Perspektive Bibliothek 4.2 (2015), S. 33-59 doi: 10.11588/pb.2015.2.26271

Entwicklung und Implementierung eines MARC21-MARCXML-Konverters in der Programmiersprache Perl

Wolfgang Boiger

Bibliotheksakademie Bayern

Wolfgang.Boiger@bsb-muenchen.de

1. Einleitung

Die Einführung der elektronischen Datenverarbeitung hat in der Bibliothekswelt viele Vorgänge signifikant vereinfacht und neue Dienstleistungen ermöglicht.[1] Ein Teil dieser Entwicklung ist die kooperative Katalogisierung. Wurden früher Kopien (bzw. Verfilmungen) von Katalogkarten per Post verbreitet, können Bibliotheken ihre Katalogisate heute per Datenleitung ohne Zeitverlust austauschen und so nachnutzen.

1.1 Die Familie der MARC-Formate

Eine zwingende Voraussetzung für einen reibungsfreien Austausch sind einheitliche Datenformate. Die meistverwendeten Formate für den Transport von Katalogdaten entstammen der Familie der MARC-Formate.[2] Deren Geschichte geht bis in die 1960er Jahre zurück, als das ursprüngliche MARC-Format (heute MARC-I genannt) an der Library of Congress (LoC) mit dem Ziel entwickelt wurde, die Verteilung der Katalogisate der LoC an andere Bibliotheken zu erleichern. Allerdings war MARC-I primär auf die Bedürfnisse der LoC zugeschnitten.[3]

[1] Eine Analyse der Entwicklung findet sich bei Schnalke (2014).

[2] MARC ist die Abkürzung von *machine-readable cataloging*.

[3] Cole & Han (2013, Chapter 4, S. 67–69), Das (2009, Introduction, S. 1–5), Mukhopadhyay (2007, S. 5–6).

Die Weiterentwicklung von MARC-I kulminierte im Format MARC 21, dessen Spezifikationen im Jahr 2000 von der LoC publiziert wurden.[4] Ziel der Entwicklung von MARC 21 war es, ein Format für das 21. Jahrhundert zu schaffen, das international und möglichst langfristig einsetzbar ist.[5] MARC 21 hat sich seither in weiten Teilen der Welt als Standard zum Austausch bibliographischer Daten durchgesetzt.[6]

1.2 MARC und Auszeichnungssprachen

Die MARC-Formate wurden ursprünglich auf minimalen Speicherverbrauch und schnelle Verarbeitung durch Bandlaufwerke hin optimiert (vgl. Abschnitt 2., S. 4).[7] Mit dem Wandel der Speichertechnik verloren diese Ziele an Bedeutung. Die LoC begann daher, Erweiterungen von MARC zu entwickeln, welche MARC-Datensätze mittels Auszeichnungssprachen speichern. Im Gegensatz zur traditionellen Art der Datensatzrepräsentation sind Auszeichnungssprachen in ihrer Handhabung flexibler, insbesondere sind die Datensätze ohne zusätzliche Software, gewissermaßen „mit dem bloßen Auge" lesbar (vgl. Abschnitt 3., S. 5). Aus diesen Entwicklungen ist zunächst in den 1990er Jahren MARC SGML[8] hervorgegangen, konnte sich allerdings nicht durchsetzen. Dies lag unter anderem an der hohen Komplexität des Formats und dem damit verbundenen hohen Speicherbedarf.[9]

Mit der Einführung von XML[10] bot sich eine neue Auszeichnungssprache als Träger von MARC-Datensätzen an. Daher entwickelte die LoC 2002[11] MARCXML als Nachfolger von MARC SGML.[12] MARCXML bietet gegenüber der traditionellen Repräsentation Vorteile in den Bereichen Datensatz-Konversion sowie Präsentation und Analyse von Datensätzen.[13] Insbesondere ist es heute das bevorzugte Format zum Austausch von bibliographischen MARC-Datensätzen mit Web-Diensten außerhalb des Bibliothekswesens (z. B. mit Google).[14]

---

[4] Library of Congress (2000).
[5] Mukhopadhyay (2007, S. 5–6).
[6] Cole & Han (2013, Chapter 4, S. 67–69).
[7] Cole & Han (2013, Chapter 4, S. 69).
[8] *Standard Generalized Markup Language* ist eine Metasprache zur Definition von Auszeichnungssprachen.
[9] Cole & Han (2013, Chapter 4, S. 73).
[10] *Extensible Markup Language*, eine Auszeichnungssprache zur Repräsentation hierarchischer Datenstrukturen.
[11] Siehe die Dokumentation unter http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd .
[12] Cole & Han (2013, Chapter 4, S. 74).
[13] Cole & Han (2013, Chapter 4, S. 81).
[14] Cole & Han (2013, Chapter 4, S. 91).

1.3 Der B3Kat im Format MARCXML

Seit Dezember 2011 veröffentlicht der Bibliotheksverbund Bayern (BVB) im Rahmen der bayerischen OpenData-Initiative halbjährlich einen kompletten Abzug des gemeinsamen Katalogs (B3Kat) des BVB und des Kooperativen Bibliotheksverbundes Berlin-Brandenburg (KOBV) auf seiner Webpräsenz im Format MARCXML.[15] In einem weiteren Schritt werden die in MARCXML vorliegenden Daten in das Format RDF/XML[16] überführt und als Linked Open Data publiziert.[17] Die so publizierten Daten dienen unter anderem der Beteiligung von BVB und KOBV am deutschlandweiten Projekt Culturegraph, in dem die Katalogdaten aus den deutschsprachigen Verbünden gebündelt werden.[18]

Das B3Kat-Verbundsystem Aleph 500 kann die Katalogdaten im traditionellen Format MARC 21 exportieren, jedoch nicht in MARCXML.[19] Für die anschließende Konvertierung in MARCXML wird derzeit die nur unter Microsoft Windows lauffähige Software MarcEdit[20] eingesetzt, die eine manuelle Bedienung der graphischen Benutzeroberfläche mit Maus und Tastatur erfordert. Daher ist die Konvertierung mit zeitaufwändiger Handarbeit verbunden. Ziel der vorliegenden Arbeit ist die Konzeptionierung und Implementierung einer befehlszeilenorientierten Software zur Konvertierung von MARC-21- in MARCXML-Datensätze. Die Implementierung soll in der Programmiersprache Perl erfolgen, damit sie möglichst betriebssystemunabhängig einsetzbar ist und durch die Verbundzentrale des BVB gewartet werden kann.

1.4 Notation

Die Bezeichnung „MARC 21“ beschreibt üblicherweise das *bibliographische Datenformat*, unabhängig von der Art der Speicherung. „MARCXML“ beschreibt hingegen eine Art der Speicherung eines MARC-21-Datensatzes, stellt also lediglich eine Art „Container“ für MARC-Datensätze dar.

Um die traditionelle, bandlaufwerksoptimierte Darstellung eines MARC-21-Datensatzes von der XML-basierten Darstellung („MARCXML“) zu unterscheiden, bezieht sich die Bezeichnung „MARC 21“ hier und im Folgenden stets auf die traditionelle Darstellung, während „MARCXML“ die Darstellung im Format MARCXML bezeichnet.

[15] Die Katalogdaten sind abrufbar unter https://www.bib-bvb.de/web/b3kat/open-data .
[16] *Resource Description Framework* ist ein Verfahren zur Formulierung logischer Aussagen über beliebige Entitäten (z. B. Bücher); RDF/XML ist eine Sprache zur Darstellung solcher Aussagen.
[17] Die Daten sind abrufbar unter http://lod.b3kat.de .
[18] Meßmer (2013), weitere Informationen über Culturegraph finden sich bei Böhme (2014), ebenso bei Schäfer & Kett (2013).
[19] Meßmer (2013).
[20] Siehe http://marcedit.reeset.net.

2. Technische Beschreibung von MARC 21

Dieser Abschnitt beschreibt das Datenformat MARC 21, allerdings nur insoweit, als es für die Konversion in MARCXML relevant ist. Anhang A (S. 11) zeigt ein Anschauungsbeispiel eines MARC-21-Datensatzes, ferner denselben Datensatz im MARCXML-Format.

