enthält die Ergebnisse der Suche für die speziellen Filter sortiert nach Name des Filters bzw. Abfragezeitpunkt

3.4 Konfiguration

3.4.1 Basis-Konfiguration

./etc/config.ini Es gibt verschiedene Arten von Konfigurationen. Im wesentlichen gibt es eine Hauptkonfigurationsdatei `./etc/config.ini` in der die jeweiligen Verbünde automatisch ein und ausgetragen werden, wenn ein neuer Core angelegt bzw. gelöscht wird.
Alle Pfade in diesem Dokument können relativ oder absolut angegeben werden. Ist der Pfad absolut, d.h. beginnt dieser mit / wird der Pfad wie angegeben verwendet. Ist ein Pfad relativ, d.h. beginnt der Pfad mit einem Namen, wird der Pfad bis zum Hauptverzeichnis angenommen (Beispiel: das Projekt befindet sich im Ordner `/home/userX/workspace/solr-marc-filter/` und als Ergebnis-Pfad wurde `pathResults = results/` angegeben, dann wird dieser im verwendeten Skript erweitert zu `/home/fichte/workspace/solr-marc-filter/results/`.
Weiterhin ist es wichtig, dass ein Pfad immer mit einem Slash (/) enden muss, da es sonst zu Fehlern durch Pfadkonkatenationen in den jeweiligen Skripten kommen kann.
Folgende Parameter sind in der Konfigurationsdatei enthalten:

(Core-unabhängig:)

Parameter	**Beispiel**	**Erklärung**
`pathToFachkatalogGlobal`	/home/userX/workspace/solr-marc-filter/	Globaler Pfad, in dem sich das Projekt befindet
`pathIndexfile`	/home/userX/Downloads/ solrMarcSource/ script_templates/indexfile	Pfad in dem sich das SolrMarc Skript zum indizieren der Datensätze befindet
`pathLogFile`	log/log.txt	Pfad in der das Log-File abgelegt werden soll (standardmäßig im Unterverzeichnis log)
`pathLogFileAlternative`	log/log_rest.txt	Pfad in das zusätzliche Log-File abgelegt wird, was beim Erstellen von Cores angelegt wird (standardmäßig im Unterverzeichnis log)

`pathToSolrCoresDefault`	/home/userX/Downloads/solr/solr-4.8.1/example/solr/	Pfad, in dem sich die Solr Instanz befindet
`urlSolrDefault`	http://127.0.0.1:8983/solr/	Standard-URL über die die Solr Instanz zu erreichen ist
`resultsMaxRecordsPerFile`	10000	Anzahl der maximal erlaubten Records, die in einem Ergebnis-Datei sein dürfen, wird die Anzahl überschritten, wird eine neue Datei erstellt
`resultsMaxNumber`	1000000	Anzahl der maximal erlaubten Ergebnisse. Bei sehr allgemeinen Filter-Anfragen kann die Ergebnis-Menge sehr groß ausfallen. Dies kann vermieden werden, wenn ein Wert >0 gewählt wird. Bei $\leq$ 0 wird das Ergebnisset nicht beschränkt.
`serverResponseTimeoutSec`	1800	Solr-Anfrage Timeout: wenn der Server in diese Zeit nicht auf die Anfrage reagiert, wird ein Fehler ausgegeben
`pathResults`	results	Pfad, in denen die Ergebnisse der Filter-Anfragen gespeichert werden (kann in einen beliebigen Pfad geändert werden).
`resultType`	mrc	Gibt an in welchem Format die Ergebnisse zurückgebeben werden sollen (`xml` oder `mrc` möglich)

(Core-spezifisch:)

Parameter	**Beispiel**	**Erklärung**
`updateType`	http	einer von drei möglichen Update Varianten (`http, ssh, oai`)
`indexPropertiesFile`	data/swb/conf/index.properties	Pfadangabe zur Konfigurationsdatei; sollte nicht verändert werden
`configPropertiesFile`	data/swb/conf/config.properties	Pfadangabe zur Konfigurationsdatei; sollte nicht verändert werden

