Interpreting MARC: Where’s the Bibliographic Data?

The MARC data format was created early in the history of digital computers. In this article, the author entertains the notion that viewing MARC from a modern technological perspective leads to interpretive problems such as a confusion of “bibliographic data” with “catalog records.” He explores this idea through examining a specific MARC interpretation task that he undertook early in his career and then revisited nearly four years later. Revising the code that performed the task confronted him with his own misconceptions about MARC that were rooted in his worldview about what he thought “structured data” should be and helped him to place MARC in a more appropriate context.

by Jason Thomale

Introduction

The Machine Readable Cataloging (MARC) format was conceptualized in the early 1960s and first piloted in 1966 (Avram, 1975). It is now well over 40 years old. Considering just the advances in computer data representation that have happened since, today’s world is alien to the one in which MARC was conceived. In 1966, it would still be three years until Dr. Edgar F. Codd would publish his first paper describing a relational model for data as an IBM Research Report, and four years until he would revise the paper and publish it more widely in Communications for the ACM (Date, 1998); eight years until Donald Chamberlin and Raymond Boyce would first present their work on SEQUEL (SQL) (Chamberlin & Boyce, 1974); and ten years until Peter Chen would first propose the Entity Relationship model (Chen, 1976). We who work with library technology and systems cannot help but view MARC [1] through a lens colored by 44 years of rapid technological change (Figure 1). We now look at MARC based upon a worldview that is utterly different than the one that gave rise to the format; making anachronistic assumptions about how we might think it would work is all too easy. Of course we get poor results when we treat MARC data the way we would treat data in a relational database—MARC predates the earliest formal expression of relational data modeling concepts by three years!
Furthermore, the cataloging task, which generates the data that goes into a MARC record, seems enigmatic to most library technologists—and for good reason. Whereas a format built upon a modern data model would store data about bibliographic items directly [2], MARC actually stores bibliographic data by way of “cataloging record[s] ... or the information traditionally shown on a catalog card” (Furrie, 2009). From a modern systems design perspective, this seems odd. Our systems are designed to facilitate online search and retrieval of bibliographic information. Is the catalog card abstraction really the best carrier for that data in that context? But MARC was not invented to drive computerized information retrieval systems. Its original purpose was to automate the processes and tasks of a 1950s/60s technical services department—i.e., the creation and printing of catalog cards (Avram, 1975; Coyle, 2005; Coyle & Hillmann, 2007; Tennant, 2002). The same basic rules that determined how bibliographic data was to be stored and displayed on those catalog cards became the same rules that determined how data was to be formatted and stored in MARC records. In fact, though they have changed over time, the cataloging rules originated long before the advent of modern computer technology (Coyle & Hillmann, 2007; Taylor, 1999).

We can now perhaps more easily understand some of the contention we see within the current library metadata environment, which, on the surface, might seem baffling. Library catalogers and programmers clash (often passionately) about what constitutes “good library metadata.” On the one hand, many librarians consider traditional library cataloging not only to be useful, but vital for retrieval of online bibliographic information [3]. On the other, many programmers who try to work with catalog records come to conclude that they hinder online access and retrieval because they are so difficult to interpret algorithmically [4]. Indeed, I have witnessed and read conversations in which programmers and catalogers attempt to explain their points-of-view to one another. Too often both sides speak vaguely. Concrete examples—if offered—are explained in language that only one side or the other understands. Very little actual communication seems to
As a metadata librarian, I have a foot in both worlds: I must be able to read and interpret MARC data, but I must also understand systems, data design, and programming. My background in programming and database administration, however, preceded my training as a librarian. When I learned cataloging, my worldview was already set. Despite training and experience, I still find that I make automatic, mistaken assumptions about MARC. Some recent metadata clean-up work required me to revisit code that I had written much earlier in my career; it afforded me the opportunity to see firsthand some of the assumptions I made as a novice and the concrete results of those assumptions. Solving one problem in particular—cleaning up generic title metadata that had been derived from the MARC 245 field—helped me consciously to experience certain insights about working programmatically with MARC data that would have perhaps seemed counter-intuitive to me when I wrote the original code. Examining what led to these insights illustrates how programmers can think differently about MARC data to understand it better and develop methods to parse it more effectively. It also reveals evidence demonstrating the problems that traditional library cataloging poses in a modern systems context.

The Problem

Obtaining metadata for digitized items from the physical items’ MARC records is a common task. Three and a half years ago I developed a program that did such a thing for one of my institution’s first digital collections: recorded music that we stream online to support our School of Music’s teaching activities. Because this was one of our first digital collections and I was still new at working with MARC data, the resulting metadata suffered from a number of problems.