Da MARC ursprünglich für die effiziente Verarbeitung mittels Bandlaufwerken konzipiert wurde, besteht ein einzelner Datensatz aus einer einfachen langen Zeichenkette, die die relevanten Informationen in stark komprimierter Form enthält. Die meisten dieser Zeichen repräsentieren lesbare Glyphen, an einigen Stellen werden jedoch Bytes verwendet, die kein druckendes Zeichen darstellen. Diese werden im folgenden Abschnitt durch ihren Zahlenwert in hexadezimaler Schreibweise beschrieben,[21] so steht beispielsweise „`0x10`“ für das Byte mit dem Wert 16.

Die folgenden Angaben basieren im Wesentlichen auf der Webseite der Library of Congress (2013). Weiterführende Informationen, insbesondere eine Beschreibung der einzelnen Datenfelder finden sich auch bei Das (2009) und Mukhopadhyay (2007).

2.1 Leader

Die ersten 24 Bytes eines MARC-21-Datensatzes bilden den sog. Leader. Dieser enthält keine bibliographischen Informationen, sondern Informationen über den Aufbau des Datensatzes. Darunter fallen beispielsweise die Größe des Datensatzes und Informationen über dessen Natur (z. B. Normdaten). Damit kann ein Computer Datensätze, die für die Anwendung nicht von Interesse sind, schnell überspringen.

Einige Datenfelder sind für die Implementierung des Konverters von speziellem Interesse: Die Größe des Datensatzes (gezählt in Bytes) ist in den ersten fünf Bytes des Datensatzes in normaler dezimaler Schreibweise enthalten (beginnend mit der höchstwertigen Dezimalstelle). Das zehnte Byte gibt Auskunft über die in den Datenfeldern verwendete Zeichenkodierung:[22] Ein „`a`“ steht hierbei für die Kodierung UTF-8.[23] Die Bytes 13–17 geben die Position des ersten Datenfeldes im Datensatz (relativ zum Beginn des Datensatzes) an.

[21] Dies ist im Softwareentwicklerumfeld üblich und im technischen Bibliotheksumfeld ebenfalls keine Seltenheit, siehe z. B. Das (2009, S. 20–22).

[22] Zeichenkodierungen stellen eine Konkordanz zwischen Bytes (Zahlen zwischen 0 und 255, aus denen die Dateien eines Computers bestehen) und Glyphen dar. Ein in einer Datei gespeicherter Text ergibt erst mit einer vereinbarten Zeichenkodierung einen Sinn.

[23] UTF-8 ist eine Kodierung für den Zeichensatz Unicode, die – auch im bibliothekarischen Umfeld – aufgrund ihrer platzsparenden Darstellung des lateinischen Alphabets weit verbreitet ist.

2.2 Directory

Im Anschluss an den Leader folgt das sog. Directory. In diesem sind alle Datenfelder des Datensatzes einzeln aufgelistet, jeweils mit der Feldnummer (drei Bytes), der Länge des Inhalts (vier Bytes), und der Position des Datenfeldes im Datensatz (fünf Bytes, relativ zum Beginn des ersten Datenfeldes, dessen Position im Leader vermerkt ist). Die Feldnummer definiert dabei die eigentliche Bedeutung eines Feldes, beispielsweise enthält das Feld 100 den persönlichen Namen des Autors eines Werks. Die Größe des Directorys hängt von der Zahl der Datenfelder ab. Auf den letzten Eintrag des Directorys folgt ein spezielles Zeichen, der „field terminator" (Byte `0x1e`).[24]

2.3 Datenfelder

Nach dem Directory folgen die einzelnen Datenfelder unmittelbar aneinandergereiht. Während die Datenfelder mit den Nummern 001–009 ihre jeweilige Information ohne weitere – für eine Konversion nach MARCXML relevante – Unterteilung enthalten, sind die Datenfelder ab der Nummer 010 nochmals untergliedert: Die ersten beiden Zeichen des Datenfeldinhalts sind die sog. Indikatoren; ihre Bedeutung hängt von der jeweiligen Datenfeldnummer ab (beispielsweise enthält der erste Indikator des Feldes 130 Informationen über Zeichen, die bei alphabetischer Sortierung der Datensätze zu ignorieren sind). Ab dem dritten Zeichen folgen paarweise sog. Subfeld-Codes und der Inhalt des jeweiligen Subfeldes. Damit eine datensatzverarbeitende Software Subfeld-Codes von Subfeld-Inhalten trennen kann, steht vor jedem Subfeld-Code ein spezielles Zeichen (`0x1f`), anhand dessen der Beginn des Subfeldes von der Software erkannt wird.[25] Jedes Datenfeld endet mit dem field terminator (`0x1e`, siehe oben). Am Ende des gesamten MARC-21-Datensatzes, im Anschluss an den letzten field terminator, folgt außerdem der „record terminator" (`0x1d`).[26]

Mehrere MARC-21-Datensätze in einer Datei sind Byte für Byte ohne weitere Struktur aneinandergehängt. Durch die Größeninformation sowie den record terminator am Ende kann verarbeitende Software die einzelnen Datensätze erkennen und zuverlässig voneinander trennen.

3. Technische Beschreibung von MARCXML

Aufbauend auf Abschnitt 2. wird im Folgenden das Datenformat MARCXML beschrieben. Dabei wird von einem grundlegenden Verständnis der Auszeichnungssprache XML

[24] Das (2009, Chapter 1, S. 20).
[25] Das (2009, Chapter 1, S. 22).
[26] Das (2009, Chapter 1, S. 20).

ausgegangen.[27]

Inhaltlich unterscheidet sich MARCXML nicht von MARC 21; ein MARCXML-Datensatz kann dieselben Felder enthalten, die in einem MARC-21-Datensatz erlaubt sind. Im Unterschied zu traditionellem MARC 21, wo die Strukturierung der einzelnen Datenfelder speichersparsam mittels Directory und Subfeld-Codes erfolgt, wird diese Struktur bei MARCXML mit Hilfe von XML-Elementen und -Attributen dargestellt. Abschnitt A2 im Anhang (S. 11) enthält einen Datensatz im MARCXML-Format, auf dessen Zeilen im Folgenden als Anschauungsbeispiel verwiesen wird.

MARCXML nutzt die hierarchische Struktur von XML, um Datensätze zu beschreiben und zu trennen. Ein Datensatz wird dabei stets von einem `<record>`-Element umschlossen (vgl. Zeile 3). Ein einzelner Datensatz kann so in einer MARCXML-Datei gespeichert werden; das `<record>`-Element ist dann das Wurzelelement der Datei. Um mehrere Datensätze in einer MARCXML-Datei zu speichern, müssen die `<record>`-Elemente in einem `<collection>`-Element zusammengefasst werden; in diesem Fall ist das `<collection>`-Element das Wurzelelement (vgl. Zeile 2).[28]

Jedes `<record>`-Element muss *genau ein* `<leader>`-Element sowie eine beliebige Anzahl von `<controlfield>`- und `<datafield>`-Elementen enthalten. Das `<leader>`-Element (vgl. Zeile 4) enthält den 24 Zeichen langen Leader des Datensatzes (siehe Abschnitt 2., S. 4). Allerdings sind einige Datenfelder des Leaders für die Repräsentation im MARCXML-Format nicht relevant; beispielsweise hat die Größe des Datensatzes für verarbeitende Software keine Bedeutung mehr, da sich diese aus dem Aufbau der XML-Struktur ergibt und ohnehin von der Größe des zugehörigen MARC-21-Datensatzes abweicht. Diese Teile des Leaders dürfen beliebig belegt werden, traditionell übernimmt man die Daten aus dem MARC-21-Datensatz.[29]

`<controlfield>`-Elemente enthalten Datenfelder der Nummern 001–009, also Datenfelder, die nicht in Subfelder unterteilt sind. Jedes Datenfeld wird durch ein eigenes `<controlfield>`-Element dargestellt. Dabei wird die Datenfeldnummer in dem Attribut `tag`, der eigentliche Inhalt des jeweiligen Datenfeldes als vom Element umschlossener Text hinterlegt (vgl. Zeile 5).