`updateIntervalInDays`	6	Anzahl der Tage nachdem wieder nach neuen Updates gesucht werden soll, seit dem letzten Update, `6` würde wöchentliche Updates bedeuten, wenn immer zur gleichen Zeit, bspw. Mitternachts nach Updates gesucht wird (**Hinweis:** da das Herunterladen auch Zeit benötigt, sollte der Parameter immer um eins verringert werden)
`updates`	data/swb/updates/	Pfad in dem die Updates gespeichert werden; sollte nicht verändert werden
`initialDataFormat`	xml	Formatangabe der Initialdaten (`xml` oder `mrc` möglich)
`updateIsRunning`	0	Lock-Flag, damit bei lang andauernden Updates nicht mehrmals dasselbe Update gerufen wird. Ist der Parameter 1, wird gerade nach Updates gesucht bzw. geladen und dadurch dieser Core gesperrt. Bei 0 ist er freigegeben.
`updateFormat`	xml	Format in dem die Updates zur Verfügung gestellt werden (`xml` oder `mrc` möglich)
`urlSolrCore`	http://127.0.0.1:8983/solr/#/swb	URL unter der der spezifische Core zu erreichen ist
`check`	0	Gibt an, ob der jeweilige Core geupdatet werden soll (`1`) oder nicht (`0`).

`lastUpdate`	2015-09-03T12:00:00Z	Zeitpunkt des letzten durchgeführten Updates für den jeweiligen Core (dies hat nichts mit dem Datum bzw. der Aktualität des Updates an sich zu tun). Um das letzte heruntergeladene Update zu identifizieren, muss in die letzte Zeile der Datei `lastUpdates.txt` geschaut werden.
`initial`	data/swb/initialData/	Pfad, in der sich die initialen Daten für die erstmalige Indizierung befinden.

(HTTP-spezifisch:)

`httpUrl`	http://swblod.bsz-bw.de/od/	URL unter der die Updates ohne Zugangsbeschränkung oder weitere URL-Parameter erreicht und heruntergeladen werden können.
`httpUrlDeletions`	http://swblod.bsz-bw.de/loeschungen/	URL unter der die Löschungen ohne Zugangsbeschränkung oder weitere URL-Parameter erreicht und heruntergeladen werden können (gibt es für den jeweiligen Core keine separate URL für Löschungen, wird hier automatisch `httpUrl` verwendet).

(OAI-spezifisch:)

`oaiUrl`	http://bvbr.bib-bvb.de:8991/aleph-cgi/oai/oai_opendata.pl	URL unter der die OAI-Updates ohne Zugangsbeschränkung erreicht werden können (die `oaiUrl` ist im Grunde auch eine http-Webadresse, hinter der sich aber eine OAI-Schnittstelle befindet)

`oaiMaxRecordsPerUpdatefile`	10000	Da die Updates über die OAI-Schnittstelle geladen werden und unterschiedlich viele Updates pro Tag vorliegen können aber nicht alle Updates in eine einzige Datei geschrieben werden sollte, kann mit diesem Parameter angegeben werden, wieviele Records maximal in einer Update-Datei abgelegt werden (Standardwert `10000`).
`oaiMaxDaysPerUpdatefile`	0	Alternativ zum vorherigen Parameter kann auch angegeben werden, dass die angefallenen Updates tageweise zusammengefasst werden. Dann muss aber der Parameter `oaiMaxRecordsPerUpdatefile=0`
`oaiOldestUpdate`	2015-05-01T00:00:00Z	Stellt das Datum des letzten Gesamtabzugs und damit ältestes Update Datum dar. D.h. sollten die veröffentlichten Initialdaten des jeweiligen Verbundes vom 1. Mai sein, gibt es nur Updates nach diesem Datum.

(SCP-spezifisch:)

`sshDataPath`	/home/user/opendata/	Pfad auf dem Server, der zu dem Verzeichnis mit den kompletten Updates führt, die dort auch immer regelmäßig neu eingespielt werden
`sshUser`	user	Benutzername, der Zugriff auf den entfernten Server besitzt
`sshHost`	kobv	IP-Adresse des Servers oder ssh Kürzel (bei Kürzel muss dieser in `~.ssh/config` eingetragen sein)