When I recently undertook a revision of my original code to fix these problems, one issue near the top of my “to-fix” list seemed simple on the surface. Title metadata (that is, titles for albums—not individual pieces) sometimes exhibited odd formatting quirks. Indeed, over the years I had fixed a number of records manually that simply looked weird, and more recently I had begun to notice titles that were both formatted oddly and seemingly incomplete.

Looking back at the code that I wrote three and a half years ago, I had to try to remember how I originally intended the title element to look. The data came from the MARC record’s 245 field. Although I could have simply followed cataloging conventions and used the entire title statement exactly as it appeared in the MARC record, the formatting seemed arcane, stilted, and jarring to the untrained eye. It would not serve our users well, I thought. In addition, the 245 tended to contain extraneous information that I did not think users would recognize as belonging to the title, such as the statement of responsibility. Removing the extraneous information required reformatting what remained. Ultimately, I wanted the metadata to appear as natural, simple, and readable as possible—based more on the rules of English grammar than the cataloging rules. When I wrote the first version of the title-extraction routine, I did not realize that it would have such a high failure rate, but I later noticed that, in roughly 25% of cases, odd formatting errors had crept into the output that thwarted my goal.

The Original Solution

The original algorithm was simple. I began with the 245$a (main part of the title),
which was all that was actually required. If the 245 contained a ¶b (remainder of the title—usually used for a subtitle) or a ¶n or ¶p (number/name of a part or section of a work), I added a colon and then the ¶b, ¶n, and/or ¶p, in that order, onto the title string. I decided to ignore the ¶c (statement of responsibility) because it was not actually part of the title, and I decided to ignore the ¶h because it tended just to contain “[sound recording].” Other subfields were not commonly used, so I ignored them.

Further, I recognized that any equals signs (=) in the data indicated words that were translations of what had been transcribed from the item. Such translations I thought would be important to keep but that the equal-sign notation would be jarring to end users, so I converted equal signs to commas. I also removed extraneous punctuation from the end of the string (such as forward slashes (/)). Below are three examples of titles that did not convert as intended.

```
=245 10#a3 symphonies#h[sound recording] ;+bThe rock = Der Fels
= Le rocher /+cSerge Rachmaninoff.
```

Came out as:

```
3 symphonies: The rock, Der Fels, Le rocher
```

By sheer coincidence, this formatting makes it look as though *The rock, Der Fels, and Le rocher* are the titles of the three symphonies. In reality, *The rock* is its own work separate from 3 symphonies whose title just happens to be translated into German (Der Fels) and French (Le rocher) in the MARC record.

```
=245 10#aFantasie in C major, op. 15#h[sound recording]
:+bWandererfantasie /+cFranz Schubert. Fantasie in C major,
| op. 17 / Robert Schumann.
```

Came out as:

```
Fantasie in C major, op. 15: Wandererfantasie
```

This time the formatting is not bad, but it completely misses the second work on the album (Schumann’s *Fantasie in C Major, op. 17*).

```
=245 10#aConcerto no. 1 for piano & orchestra, op. 15, C major#h[sound recording] =+bC-Dur = ut majeur ; Concerto no. 3 for piano & orchestra, op. 37, C minor = c-moll = ut mineur /
+cludwig
| van Beethoven.
```

Came out as:

```
Concerto no. 1 for piano & orchestra, op. 15, C major: C-Dur, ut majeur ; Concerto no. 3 for piano & orchestra, op. 37, C minor, c-moll, ut mineur
```

In this case, the resulting title is at least decipherable, but it does not follow my “natural, simple, and readable” criteria. Specifically, the punctuation seems inconsistent—even random.

### A Different Approach

My revision began with a systematic investigation about what I had done wrong. I noted some of the problem titles and examined those titles’ MARC 245s vis-à-vis
my original code. I asked myself: what was the problem? Had I made assumptions the first time that did not match how certain items’ MARC was structured? Did those items simply contain cataloging errors? If the mistakes were in the cataloging, were they regular and intelligible enough that I could code around them? If the mistakes were mine, how could I make sure that I was not simply writing revised code based on a new set of bad assumptions about the data?

Examinaing the MARC proved to be eye opening. The majority of errors appeared not to be in the cataloging, but rather in how I had interpreted it. Seeing my misinterpretations so clearly made me realize that I needed to change my entire approach to the problem. The three examples below illustrate my mistaken interpretations.

---

=245 1\ laMusique de chambre.\nVol. II =bChamber music, Vol. II =hKammermusik, Vol. II\n=cFauré.

=245 10\l=afantasien op. 116\n=cFantasias ; Intermezzi op. 117 ; Klavierstücke op. 118 & 119 = Pieces for piano / \n=cJohannes Brahms.