Datenfelder ab der Nummer 010 werden von `<datafield>`-Elementen repräsentiert. Auch hier wird ein Element pro Datenfeld verwendet und die Nummer im Attribut `tag` hinterlegt (vgl. Zeile 9). Die beiden Indikatoren stehen in den Attributen `ind1` und `ind2` als einzelne Zeichen. Die Subfelder jedes Datenfeldes werden in – dem jeweiligen `<datafield>`-Element untergeordneten – `<subfield>`-Elementen gespeichert:

[27] Eine anschauliche Einführung in XML findet sich bei Pott & Wielage (1999).
[28] Cole & Han (2013, Chapter 4, S. 75).
[29] Cole & Han (2013, Chapter 4, S. 79).

Der Subfeld-Code steht dabei im Attribut `code` des `<subfield>`-Elements (wieder als einzelnes Zeichen), der eigentliche Inhalt ist vom `<subfield>`-Element umschlossener Text (vgl. Zeile 10).

4. Implementierung des Konverters

Dieser Abschnitt beschreibt den Aufbau des Konverters und erläutert insbesondere einige Details der Implementierung, die eventuell für eine spätere Erweiterung relevant sind. Dabei wird von einer grundlegenden Kenntnis der Programmiersprache Perl ausgegangen.[30] Der Perl-Code des Konverters, der im Rahmen der vorliegenden Arbeit entwickelt wurde, ist in Anhang C (S. 15) abgedruckt. Die Zeilenangaben im Folgenden beziehen sich auf diesen Code.

Der UNIX-Philosophie[31] folgend hat das Programm eine einfache Ein- und Ausgabestrategie: Die MARC-21-Datensätze werden von der Standardeingabe gelesen[32], die MARCXML-Datensätze werden an die Standardausgabe geschrieben.[33] Weitere Informationen über die Nutzung des Programms finden sich auch in der „Handbuchseite“ (engl. *manual page*), die in Anhang B (S. 13) abgedruckt ist.

Das Programm besteht im Wesentlichen aus zwei Teilen:

- Der *Parser* liest und interpretiert einen MARC-21-Datensatz, und erzeugt daraus eine interne Struktur, die der Gliederung des Datensatzes entspricht.
- Der *Dumper* erzeugt aus dieser Struktur XML-Elemente und gibt diese aus.

Der Parser ist das komplexere Gebilde, unter anderem da er Fehler in den Datensätzen zuverlässig erkennen können muss. Um die verschiedenen Phasen des Parsers sauber zu trennen und eventuell auftretende Fehler rechtzeitig zu behandeln, sind alle Methoden des Parsers in der Klasse `MARC21Record` (ab Zeile 95, in Perl ein sog. „`package`“) zusammengefasst. Diese Klasse speichert die vom Parser gelesenen Daten und enthält der Einfachheit halber auch den Code des Dumpers. Alle Methoden der Klasse bieten eine möglichst allgemeine Schnittstelle, damit die Klasse bei Bedarf auch in größere Perl-Projekte integriert werden kann; so können dem Parser und dem Dumper beispielsweise Dateihandles[34] übergeben werden, um Ein- und Ausgaben programmatisch umzuleiten.

[30] Eine Einführung in die Programmiersprache Perl sowie eine Referenz für die hier verwendeten Begriffe und Techniken findet sich bei Poe (2012).

[31] Raymond (2003, Section 1.6.1, Section 1.6.3).

[32] Die Standardeingabe („stdin“) ist gewöhnlich mit der Tastatur verbunden, sollte hier aber vom Aufrufer mit einer MARC-Datei verknüpft werden.

[33] Die Standardausgabe („stdout“) wird gewöhnlich am Bildschirm ausgegeben und sollte daher hier vom Aufrufer mit einer Ausgabedatei verknüpft werden.

[34] Ein Dateihandle repräsentiert in den meisten Betriebssystemen und auch in Perl den Zugang zu einer geöffneten Datei. Es wird benötigt, um aus der Datei zu lesen und in sie zu schreiben.

4.1 Der Parser

Der Parser gliedert sich in vier Subroutinen, die – in der folgenden Reihenfolge aufgerufen – Stück für Stück die interne Repräsentation eines Datensatzes aufbauen.

- `read_record` (ab Zeile 207) liest einen Datensatz vollständig ein und speichert ihn im Attribut[35] `record`.
- `parse_leader` (ab Zeile 232) interpretiert den Leader des Datensatzes. Anhand der Daten des Leaders werden aus dem Datensatz das Directory und die Datenfelder (beide jeweils als eine lange Zeichenkette) extrahiert und in den Attributen `dirstring` und `dataarea` hinterlegt. Daneben wird geprüft, ob alle Informationen im Leader konsistent sind.
- `parse_directory` (ab Zeile 277) zerlegt die im Attribut `dirstring` enthaltene Zeichenkette in ihre einzelnen Einträge und speichert diese in ein vom Attribut `directory` referenziertes Array.
- `parse_fields` (ab Zeile 329) verwendet dieses Array, um die einzelnen Datenfelder aus dem Attribut `dataarea` zu extrahieren und in den Arrays in den Attributen `datafields` und `controlfields` zu speichern. In `controlfields` werden die Einträge mit den Datenfeldnummern 000–009 gespeichert, in `datafields` die Einträge ab 010. Dabei werden die Nummer und der Inhalt getrennt abgelegt. Bei den Feldern ab der Nummer 010 werden die beiden Indikatoren extrahiert; die Subfelder werden von der Methode `parse_subfields` (ab Zeile 300) ausgelesen und gespeichert, die von `parse_fields` aufgerufen wird.

Diese Subroutinen sind Methoden der Klasse `MARC21Record`. Die genaue Struktur der Attribute, die von diesen Subroutinen erzeugt werden, geht aus dem Quelltext-Kommentar ab Zeile 103 hervor.

Jede dieser Subroutinen kann zudem Warnungen und Fehlermeldungen erzeugen, die – wenn sie auftreten – in den Attributen `warning` und `error` gespeichert werden. Dabei gilt die Konvention: Eine Warnung weist auf einen unerwarteten Aufbau des Datensatzes hin (beispielsweise ein Konflikt mit den Spezifikationen von MARC 21, aber auch eine von UTF-8 verschiedene Zeichenkodierung), kann aber ignoriert werden. Ein Fehler verhindert, dass der betroffene Datensatz weiterverarbeitet wird. Zum Füllen der Attribute werden die Methoden `set_warning` (ab Zeile 179) und `set_error` (ab Zeile 166) verwendet, die auch sicherstellen, dass eine Warnung bzw. eine Fehlermeldung nicht später überschrieben wird.

[35] In Abschnitt 4. bezieht sich der Begriff „Attribut“ auf Variablen der Klasse `MARC21Record` und ist nicht zu verwechseln mit Attributen von XML-Elementen.

Die erste Routine, `read_record`, wird bereits bei der Initialisierung einer Instanz von `new` automatisch aufgerufen (Zeile 162). Ein Fehler in `read_record` hat zur Folge, dass keine weiteren Datensätze eingelesen werden können, da entweder der Eingabestrom (die Standardeingabe) während des Lesens des Datensatzes versiegt ist, oder wegen eines Fehlers im Leader nicht klar ist, wo der aktuelle Datensatz endet und ein neuer beginnt. Falls `read_record` kein einziges Byte lesen kann (d. h. bereits beim Lesen des ersten Bytes auf das Dateiende stößt), liegt kein echter Fehler vor. Vielmehr sind lediglich keine weiteren Datensätze zur Verarbeitung vorhanden. Die Routine `new` erkennt dies und gibt `undef` statt einer Instanz der Klasse `MARC21Record` zurück.

4.2 Der Dumper

Wurde der MARC-21-Datensatz erfolgreich von den Parser-Routinen zerlegt, kann der zugehörige MARCXML-Datensatz mit geringem Aufwand erzeugt werden. Die Subroutine `dump_xml` (ab Zeile 414) iteriert über die in den Attributen `controlfields` und `datafields` enthaltenen Arrays und gibt diese sowie den Leader mit den notwendigen XML-Auszeichnungen aus. Die für valides XML erforderlichen Zeichenersetzungen (beispielsweise „<“ in „`<`“) werden dabei von der Subroutine `xml_escape` (ab Zeile 388) übernommen.