Konfigurationsbeispiel `config.ini`

(Dies dient nicht als Konfigurationsvorlage sondern nur als Beispiel, da die nicht-globalen Einträge für jeden Core über das Skript zum Erstellen von Cores automatisch angelegt werden.)

```
pathToFachkatalogGlobal = /home/user/solr-marc-filter/
pathIndexfile = /home/user/solrMarcSource/script_templates/indexfile
pathLogFile = log/log.txt
pathLogFileAlternative = log/log_rest.txt
pathToSolrCoresDefault = /home/user/solr/solr-4.8.1/example/solr/
urlSolrDefault = http://127.0.0.1:8983/solr/
resultsMaxRecordsPerFile = 10000
resultsMaxNumber = 0
pathQuery = etc/solrQuery.ini
pathToSolrMarcDefault = /home/user/solrMarcSource/
serverResponseTimeoutSec = 1800
pathResults = results/
resultType = mrc

[swb]
updateType = http
indexPropertiesFile = data/swb/conf/index.properties
configPropertiesFile = data/swb/conf/config.properties
updateIntervalInDays = 14
updates = data/swb/updates/
initialDataFormat = xml
updateIsRunning = 0
updateFormat = xml
urlSolrCore = http://127.0.0.1:8983/solr/#/swb
check = 1
lastUpdate = 2015-09-03T12:00:00Z
httpUrl = http://swblod.bsz-bw.de/od/
httpUrlDeletions = http://swblod.bsz-bw.de/loeschungen/
initial = data/swb/initialData/
```

./data/core/conf/config.properties Es handelt sich hier um eine von SolrMarc benötigte Core-spezifische Konfigurationsdatei, in der u.a. Pfade, die Solr-URL und das Format angegeben sind. Diese Datei wird automatisch beim Anlegen des dazugehörigen Cores erstellt und muss i.d.R. nicht nachträglich verändert werden.

Konfigurationsbeispiel `config.properties`

```
1 solrmarc.solr.war.path = /home/user/solr/solr-4.8.1/example/solr-webapp/
      webapp/WEB-INF/lib
2 solr.path = REMOTE
3 solr.hosturl = http://127.0.0.1:8983/solr/b3kat
4 solr.data.dir = /home/user/solr/solr-4.8.1/example/solr/coreX/data
5 solr.core.name = coreX
6 marc.to_utf_8 = false
7 marc.permissive = true
8 marc.default_encoding = MARC8
9 marc.include_errors = false
```

./data/core/conf/index.properties Es handelt sich hier um eine von SolrMarc benötigte Core-spezifische Konfigurationsdatei, in der aufgelistet wird, welche Felder der Marc-Datensätze indiziert werden sollen. Dabei können die von SolrMarc bereitgestellten Methoden, wie z.B. `getAllAlphaSubfields` und `FullRecordAsMARC` aber auch die eigens angefertigten BeanShell-Erweiterungen verwendet werden. Hierbei ist zu beachten: es können immer nur solche Felderindiziert werden, die bekannte Funktionen verwenden und die auch in dem Solr-Schema definiert sind. Im folgenden Konfigurationsbeispiel müssen also die Skripte `getCategory.bsh` und `topicFacet.bsh` existieren (siehe BeanShell-Erweiterungen).

Konfigurationsbeispiel `index.properties`

```
1 id = 001, first
2 publishDate = 362a
3 keywords = script(topicFacet.bsh), getKeywords("s")
4 BCL_category = script(getCategory.bsh), getCertainCategoryNr("bcl")
5 SDNB_category = script(getCategory.bsh), getCertainCategoryNr("sdnb")
6 SSGN_category = script(getCategory.bsh), getCertainCategoryNr("ssgn")
7 RVK_category = script(getCategory.bsh), getCertainCategoryNr("rvk")
8 verbund = script(getCategory.bsh), writeFixName("GBV")
```

3.4.2 Basisdaten Indizierung

Nachdem zunächst ein neuer Core über das Skript `createCore.sh` angelegt wurde, müssen anschließend die initialen Daten in das Core-spezifische Verzeichnis verschoben werden (siehe Abschnitt 3.3). Ist das erfolgt, sollte noch einmal geprüft werden, ob alle Parameter in den Basis-Konfigurationen gesetzt wurde und der Core korrekt angelegt wurde.
Es können anschließend Daten mit dem Skript `buildIndex.sh` indiziert werden. Als Namenskonvention gilt, wie bei Updates und Löschungen im Index, dass alle initialen Daten als MARC oder MARC-XML Daten (`.mrc` oder `.xml`) vorliegen müssen, um von SolrMarc indiziert werden zu können.