---

In my initial solution, I was treating each subfield as a discrete piece of data. I picked and chose the subfields I thought I needed and simply ignored the others. In many cases, however, the subfields I was ignoring—the $h$ and the $c$—actually contained important information. In the second example, above, the $h$ contains a translation-indicator (=) that tells me that the next subfield begins with a translation, and is probably not actually a subtitle; in the third example, the $c$ contains a lengthy list of titles of works that appear on the album in addition to what is already listed in the $a$.

My initial solution also disregarded the order in which subfields appeared. It assumed that it could impose a standard order ($tabnp$) regardless of what appeared in the data—indicating that I thought either that order does not matter or that subfields always appear in that particular order. In the first example, not only does a $n$ appear between a $a$ and a $b$, but the $n$ also contains an “=” immediately before the $b$. Taken out of order, the “=” would apply to something other than what is obviously intended—that “Chamber music” is a translation.

On a more general level, I realized that the contents of these 245s looked and acted less like data from a structured data record and more like textual markup from a document [6]. Structured data formats are intended for machine consumption and tend to follow rules that are simple and consistent—simpler rules make data easier to parse. Also, the more structured a data record is, the more explicit the semantics tend to be. Meaning is clear and encapsulated—the overall context in which data appears within a record is irrelevant because, apart from what might be specified in the data model, context carries no semantic
meaning. Interpreting a structured data record is no more complex than reading each data element and interpreting it according to that format's specifications. A document, however, is less straightforward, as the information it contains is meant more for human consumption; even if the document is marked up to aid machine-processing, the underlying structure is based on linguistics rather than a format that was designed to be machine-readable. The information in a marked-up document therefore behaves more like language than data. Meaning is not clear and encapsulated. The document as a whole contains semantic meaning beyond the sum of its marked-up data elements. Interpreting the document requires reliance on subtle ambiguities and context clues more than it does simple rules and specifications. Removing or refactoring data elements, unless done in a way that takes context into account, easily changes the meaning of the document as a whole. Looking at it from this perspective, I saw that the MARC 245s that I had been studying demonstrate the characteristics of marked-up documents. Each 245 field functions as a complete unit. The subfields help make meaning explicit by delineating the parts of that unit, but the context in which the subfields appear also carries meaning: if you remove a subfield without considering how it relates to the whole, or if you change the order of subfields without considering how they relate to one another, you can inadvertently change the meaning of the whole. The 245 has an implicit structure that is not defined by the subfields; rather it is defined by the cataloging construct known as the Title Statement. The subfields merely help demarcate some of the parts of that Title Statement.

Because my original solution to the problem had come from a structured-data perspective, its approach was decidedly rules-based. If I had maintained that approach throughout the revision process, my next step would have been to examine the MARC and cataloging documentation more closely to determine the precise rules upon which the contents of the 245s were supposed to be based. Then I would have developed methods for handling the subfields that followed these rules. I probably would have ended up with a complex series of loops and if/then/else statements that would have attempted to arrive at appropriate decisions based on which subfields appeared in what order and what punctuation they contained. The code would have become messy quickly.

But what if I treated the data more as it was now beginning to look to me: as textual markup? Looking at the problem this way revealed new possibilities. I began to think of a solution that would recognize and match the patterns inherent in the data instead of attempting to deconstruct and then reconstruct the data based upon the rules on which it had been built.

Finally, to ensure that I did not repeat my earlier mistake of coding to a data set that was too limited, I employed a more systematic workflow (Figure 2):

1. I created a script that would apply my title extraction routine to each 245 in the entire batch of MARC records for the collection and output the results in a text file for me to examine. I also created a script that would apply the same routine to a single title that I defined within the script (for testing and developing).

2. I ran the batch conversion script on the entire MARC record store. The script generated a plain text file containing each converted title.

3. I looked through the output to find the next title that did not convert correctly.

4. I looked up the 245 in a human-readable version of the MARC record store.
5. I used the new 245 as my test case and rewrote or tweaked my code until that 245 converted to an acceptable-looking title without breaking my previous test cases.

6. I repeated steps 2 through 5 until I could find no more titles in the batch output that looked incorrect.

Figure 2. Title-conversion-routine-revision workflow

**Toward a Revised Solution**

When I was ready to write my new algorithm, I began by browsing through my MARC record dump to see what consistent patterns I could find in the 245 data. I noticed several. Periods and semicolons seemed to delimit major components of the overall title—especially when titles of separate works were contained within the overall album title. Forward slashes delimited the beginning of a list of responsible parties, which always ended with a period. Equals signs indicated translations. Colons and commas indicated minor subdivisions within a component of a single work’s title. Brackets indicated information that the cataloger had included but had not transcribed directly from the physical item during cataloging. From these particular patterns, I could divide punctuation appearing in the title into three separate categories: punctuation that served as a major component separator (.;), punctuation that indicated information I wanted to remove (/=[]), and punctuation that served as a minor component separator (:).