Um die Verarbeitung vieler Datensätze zu beschleunigen, werden die nötigen Auszeichnungen bereits vor dem Einlesen des ersten Datensatzes von der Subroutine `make_xml_templates` (ab Zeile 22) gebildet und für `dump_xml` vorgehalten.

4.3 Verwendung der Klasse `MARC21Record`

Der gesamte Programmablauf wird durch die Subroutine `main` (ab Zeile 499, inspiriert durch die Programmiersprache C) organisiert. Sie interpretiert zunächst die Befehlszeilenargumente (mit Hilfe der Subroutine `process_command_line`, ab Zeile 457) und verarbeitet dann Datensatz für Datensatz in einer Schleife (ab Zeile 564), bis entweder ein Fehler auftritt oder der letzte Datensatz konvertiert wurde.

Wird ein Datensatz als fehlerhaft erkannt, ist das Verhalten abhängig von den vom Nutzer angegebenen Befehlszeilenargumenten: Der betroffene Datensatz wird entweder übersprungen, oder die Verarbeitung wird abgebrochen. Warnungen können auch ignoriert werden. Da nach jedem Schritt des Parsers geprüft werden muss, ob ein Fehler aufgetreten ist, wurde der Fehlerbehandlungscode in die Subroutine `record_report_skip` ausgelagert (ab Zeile 535, ein sog. Closure innerhalb der Subroutine `main`). Diese wird nach jedem Schritt des Parsers ausgeführt; im Fehlerfall wird eine entsprechende Meldung ausgegeben. Da die Ausgabe datensatzbezogener Meldungen an verschiedenen Stellen

notwendig ist, existiert hierfür die Subroutine `record_msg` (ab Zeile 520, ebenfalls ein Closure innerhalb von `main`). Diese annotiert jede Meldung mit der laufenden Nummer des betroffenen Datensatzes und dem Inhalt von Datenfeld 001. Letzteres wird mittels der Methode `get_controlfield` (ab Zeile 192) aus dem Datensatz gewonnen und enthält den lokalen Identifier des Datensatzes.[36]

5. Erste Erfahrungen und Schlussbemerkungen

Das im Rahmen dieser Arbeit entwickelte Programm erlaubt, MARC-21-Datensätze schnell und automatisiert in MARCXML zu konvertieren. Ein Testlauf mit allen 25,6 Millionen Datensätzen des B3Kat benötigte 15,6 Stunden (Testsystem mit Intel Core i3-2100, 3.10GHz). Durch die Prüfung der MARC-Spezifikationen im Programm können auch fehlerhafte Datensätze aufgedeckt werden. Bei den Testläufen im Rahmen der Entwicklung wurden 54 Datensätze mit fehlerhaft formatierten Subfeldern erkannt. Erste Recherchen durch die BVB-Verbundzentrale legen einen bislang unentdeckten Fehler bei der Erzeugung der MARC-21-Datensätze nahe, der nun untersucht und behoben werden kann.[37]

Für den Umgang mit den unterschiedlichen Datenformaten ist die hier konzipierte und implementierte Konversionsroutine nur ein kleiner Schritt, insbesondere da MARC 21 und MARCXML dieselben Datenfelder verwenden. Die führende Rolle von MARC unter den bibliographischen Datenformaten wird bereits seit geraumer Zeit in Frage gestellt.[38] Ein vielversprechender Nachfolger ist das RDF-basierte bibliographische Format BIBFRAME, das im November 2012 von der LoC vorgestellt wurde.[39] Diese Entwicklungen sind wichtig, um auch in Zukunft eine effiziente Verarbeitung und Präsentation der Katalogdaten sicherzustellen; sie werden aber auch zukünftig die Entwickung von Konvertierungsroutinen notwendig machen, um bestehende Daten weiternutzen zu können.

[36] Im B3Kat ist der lokale Identifier die sog. BV-Nummer, die den Datensatz innerhalb des B3Kat eindeutig identifiziert.
[37] Gespräch mit Mathias Kratzer und Norbert Lauer am 3. März 2015.
[38] Tennant (2002).
[39] Kroeger (2013).

A Beispielhafter MARC-Datensatz

Dieser Anhang illustriert anhand eines kurzen Datensatzes die Formate MARC 21 und MARCXML. Der Datensatz selbst stammt von der Library of Congress (2013). Er wurde für die Illustration in dieser Arbeit leicht abgewandelt.

A1 Datensatz im Format MARC 21

Jede Zeile des folgenden Datensatzes enthält 50 Zeichen. Die Zeilenumbrüche selbst sind nicht Teil des Datensatzes, sie sind lediglich der beschränkten Seitenbreite geschuldet – der Datensatz selbst enthält keine Zeilenumbruchszeichen. Ebenso wurden die (nicht druckenden) Steuerzeichen `0x1d`, `0x1e` und `0x1f`, durch die Zeichen „*“, „^“ und „~“ ersetzt.

Leader

Directory (Eintrag für Feld 100 hervorgehoben)

Inhalt von Datenfeld 100 (Subfeld-Codes hervorgehoben)

```
01041cam a2200265 a 4500001002000000003000400020005001700024008004100041010002400082020002500106020004400131040001800175050002400193082001800217100003200235245008700267246003600354250001200390260003700402300002900439500004200468520022000510650003300730650001200763^   89048230 /AC/r91^DLC^19911106082810.9^891101s1990    maua   j      000 0 eng  ^  ~a   89048230 /AC/r91^  ~a0316107514 :~c$12.95^  ~a0316107506 (pbk.) :~c$5.95 ($6.95 Can.)^  ~aDLC~cDLC~dDLC^00~aGV943.25~b.B74 1990^00~a796.334/2~220^10~aBrenner, Richard J.,~d1941-^10~aMake the team. ~pSoccer :~ba heads up guide to super soccer! /~cRichard J. Brenner.^30~aHeads up guide to super soccer.^  ~a1st ed.^  ~aBoston :~bLittle, Brown,~cc1990.^  ~a127 p. :~bill. ;~c19 cm.^  ~a"A Sports illustrated for kids book."^  ~aInstructions for improving soccer skills. Discusses dribbling, heading, playmaking, defense, conditioning, mental attitude, how to handle problems with coaches, parents, and other players, and the history of soccer.^ 0~aSoccer~vJuvenile literature.^ 1~aSoccer.^*
```

A2 Datensatz im Format MARCXML

Im Folgenden ist derselbe Datensatz im MARCXML-Format abgedruckt. Einige Zeilen des Datensatzes sind zu lang und daher hier auf mehrere Zeilen umgebrochen. Diese sind an den fehlenden Zeilennummern zu erkennen.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <collection xmlns="http://www.loc.gov/MARC21/slim"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.loc.gov/MARC21/slim
     http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd">
3    <record>
4      <leader>01041cam a2200265 a 4500</leader>
5      <controlfield tag="001">   89048230 /AC/r91</controlfield>
6      <controlfield tag="003">DLC</controlfield>
7      <controlfield tag="005">19911106082810.9</controlfield>
```