3.4.3 Updates und Löschungen im Index

Updates sind Aktualisierungsdateien, die in regelmäßigen Abständen von den entsprechenden Verbünden bereit gestellt werden. Sie lassen sich unterteilt in:

- Lösch-Updates:
  Die Dateien enthalten in der Regel immer nur die Identifikatoren (IDs) der Datensätze, die gelöscht werden sollen, wobei pro Zeile genau eine ID gelistet wird. Bei OAI Schnittstellen können Lösch-Updates und Daten-Updates in derselben Ergebnismenge liegen.
  Namenskonvention (bei SCP und HTTP):
  `file.del` (z.B. `gbv-catalog-delete-2015-02-01.del`)

- Daten-Updates:
  Die Dateien enthalten die eigentlichen neuen oder überarbeiteten Meta-Informationen.
  Namenskonvention (bei SCP und HTTP):
  `file.xml` oder `file.mrc` (z.B. `gbv-catalog-update-2015-02-01_1of4.mrc`)

Download-Formate Die bezogenen Updates werden im .tar.gz Format erwartet. Sollte andere Formate bzw. Komprimierungen vorliegen, muss entsprechend das Skript `./lib/getUpdates.pl` verändert werden. Die dann entpackten Updates aus den jeweiligen komprimierten Dateien, müssen der obrigen Namenskonvention entsprechen, da SolrMarc sonst nicht korrekt indizieren kann.
Die Updates der jeweiligen Cores werden abhängig von ihrem jeweiligen Quellen unterschiedlich geladen. Aktuell werden hier drei verschiedene Methoden angeboten:

- OAI
  Open Archives Initiative bietet ein Protokoll zum Harvesten von Metadaten. Das Protokoll basiert auf XML und REST und im wesentlichen wird bei dieser Art der Update-Konfiguration eine REST-Anfrage generiert und abgeschickt. Die gelieferten Datensätze, die einem bestimmten Zeitraum zuzuordnen sind werden dann in XML-Format gespeichert.

- SCP

Ist eine simple Konfigurations-Variante, die es automatisiert ermöglicht per Secure Copy die bereitgestellten Updates vom Server auf den lokalen Rechner zu kopieren. Damit dieses Verfahren automatisch funktioniert (z.B. per Cronjob), muss die Verbindung zum entsprechenden Server per Public-Private-Key Authentifizierung vollzogen werden. Dazu muss der entsprechende Private-Key mit Hilfe des Skriptes `startSshAgent.sh` entschlüsselt werden.

- HTTP
  Ist eine simple Konfigurations-Variante, die es automatisiert ermöglicht, angebotene öffentliche Updates, die sich auf dem jeweiligen Verbundsserver unter einer bestimmten URL befinden, herunterzuladen.

3.4.4 Solr-Schema

Das Schema des Indizes bildet die Grundlage für die Filterung der Titeldaten. Definiert ist es in der schema.xml und ist jeweils spezifisch für einen entsprechenden Core.
In ihm werden die folgenden Felder definiert.

Feld-Name	Typen-Klasse	Indexed	Stored	Multi-Valued	Required	Unique	default	Beschreibung	Beispiel
`id`	`solr.StrField`	x	x	-	x	x	-	ID des Marc-Datensatzes; Marc[001] ControlNumber	`id: "515695505"`
`_version_`	`solr.TrieLongField`	x	x	-	-	-	-	Solr internes Feld zur Versionierung	`_version_: 1486110444413255700`
`timestamp`	`solr.TrieDateField`	-	-	-	-	-	NOW	Datum der Indizierung	`timestamp: 2014-11-29T22:36:28.633Z`
`publishDate`	`solr.StrField`	x	-	x	-	-	-	Datum der Veröffentlichung; Marc[362:a]	`publishDate: [1.1815(1817) - 2.1816(1818)]`
`keywords`	`solr.TextField`	x	-	x	-	-	-	Schlagwort; Marc[689:x], wobei x einer von 8 möglichen Subfeldern sein kann (siehe Abschnitt 3.1.2)	