```
<controlfield tag="008">891101s1990    maua   j      000 0 eng
  </controlfield>
<datafield tag="010" ind1=" " ind2=" ">
  <subfield code="a">   89048230 /AC/r91</subfield>
</datafield>
<datafield tag="020" ind1=" " ind2=" ">
  <subfield code="a">0316107514 :</subfield>
  <subfield code="c">$12.95</subfield>
</datafield>
<datafield tag="020" ind1=" " ind2=" ">
  <subfield code="a">0316107506 (pbk.) :</subfield>
  <subfield code="c">$5.95 ($6.95 Can.)</subfield>
</datafield>
<datafield tag="040" ind1=" " ind2=" ">
  <subfield code="a">DLC</subfield>
  <subfield code="c">DLC</subfield>
  <subfield code="d">DLC</subfield>
</datafield>
<datafield tag="050" ind1="0" ind2="0">
  <subfield code="a">GV943.25</subfield>
  <subfield code="b">.B74 1990</subfield>
</datafield>
<datafield tag="082" ind1="0" ind2="0">
  <subfield code="a">796.334/2</subfield>
  <subfield code="2">20</subfield>
</datafield>
<datafield tag="100" ind1="1" ind2="0">
  <subfield code="a">Brenner, Richard J.,</subfield>
  <subfield code="d">1941-</subfield>
</datafield>
<datafield tag="245" ind1="1" ind2="0">
  <subfield code="a">Make the team.</subfield>
  <subfield code="p">Soccer :</subfield>
  <subfield code="b">a heads up guide to super soccer! /</subfield>
  <subfield code="c">Richard J. Brenner.</subfield>
</datafield>
<datafield tag="246" ind1="3" ind2="0">
  <subfield code="a">Heads up guide to super soccer.</subfield>
</datafield>
<datafield tag="250" ind1=" " ind2=" ">
  <subfield code="a">1st ed.</subfield>
</datafield>
<datafield tag="260" ind1=" " ind2=" ">
  <subfield code="a">Boston :</subfield>
  <subfield code="b">Little, Brown,</subfield>
  <subfield code="c">c1990.</subfield>
</datafield>
<datafield tag="300" ind1=" " ind2=" ">
  <subfield code="a">127 p. :</subfield>
  <subfield code="b">ill. ;</subfield>
  <subfield code="c">19 cm.</subfield>
</datafield>
<datafield tag="500" ind1=" " ind2=" ">
  <subfield code="a">&amp;quot;A Sports illustrated for kids
    book.&amp;quot;</subfield>
</datafield>
<datafield tag="520" ind1=" " ind2=" ">
  <subfield code="a">Instructions for improving soccer skills. Discusses
    dribbling, heading, playmaking, defense, conditioning, mental attitude,
    how to handle problems with coaches, parents, and other players, and the
    history of soccer.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="0">
  <subfield code="a">Soccer</subfield>
  <subfield code="v">Juvenile literature.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="1">
  <subfield code="a">Soccer.</subfield>
```

```
71      </datafield>
72    </record>
73  </collection>
```

B Englischsprachige Handbuchseite

Das Konverter-Modul enthält eine „Handbuchseite“ (engl. *Manual Page*), die in der für Perl-Module und -Programme üblichen Auszeichnungssprache POD[40] verfasst und Teil der Skript-Datei ist (ab Zeile 615). Um eine über Sprachgrenzen hinweg reichende Verbreitung des Programms zu ermöglichen, ist die Handbuchseite in englischer Sprache verfasst. Ihr Inhalt ist in diesem Anhang abgedruckt.

NAME

marc21-marcxml - convert MARC 21 records to MARCXML

SYNOPSIS

marc21-marcxml [options]

DESCRIPTION

Read one or several concatenated MARC 21 records from standard input and write corresponding MARCXML records to standard output.

OPTIONS

Record Processing

`--strict`

> Abort conversion if a record violates MARC 21 specifications (as far as known to this program).
>
> By default, such violations trigger warnings, which aren't visible unless `--verbose` is used.

`--skip` {*/path/to/waste-basket*|”}

> Don't abort conversion if a record causes an error (possibly due to `--strict`), skip the record instead. By default, conversion is aborted whenever a record cannot be processed.
>
> This option expects the path to a file as argument; skipped records will be appended to this file. To simply drop skipped records, provide the empty string as argument (possibly a pair of apostrophes, depending on your shell).
>
> Note that conversion is still aborted when the record length (first five bytes of the MARC leader) cannot be parsed.

[40] Poe (2012, Chapter 11, S. 338–344).

XML Generation

`--header`

Add an XML declaration and `collection` root element to the output. If this option is ommitted, only bare `record` elements will be written to standard output. This allows for easy concatenation of the outputs of several invocations, but the result likely needs to be enclosed in a `collection` element for further processing.

`--namespace` {*name*|"}

Explicitely define and use the given namespace for all XML elements (`marc` if the empty string is provided as argument). If ommitted, a default namespace is set in the `collection` root element.

`--indent`

Add indentation to XML output, so as to make it more human-readable.

Miscellaneous Options

`--verbose`

Print warnings to standard error, in addition to error messages. Use twice to print a summary line for each record processed, and three times to print additional debug information about each record.

`--version`

Print version information and exit.

`--help`

Print usage information and exit. Use twice to obtain a short manual page.

EXIT STATUS

`4`

The file given to the `--skip` option could not be opened for appending.

`3`

Conversion was aborted because a MARC 21 record could not be read completely (possibly due to a broken leader).

`2`

Conversion was aborted while a record was being processed, due to an error or due to `--strict`.

`1`

At least one record was skipped.

`0`

All records have been converted successfully.

AUTHOR

Written by Wolfgang Boiger (Wolfgang.Boiger@bsb-muenchen.de) as part of a librarian clerkship assignment.

COPYRIGHT

C Perl-Implementierung

Dieser Anhang enthält den gesamten Perl-Code, der im Rahmen der vorliegenden Arbeit entwickelt wurde.[41]

```
#!/usr/bin/perl

use strict;
use warnings;
use open IO => ':bytes';

use Encode qw'decode encode';
use Pod::Usage qw'pod2usage';
use Getopt::Long qw':config no_auto_abbrev no_getopt_compat
             no_ignore_case no_bundling auto_version auto_help';


## global constants
my @version = (1,0,2);
my $xml_indent = "\t";
my $xml_badchar_substitute = " ";



## xml markup template generator

sub make_xml_templates
# Return a reference to a hash containing xml template strings
# used to assemble a MARCXML file.  Boolean args (in order):
#  * pretty formatting / indentation
#  * add explicit namespace ("marc" by default) to tags
#  * add xml header and <collection> root element
{
  my $indent = shift;
  my $namespace = shift;
  my $header = shift;
```

[41] Die shar-Prüfsumme der zugehörigen Datei lautet `066632c46d98ea1b0c92a536f2e13131681e6ecc`. Die elektronische Version dieses Dokuments enthält die Implementierung auch als eingebettete Datei.

```
  # evaluate arguments
  my $newline = $indent?"\n":"";
  $indent = $indent?$xml_indent:"";
  $namespace = defined($namespace)?($namespace or "marc"):"";
  my $nscolon = $namespace?":":"";
  # two helper functions to create templates
  my $mkstart = sub  # args: indent, name+attrs, final_newline?
  {
    return sprintf('%s<%s%s%s>%s',$indent x $_[0],$namespace,
                                $nscolon,$_[1],$_[2]?$newline:"");
  };
  my $mkend = sub  # args: indent, name
  {
    return sprintf('%s</%s%s%s>%s',$indent x $_[0],$namespace,
                                       $nscolon,$_[1],$newline);
  };
  # assemble templates
  my $xml = {
      header => [
          sprintf('%s%s<%s%scollection xmlns%s%s="%s" %s %s>%s',
              '<?xml version="1.0" encoding="UTF-8" ?>',
              $newline,
              $namespace,
              $nscolon,
              $nscolon,
              $namespace,
              'http://www.loc.gov/MARC21/slim',
              'xmlns:xsi='.
                  '"http://www.w3.org/2001/XMLSchema-instance"',
              'xsi:schemaLocation="'.
                  'http://www.loc.gov/MARC21/slim'.
                  ' '.
                  'http://www.loc.gov/standards/'.
                  'marcxml/schema/MARC21slim.xsd'.
                 '"',
              $newline,
             ),
          &$mkend(0,'collection'),
         ],
      record => [&$mkstart(1,'record',1),&$mkend(1,'record')],
      leader => [&$mkstart(2,'leader',0),&$mkend(0,'leader')],
      controlfield => [
          &$mkstart(2,'controlfield tag="%03d"'),
          &$mkend(0,'controlfield'),
         ],
      datafield => [
          &$mkstart(2,
                'datafield tag="%03d" ind1="%1s" ind2="%1s"',1),
          &$mkend(2,'datafield'),
         ],
      subfield => [
          &$mkstart(3,'subfield code="%1s"'),
          &$mkend(0,'subfield'),
         ],
     };
  # clear header template if no header is requested
  $xml->{header} = ["",""] unless ($header);
  return $xml;
}


## core record parser and dumper class

{package MARC21Record;
# A class that reads, parses and
# dumps (as xml) MARC 21 records.
# Each step is encapsulated in its own function and
# stores messages in the attributes 'warning'
```

```
100 # and/or 'error' if such a thing occurs.
101 # Then the caller may decide on how to proceed.
102 #
103 # The attribute structure of this class reads as follows:
104 # {
105 #   "record" => <record as MARC 21 string>,
106 #   "leader" => <24 character leader substring of the record>,
107 #   "dirstring" => <directory substring of the record>,
108 #   "dataarea" => <data fields substring of the record>,
109 #   "directory" => [
110 #     {
111 #       "tag" => <tag number>,
112 #       "length" => <length of data field>,
113 #       "address" => <address (in dataarea) of data field>,
114 #     }, ...
115 #   ],
116 #   "controlfields" => [
117 #     {
118 #       "tag" => <3 digits number in the range 0--9 !!>,
119 #       "content" => <field content as character string>,
120 #     }, ...
121 #   ],
122 #   "datafields" => [
123 #     {
124 #       "tag" => <3 digits number in the range 10--999 !!>,
125 #       "ind1" => <first indicator character>,
126 #       "ind2" => <second indicator character>,
127 #       "subfields" => [
128 #         {
129 #           "code" => <code character>,
130 #           "content" => <subfield content as character string>,
131 #         },...
132 #       ],
133 #     },...
134 #   ],
135 #   "error" => <parsing error (prevents further processing)>,
136 #   "warning" => <parsing warning (e.g. spec violation)>,
137 # }
138 #
139 # Note on comments in sub heads: argument numbers do *not*
140 # include $this, so "arg 1" is obtained with the *second* shift.
141
142   sub new
143   # Initialize the object, read the record from
144   # the filehandle (arg 1) via read_record.
145   # Return the blessed object, or undef if
146   # EOF is hit while reading the first byte.
147   {
148     my $classname = shift;
149     my $filehandle = shift;
150     my $this = {
151         record=>undef,
152         leader=>undef,
153         dirstring=>undef,
154         dataarea=>undef,
155         directory=>[],
156         controlfields=>[],
157         datafields=>[],
158         error=>undef,
159         warning=>undef,
160        };
161     bless $this,$classname;
162     $this->read_record($filehandle);
163     return ($this->{record}?$this:undef);
164   }
165
166   sub set_error
167   # Unless the condition (arg 1) is true or the
```

```
168     # attribute is already occupied, store the
169     # error message (arg 2) in the 'error' attribute.
170     # Return the state of the attribute (1 or 0).
171     {
172       my $this = shift;
173       my $cond = shift;
174       my $msg = shift;
175       $this->{error} = $msg unless ($cond or $this->{error});
176       return defined($this->{error});
177     }
178
179     sub set_warning
180     # Unless the condition (arg 1) is true or the
181     # attribute is already occupied, store the
182     # warning message (arg 2) in the 'warning' attribute.
183     # Return the state of the attribute (1 or 0).
184     {
185       my $this = shift;
186       my $cond = shift;
187       my $msg = shift;
188       $this->{warning} = $msg unless ($cond or $this->{warning});
189       return defined($this->{warning});
190     }
191
192     sub get_controlfield
193     # Find and return first content of first occurrence
194     # of the controlfield with the given tag (arg 1),
195     # or undef if it doesn't exsit (yet).
196     # This is mainly used to aid in error reporting.
197     {
198       my $this = shift;
199       my $tag = shift;
200       for my $field (@{$this->{controlfields}})
201       {
202         return $field->{content} if $field->{tag}==$tag;
203       }
204       return;
205     }
206
207     sub read_record
208     # Read the record from the given file handle (arg 1)
209     # and store it in the 'record' attribute.
210     # If no bytes can be read, no error is reported,
211     # but 'record' won't be touched.
212     {
213       my $this = shift;
214       my $filehandle = shift;
215       return if defined($this->{error});
216       # first try to read the size (5 bytes), store whatever comes
217       return unless read($filehandle,my $length,5);
218       $this->{record} = $length;
219       # check if the size makes sense
220       return if $this->set_error(scalar($length=~m'\d{5}'),
221                                        "invalid record length");
222       return if $this->set_error(scalar($length>5),
223                                          "record is too small");
224       # attempt to read the rest, check for completeness
225       read($filehandle,my $record,$length-5);
226       $this->{record} .= $record;
227       $this->set_error(length($this->{record})==$length,
228                                  "failed to read complete record");
229       return;
230     }
231
232     sub parse_leader
233     # Fill 'leader', 'dirstring' and 'dataarea',
234     # also perform some sanity checks on the leader.
235     {
```

```
236       my $this = shift;
237       return if defined($this->{error});
238       # fill 'leader' attribute, then extract interesting fields
239       $this->{leader} = substr($this->{record},0,24);
240       my @values = ($this->{leader}=~
241                       m'^(\d{5})....(.)(..)(\d{5})...(...)(.)$');
242       return if $this->set_error(scalar(@values),
243                                              "malformed leader");
244       my %ldr;
245       @ldr{(qw'length coding is_cnt data_addr
246                                        lsi_cnt undef')} = @values;
247       # check fields for sanity
248       # (see <https://www.loc.gov/marc/umb/um07to10.html#part9>
249       # or <http://www.loc.gov/standards/
250       # marcxml/schema/MARC21slim.xsd>)
251       $this->set_error($ldr{data_addr}<$ldr{length},
252                    "invalid data address (beyond record length)");
253       $this->set_error($ldr{data_addr}>=25,
254                          "invalid data address (within leader)");
255       return if defined($this->{error});
256       $this->set_warning($ldr{coding} eq 'a',
257                                  "character coding is not UTF-8");
258       $this->set_warning(scalar($ldr{is_cnt}=~m'[2 ][2 ]'),
259                    "indicator and/or subfield count is not '2'");
260       $this->set_warning(scalar($ldr{lsi_cnt}=~m'[4 ][5 ][0 ]'),
261                        "portions lengths violate specification");
262       $this->set_warning($ldr{undef} eq '0',
263                                     "undefined field is not '0'");
264       $this->set_warning(
265                    substr($this->{record},$ldr{data_addr}-1,1),
266            "field terminator (0x1e) missing at end of directory");
267       $this->set_warning(
268                substr($this->{record},$ldr{length}-1) eq "\x1d",
269                           "record terminator (0x1d) not found");
270       # fill remaining fields
271       $this->{dirstring} = substr($this->{record},24,
272                                              $ldr{data_addr}-25);
273       $this->{dataarea} = substr($this->{record},$ldr{data_addr});
274       return;
275     }
276
277     sub parse_directory
278     # Parse the 'dirstring' attribute, fill 'directory'.
279     {
280       my $this = shift;
281       return if defined($this->{error});
282       # check length of directory
283       return if $this->set_error(length($this->{dirstring})%12==0,
284                                           "malformed directory");
285       # loop over 12-byte-blocks
286       my $count = 0;  # for error reporting
287       for my $block ($this->{dirstring}=~m'.{12}'g)
288       {
289         my @data = ($block=~m'(\d{3})(\d{4})(\d{5})');
290         return if $this->set_error(scalar(@data),
291                               "directory entry #$count malformed");
292         my %entry;
293         @entry{(qw'tag length address')} = @data;
294         push @{$this->{directory}}, \%entry;
295         $count++;
296       }
297       return;
298     }
299
300     sub parse_subfields
301     # Parse the subfields of a given datafield string (tag>9).
302     # Return the array (reference thereof)
303     # needed for the 'subfields' element.
```