verbund	solr.StrField	x	-	x	- - -	Kennung der jeweiligen Datenquelle, spez. in index.properties	verbund: [GBV]
sigel	solr.Text Field	x	-	x	- x -	Standort-Nachweise der einzelnen Exemplare, als Bibliothekssigel	sigel: ["Ka51"]
marc_display	solr.Text Field	-	x	-	- - -	kompletter Marc-Datensatzes	"marc_display": "03392nam a2200865 cc45000010010000 0000300070001000 5001700017008 ..."
*_category	solr.Text Field	x	-	x	- - -	dynamische Felder für die Indizierung der Kategorien; Zuordnung der Felder in index.properties; Marc[084:2]	"BCL_category": ["52.88"], SSFB_category": ["TECH 425"]

Leider befinden sich in vielen Feldern sehr unterschiedliche Inhalte, welche eine Normalisierung quasi unmöglich machen. Hier sei bspw. das Feld „publishDate“ genannt, bei welchem eine Formatierung als Datumswert wünschenswert wäre, um dieses Feld sinnvoll zu filtern.

3.5 Abfrage

Die Abfragen an das Solr System werden standardmäßig in der Solr-Query-Syntax via curl gestellt. Abfragen können sowohl manuell, als auch über das Skript getResults.sh gesteuert werden, wobei sich letzteres auf Grund von Logging und dem Speichern der Records besser eignet.

3.5.1 ./etc/solrQuery.ini

Dieses Datei enthält die jeweiligen Filter-Anfragen zu den spezifischen Cores. Es können mehrere Anfragen hintereinander gestellt werden aber aktuell keine kombinierte Anfrage über alle Cores. Einige Beispielanfragen werden im folgenden beschrieben.

- Beispiel für eine Abfrage einer konkreten RVK-Klassifikation (RA 1000 = Geographie, Zeitschriften) vom Core GBV:

```
[filter1]
verbund=gbv
query=RVK_category:"ra 1000"
```

- Beispiel für eine Abfrage für einen konkreten RVK-Klassifikations-Bereich (ST 240 – ST 250 = Informatik, Programmiersprachen) im Core GBV:

```
[filter2]
verbund=gbv
query=RVK_category:["st 240" TO "st 250"]
```

- Beispiel für eine logisch verknüpfte Anfrage:

```
[filter3]
verbund=gbv
query=(BCL_category:"17.28" OR SDNB_category:"33") AND  keywords:"landeskunde"
```

Alle Ergebnisse werden standardmäßig unter `results` als `filterName_DatumTUhrzeitZ_query.mrc` mit zusätzlichen Abfrageinformationen in einer gleichnamigen Textdatei abgelegt.

4 Workaround

Im Wesentlichen sind folgende Schritte durchzuführen:

1. Repository clonen

   ```
   $ git clone https://github.com/UB-HU-Berlin/solr-marc-filter.git
   ```

2. die unter Abschnitt 3.2.2 genannten CPAN Module installieren (dabei müssen jeweils auch alle Referenzmodule mit installiert, sowie benötigte Systempakete ggf. via apt-get nachinstalliert werden)

   - prüfen, ob alle Module korrekt installiert wurden:

     ```
     $ perl -e "use Apache::Solr; use Archive::Extract; use Config::INI; use Date::Simple,
     use HTTP::OAI; use LWP::Simple; use LWP::UserAgent; use MARC::Batch; use MARC::Field;
     use MARC::File; use Net::SCP; use Net::SSH; use Sys::Info; use Text::Unidecode;
     use Time::Piece; use Try::Tiny; use XML::Simple;"
     ```

   - wird kein Fehler geworfen wie `Can't locate Module in @INC`, sind die Module korrekt installiert

3. Solr herunterladen (Link siehe Abschnitt 3.2.2)

4. zum Herunterladen und konfigurieren von SolrMarc `$ ./prepareSolrMarc.sh` ausführen oder alternativ Schritte manuell durchführen:

   (a) SolrMarc SVN checkout im Überverzeichnis von solr-marc-filter ausführen

   ```
   $ svn checkout http://solrmarc.googlecode.com/svn/trunk/ solrmarc
   ```

   (b) Ins Verzeichnis `solrmarc` wechseln, bauen und Eingabeaufforderungen folgen

   ```
   $ ant init
   ```

   (c) SolrMarc.jar ins übergeordnete Verzeichnis kopieren

   ```
   $ cp local_build/lib/SolrMarc.jar local_build
   ```

   (d) Ersetzung in Datei vornehmen

   ```
   $ sed -i 's/@MEM_ARGS@/-Xmx256m/' local_build/script_templates/indexfile
   ```

   (e) Rechte zum Benutzen der Indizierungs-Skripte setzen

   ```
   $ chmod u+x local_build/script_templates/*
   ```

   (f) Kopieren der Template Bash-Skripte in das SolrMarc Unterverzeichnis

   ```
   $ cp -r ../solr-marc-filter/lib/templates/solrMarc/*.bsh local_build/index_scripts/
   ```

5. `config.ini` erstellen (dazu kann `config.ini.plain` kopiert werden) und genannte Pfad-Variablen anpassen

6. Solr Instanz starten (`$ java -jar -Xmx2048M -Xms512M start.jar`)

7. neuen Solr-Core erstellen via `$ ./createCore.sh` und Eingabeaufforderungen folgen
8. (optional) fehlerhaft oder testweise angelegten Solr-Core löschen via `$ ./removeCore.sh` und Eingabeaufforderungen folgen
9. Initialdaten (Marc- oder Marc-XML-Daten) in das automatisch angelegten Core-Verzeichnis `solr-marc-filter/data/coreX/initialData/` hineinkopieren
10. in `index.properties` festlegen welche Felder für diesen Core indiziert werden sollen und `"CHANGE_THIS_NAME"` in gewünschten Namen des Cores ändern (sinnvollerweise derselbe Name, wie beim Erstellen des Cores zuvor)
11. Daten im Solr-Core indizieren via `$ ./buildIndex.sh`
12. (optional) in Vorbereitung auf Updates ssh-agent starten via `$ ./startSshAgent.sh`
13. (optional) alle alten Updates in Solr-Core einpflegen via `$ ./runUpdates.sh`
14. (optional) Cron-Job für Auto-Updates einrichten, welcher regelmäßig (beispielsweise täglich um Mitternacht) das Skript `runUpdates.sh` startet
    - Crontab öffnen: `$ crontab -e`
    - folgende Zeile anfügen: `@midnight /CHANGE/THIS/PATH/solr-marc-filter/bin/runUpdates.sh`
15. neuen Filter-Regel erstellen in Datei `solrQuery.ini` (dazu kann `solrQuery.ini.plain` kopiert werden)
16. Anfrage(n) an Solr-Core stellen via `$ ./getResults.sh`

5 Einschränkungen

- Indizierung
  - Indizierung (sowohl initial als auch Updates) nur von Metadaten, die im MARC-XML oder MARC-Format vorliegen (momentan keine Unterstützung von RDF-Daten)
  - Indizierung nur per REST Schnittstelle, d.h. die Indizierung kann relativ viel Zeit in Anspruch nehmen, da Daten nicht direkt im Index abgelegt werden
- Testinstanz
  - nur auf Ubuntu getestet
  - möglicherweise Pfadprobleme etc. unter Windows
  - Updates via SCP, HTTP ausführlich getestet
  - Updates via OAI nur im geringen Umfang getestet
  - Updates via OAI nur wenn Schnittstellen-Dateiformat XML
- Abfragen
  - nur auf Basis des Indizierungsschemas
  - keine nachträgliche Änderung des Index-Schemas ohne vollständige Neuindizierung (Hinzufügen von sinnvoller Auswahl an dynamischen Feldern notwendig)
  - Beschränkungen durch Inhalte der Marc-Daten (z.B. kann nicht allgemein nach Erscheinungsjahr gesucht werden, wenn kein einheitliches Format umgesetzt wird)
  - keine Anfragen über alle Cores (z.B. via shards, da dies zu lange Response-Zeiten verursachen)

6 ToDos

- Identifizierung von Standorten innerhalb des GBV
- Erfassung von RDF Quellen
- Kommandozeilen-Lösungen für MS Windows
- Configs für verteilte Solr Instanzen (aktuell immer nur ein Verzeichnis)