```
304   {
305     # note: perlcritic dislikes undef as explicit return value,
306     # but this suppresses an interpreter
307     # warning in case of broken records.
308     my $this = shift;
309     my $str = shift;
310     return undef if defined($this->{error});  ## no critic
311     # each subfield starts with separator (0x1f)
312     return undef  ## no critic
313                if $this->set_error(substr($str,0,1) eq "\x1f",
314                 "subfield separator (0x1f) mismatch/not found");
315     my @fields;
316     my $count = 0;  # for error reporting
317     for my $block (split("\\\x1f",substr($str,1)))
318     {
319       # each subfield starts with the subfield code
320       return undef if $this->set_error($block,  ## no critic
321                           "subfield #$count: no subfield code");
322       push @fields,
323            {code=>substr($block,0,1),content=>substr($block,1)};
324       $count++;
325     }
326     return \@fields;
327   }
328
329   sub parse_fields
330   # Parse the 'dataarea' attribute,
331   # fill 'controlfields' and 'datafields'.
332   {
333     my $this = shift;
334     return if defined($this->{error});
335     my $count = 0;  # for error reporting
336     for my $entry (@{$this->{directory}})
337     {
338       # warn if terminator character at end of block is missing
339       my $sep = substr($this->{dataarea},
340                        $entry->{address}+$entry->{length}-1,1);
341       $this->set_warning($sep eq "\x1e",
342                          "field #$count (tag $entry->{tag}): ".
343                                 "terminator mismatch/not found");
344       # also check special tag 8 (specs require 40 characters)
345       $this->set_warning(
346                  ($entry->{tag}!=8) or ($entry->{length}==41),
347                                    "field #$count (tag 008): ".
348                                   "length is not 40 characters");
349       # control fields (tag<10) and data fields (tag>=10)
350       # need different treatment
351       if ($entry->{tag}<10)
352       {
353         my $field = {
354             tag     => $entry->{tag},
355             content => substr($this->{dataarea},
356                               $entry->{address},$entry->{length}-1),
357           };
358         push @{$this->{controlfields}},$field;
359       }
360       else
361       {
362         my $field = {
363             tag       => $entry->{tag},
364             ind1      => substr($this->{dataarea},
365                                         $entry->{address},1),
366             ind2      => substr($this->{dataarea},
367                                       $entry->{address}+1,1),
368             subfields => $this->parse_subfields(
369                 substr($this->{dataarea},
370                        $entry->{address}+2,$entry->{length}-3)),
371           };
```

```
372         # if an error occured, add the current data field's
373         # number and tag to the message, then stop processing
374         if (defined($this->{error}))
375         {
376           $this->{error} =
377                         "field #$count (tag $entry->{tag}): ".
378                                                 $this->{error};
379           return;
380         }
381         push @{$this->{datafields}},$field;
382       }
383       $count++;
384     }
385     return;
386   }
387
388   sub xml_escape
389   # XML-escape '&', '<', '>', '"',
390   # replace illegal characters in the given string with
391   # $xml_badchar_substitute (see global constants).
392   # (This is not a class method!)
393   {
394     my $s = shift;
395     $s = main::decode("utf8",$s);
396     $s =~ s/&/&amp;/g;
397     $s =~ s/</&lt;/g;
398     $s =~ s/>/&gt;/g;
399     $s =~ s/"/&quot;/g;  #  "  #  (fix LaTeX code highlighting)
400     # escape illegal characters as detailed here
401     # <http://www.w3.org/TR/2000/REC-xml-20001006#charsets>
402     $s =~ s/([^
403         \x{9}
404         \x{a}
405         \x{d}
406         \x{20}-\x{d7ff}
407         \x{e000}-\x{fffd}
408         \x{10000}-\x{10ffff}
409        ])/$xml_badchar_substitute/gex;
410     $s = main::encode("utf8",$s);
411     return $s;
412   }
413
414   sub dump_xml
415   # Use the templates (arg 2, from make_xml_templates) to dump
416   # the record to the given file handle (arg 1) as MARCXML.
417   {
418     my $this = shift;
419     my $filehandle = shift;
420     my $xml = shift;
421     return if defined($this->{error});
422     print $filehandle
423         $xml->{record}->[0],
424         $xml->{leader}->[0],
425         xml_escape($this->{leader}),
426         $xml->{leader}->[1],
427        ;
428     for my $field (@{$this->{controlfields}})
429     {
430       printf $filehandle $xml->{controlfield}->[0],
431                                                 $field->{tag};
432       print $filehandle xml_escape($field->{content}),
433                                     $xml->{controlfield}->[1];
434     }
435     for my $field (@{$this->{datafields}})
436     {
437       printf $filehandle $xml->{datafield}->[0],$field->{tag},
438                                     xml_escape($field->{ind1}),
439                                     xml_escape($field->{ind2});
```

```
      for my $subfield (@{$field->{subfields}})
      {
        printf $filehandle $xml->{subfield}->[0],
                                  xml_escape($subfield->{code});
        print $filehandle xml_escape($subfield->{content}),
                                       $xml->{subfield}->[1];
      }
      print $filehandle $xml->{datafield}->[1];
    }
    print $filehandle $xml->{record}->[1];
    return;
  }
}


## command line interface

sub process_command_line
# Parse command line, return array with
#  * hash reference to xml templates
#  * verbosity (0,1,2)
#  * strictness (specs violations are errors)
#  * skip-file (skip broken records, write them to file)
{
  # parse command line
  my %options;
  if (not GetOptions(\%options,(
      "strict",
      "skip=s",
      "header",
      "namespace=s",
      "indent",
      "verbose+",
      "version",
      "help+",
     )))
  {
    print STDERR "Use '--help' for usage hints\n";
    exit 2;
  }
  # handle secondary ("Miscellaneous") options
  if (defined($options{version}))
  {
    $0 =~ m'.*/([^/]+)';
    print STDERR "$1 ",join(".",@version),"\n";
    exit;
  }
  if (defined($options{help}))
  {
    pod2usage(-verbose=>8) if ($options{help}>1);
    pod2usage(-verbose=>99,-sections=>['OPTIONS']);
  }
  # get xml templates, return
  my $xml = make_xml_templates(
                        @options{(qw'indent namespace header')});
  return ($xml,($options{verbose} or 0),
                                    @options{(qw'strict skip')});
}

sub main
{
  my ($xml,$verbose,$strict,$skip) = process_command_line();
  my $status = 0;  # proposed exit status, indicates abortion
  my $count;  # for progress/warning/error reporting
  my $record;  # holds the current record instance
  my $skipfh;  # file handle for skip file
  # if $strict is set, warnings can't be hidden
  $verbose = $verbose>1?$verbose:1 if $strict;
```

```
508   # open skip file
509   if ($skip)
510   {
511     # perlcritic says: close filehandle as soon as possible;
512     # however, the filehandle is still needed later
513     if (not open $skipfh, ">>", $skip)  ## no critic
514     {
515       print STDERR "error: failed to open '$skip'\n";
516       return 4;
517     }
518   }
519   # two helper functions to check record status
520   my $record_msg = sub
521   # Prefix the given message (arg 1) with a
522   # short string, which identifies the current record,
523   # then send it to stderr.
524   {
525     my $msg = shift;
526     if ($record)
527     {
528       my $cf = $record->get_controlfield(1);
529       $msg = sprintf("record %9s%s: %s",
530                      "#$count",$cf?" [tag001=$cf]":"",$msg);
531     }
532     print STDERR $msg, "\n";
533     return;
534   };
535   my $record_report_skip = sub
536   # Evaluate the warning/error attributes of the current record.
537   # Print error or warning message if appropriate.
538   # Update $status to indicate skipping or abortion.
539   # Also indicate skipping of record with return value 1.
540   # Errors don't cause abortion if $skip is true;
541   # to enforce abortion (after read_record), set arg 1 to 1.
542   # Note: Warnings are ignored (even with $strict, unless arg 2
543   # is 1) to defer warning messages until tag 001 got parsed.
544   {
545     my $abort = shift;  # force abort on error, even with $skip
546     my $warn = shift;  # print warning, even if harmless
547     # check and print warning
548     my $warning = $record->{warning};
549     my $error = $record->{error};
550     $warn ||= defined($error);
551     $warn &&= defined($warning) && $verbose;
552     &$record_msg("warning: $warning") if $warn;
553     return unless (defined($error) or ($warn and $strict));
554     # at this point, the record will certainly be skipped
555     &$record_msg("error: $error") if defined($error);
556     &$record_msg("skipped");
557     print $skipfh $record->{record} if ($skipfh and not $abort);
558     $status = $abort?3:defined($skip)?1:2;
559     return 1;
560   };
561   # core loop
562   $count = 0;
563   print STDOUT $xml->{header}->[0];
564   while ($record=MARC21Record->new(*STDIN))
565   {
566     next if &$record_report_skip(1,0);
567     # parse leader
568     $record->parse_leader();
569     &$record_msg(sprintf("%5d bytes, leader: '%s'",
570                  length($record->{record}),$record->{leader}))
571                                                 if $verbose>2;
572     next if &$record_report_skip(0,0);
573     # parse directory
574     $record->parse_directory();
575     next if &$record_report_skip(0,0);
```

```
576       &$record_msg(sprintf("tags: %s",
577             join(" ",map({$_->{tag}} @{$record->{directory}}))))
578                                                   if $verbose>2;
579       # parse data fields
580       $record->parse_fields();
581       next if &$record_report_skip(0,1);
582       # final log message
583       &$record_msg(sprintf(
584           "%5d bytes, %3d control field(s), %3d data field(s)",
585           length($record->{record}),
586           scalar(@{$record->{controlfields}}),
587           scalar(@{$record->{datafields}}),
588          )) if $verbose>1;
589       # dump xml
590       $record->dump_xml(*STDOUT,$xml);
591     }
592     continue
593     {
594       if ($status>1)
595       {
596         print STDERR "aborting conversion\n";
597         last;
598       }
599       $count++;
600     }
601     print STDOUT $xml->{header}->[1];
602     close $skipfh if $skip;
603     return $status;
604   }
605
606   exit(main());
607
608
609
610   ################################################################
611   __END__
612
613
614
615   =head1 NAME
616
617   marc21-marcxml - convert MARC 21 records to MARCXML
618
619   =head1 SYNOPSIS
620
621   marc21-marcxml [options]
622
623   =head1 DESCRIPTION
624
625   Read one or several concatenated MARC 21 records
626   from standard input and write corresponding
627   MARCXML records to standard output.
628
629   =head1 OPTIONS
630
631   =head2 Record Processing
632
633   =over
634
635   =item C<--strict>
636
637   Abort conversion if a record violates MARC 21 specifications
638   (as far as known to this program).
639
640   By default, such violations trigger warnings,
641   which aren't visible unless C<--verbose> is used.
642
643   =item C<--skip> {F</path/to/waste-basket>|''}
```

```
644
645 Don’t abort conversion if a record causes an error
646 (possibly due to C<--strict>), skip the record instead.
647 By default, conversion is aborted
648 whenever a record cannot be processed.
649
650 This option expects the path to a file as argument;
651 skipped records will be appended to this file.
652 To simply drop skipped records,
653 provide the empty string as argument
654 (possibly a pair of apostrophes, depending on your shell).
655
656 Note that conversion is still aborted when the record length
657 (first five bytes of the MARC leader) cannot be parsed.
658
659 =back
660
661 =head2 XML Generation
662
663 =over
664
665 =item C<--header>
666
667 Add an XML declaration and C<collection>
668 root element to the output.
669 If this option is ommitted, only bare C<record>
670 elements will be written to standard output.
671 This allows for easy concatenation of the outputs of
672 several invocations, but the result likely needs to be
673 enclosed in a C<collection> element for further processing.
674
675 =item C<--namespace> {I<name>|’’}
676
677 Explicitely define and use the given namespace for all XML
678 elements (C<marc> if the empty string is provided as argument).
679 If ommitted, a default namespace is set in the
680 C<collection> root element.
681
682 =item C<--indent>
683
684 Add indentation to XML output,
685 so as to make it more human-readable.
686
687 =back
688
689 =head2 Miscellaneous Options
690
691 =over
692
693 =item C<--verbose>
694
695 Print warnings to standard error, in addition to error messages.
696 Use twice to print a summary line for each record processed,
697 and three times to print additional
698 debug information about each record.
699
700 =item C<--version>
701
702 Print version information and exit.
703
704 =item C<--help>
705
706 Print usage information and exit.
707 Use twice to obtain a short manual page.
708
709 =back
710
711 =head1 EXIT STATUS
```

```
712
713 =over
714
715 =item C<4>
716
717 The file given to the C<--skip> option
718 could not be opened for appending.
719
720 =item C<3>
721
722 Conversion was aborted because a MARC 21
723 record could not be read completely
724 (possibly due to a broken leader).
725
726 =item C<2>
727
728 Conversion was aborted while a record was being
729 processed, due to an error or due to C<--strict>.
730
731 =item C<1>
732
733 At least one record was skipped.
734
735 =item C<0>
736
737 All records have been converted successfully.
738
739 =back
740
741 =head1 AUTHOR
742
743 Written by Wolfgang Boiger
744 (L<Wolfgang.Boiger@bsb-muenchen.de>)
745 as part of a librarian clerkship assignment.
746
747 =head1 COPYRIGHT
748
749 Copyright (C) 2015 Wolfgang Boiger, Muenchen.
750
751 Licence AGPL3+: GNU Affero GPL version 3 or later
752 L<https://www.gnu.org/licenses/agpl.html>.
753
754 This is free software;
755 you can use it, redistribute it and/or modify it under
756 the terms of the GNU Affero General Public License as
757 published by the Free Software Foundation;
758 either version 3 of the License,
759 or (at your option) any later version.
760
761 This software is distributed in the hope that it will be useful,
762 but WITHOUT ANY WARRANTY;
763 without even the implied warranty of MERCHANTABILITY
764 or FITNESS FOR A PARTICULAR PURPOSE.
765 See the GNU Affero General Public License for more details.
```

D Literatur

Böhme, C. (2014, 8. Juli). Culturegraph — Eine Plattform für die Datenvernetzung. https://www.ibi.hu-berlin.de/bbk/abstracts/ss14/boehme (abgerufen am 03. 11. 2014).

Cole, T. W. & Han, M.-J. (2013). *XML for Catalogers and Metadata Librarians*. Third millenium cataloging series. Santa Barbara, CA, USA: Libraries Unlimited.

Das, S. K. (2009). *Fundamentals of MARC 21 Bibliographical Format*. New Delhi, Indien: Ess Ess Publications.

Kroeger, A. (2013, 5. Sep.). The Road to BIBFRAME: The Evolution of the Idea of Bibliographic Transition into a Post-MARC Future. *Cataloging & Classification Quarterly*, *51*(8).

Library of Congress. (2000). *MARC 21 specifications for record structure, character sets, and exchange media*. Washington, D.C., USA: Library of Congress, Cataloging Distribution Service.

Library of Congress. (2013, 9. Sep.). Understanding MARC Bibliographic: Machine-Readable Cataloging. https://www.loc.gov/marc/umb/ (abgerufen am 16. 02. 2015).

Meßmer, G. (2013). Bibliographische Daten für alle frei zugänglich machen – der gemeinsame Verbundkatalog B3Kat als Linked Open Data. In K.-R. Brintzinger & U. Hohoff (Hrsg.), *Bibliotheken: Tore zur Welt des Wissens*, Mai 2012 (S. 72–79). Deutscher Bibliothekartag. 101. Deutscher Bibliothekartag. Hamburg. Hildesheim, Deutschland: Olms.

Mukhopadhyay, A. (2007). *Guide to MARC 21*. Oxford, England: Chandos Publishing.

Poe, C. (2012). *Beginning Perl*. Hoboken: Wiley.

Pott, O. & Wielage, G. (1999). *XML – Praxis und Referenz*. new technology. München: Markt & Technik.

Raymond, E. S. (2003, 23. Sep.). The Art of Unix Programming. http://proquest.safaribooksonline.com/0131429019 (abgerufen am 02. 03. 2015).

Schäfer, D. & Kett, J. (2013, 11. März). Culturegraph — Plattform für Wissensvernetzung. http://www.culturegraph.org/Subsites/culturegraph/SharedDocs/Downloads/Vortraege/BibTag2013CulturegraphWissensvernetzung.pdf?__blob=publicationFile (abgerufen am 10. 11. 2014).

Schnalke, M. (2014). Die Anfänge der digitalen Revolution: Der Einzug der Computertechnik in das wissenschaftliche Bibliothekswesen am Beispiel der baden-württembergischen Universitätsbibliotheken Konstanz und Ulm. *Perspektive Bibliothek*, *3*(1), 140–162. http://journals.ub.uni-heidelberg.de/index.php/bibliothek/article/view/14024 (abgerufen am 26. 04. 2015).

Tennant, R. (2002, 15. Okt.). MARC Must Die. *Library Journal*, *127*(17), 26. http://lj.libraryjournal.com/2002/10/ljarchives/marc-must-die/ (abgerufen am 02. 03. 2015).