I noticed a couple of complicating factors, however. First—and most seriously—periods were used not only as component delimiters but also to indicate abbreviations. Second, I noticed that there were actually cases in which bracketed information was desirable to keep in the final output.

The first problem was the more challenging. There seemed to be no convenient way to tell which periods were used for abbreviations and which were not. I solved the problem using brute force—I created a separate script that looped
through each 245 in the entire MARC record batch. It read any words that ended with a period, tallied them up, and generated a sorted list:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>op</td>
</tr>
<tr>
<td>573</td>
<td>no</td>
</tr>
<tr>
<td>251</td>
<td>Mozart</td>
</tr>
<tr>
<td>250</td>
<td>Bach</td>
</tr>
<tr>
<td>109</td>
<td>Beethoven</td>
</tr>
<tr>
<td>105</td>
<td>nos</td>
</tr>
<tr>
<td>95</td>
<td>Vol</td>
</tr>
<tr>
<td>94</td>
<td>K</td>
</tr>
<tr>
<td>83</td>
<td>Haydn</td>
</tr>
<tr>
<td>70</td>
<td>Mendelssohn</td>
</tr>
<tr>
<td>69</td>
<td>Vivaldi</td>
</tr>
<tr>
<td>67</td>
<td>Schubert</td>
</tr>
<tr>
<td>65</td>
<td>Schumann</td>
</tr>
<tr>
<td>65</td>
<td>Brahms</td>
</tr>
<tr>
<td>43</td>
<td>Handel</td>
</tr>
<tr>
<td>40</td>
<td>J</td>
</tr>
<tr>
<td>35</td>
<td>Chopin</td>
</tr>
<tr>
<td>34</td>
<td>D</td>
</tr>
<tr>
<td>30</td>
<td>Shostakovich</td>
</tr>
<tr>
<td>30</td>
<td>Prokofiev</td>
</tr>
<tr>
<td>29</td>
<td>al</td>
</tr>
<tr>
<td>28</td>
<td>No</td>
</tr>
<tr>
<td>28</td>
<td>Debussy</td>
</tr>
<tr>
<td>25</td>
<td>music</td>
</tr>
<tr>
<td>25</td>
<td>Ravel</td>
</tr>
</tbody>
</table>

It took little time to eyeball the list, extract what appeared to be abbreviations, and use those abbreviations as exceptions in my pattern-matching routines. I also excepted any initials (i.e., individual alphanumeric characters appearing as words unto themselves or sets of individual alphanumeric characters separated by periods).

The second complicating factor—the brackets—was much easier to rectify. In general, I noticed that brackets appearing in the 245 contained data that I wanted removed; brackets appearing in any other subfield contained data that I did not want to remove, though I still wanted to remove the brackets themselves.

My new code takes a basic two-step approach. First, it loops through each subfield contained within a 245 field and preprocesses the subfield data—it processes abbreviations; it generates an unambiguous delimiter character (the pipe ()) to replace the periods that serve as “major component” delimiters; and it preprocesses certain other punctuation marks based upon the subfield in which they fall, such as the brackets in the 245. Second, it processes the entire title as a single string—it matches patterns in the whole string that it would otherwise miss simply looking at each subfield in isolation; it determines data to be removed and removes it; it consolidates any repeated pipes into a single pipe character; and
finally it converts all pipes to semicolons and performs other formatting cleanup.

After a few iterations of coding, testing against a single test case, testing against the entire MARC dump, and then picking a new test case, I arrived at the final routine. The Perl code for the conversion routine is as follows:

```perl
01. sub version2_convert {
02.     my $field = shift; # $field is a MARC::Field object
03.     my $abbrevs = "op|no|nos|vol|vols|ai|etc|nr|st|posth|opp|jr|"
04.         ".dr|app|arr|orchestr|orch|ed";
05.     my $title;
06.     foreach my $subfield ($field->subfields()) {
07.         my $title_part = $subfield->[1];
08.         # convert all . to |
09.         $title_part =~ s/(\.)/|g;
10.         # convert | after $abbrevs back to .
11.         $title_part =~ s/(^\p{L}[^p{N}]+)(\$abbrevs)\$/\$1$2./ig;
12.         # convert | after initials back to .
13.         $title_part =~ s/(^\p{L}[^p{N}]+)(\p{L})\$/\$1$2./g;
14.         # convert . at the end of the string to .
15.         # (periods at the end of a subfield indicate end of a section)
16.         $title_part =~ s/(\|)(\p{L})\$/\$1$2./g;
17.         $title_part =~ s/(\.)\p{L})\$/\$1$2./g;
18.         $title_part =~ s/\./\./g;
19.     }
20.     # if this is a subfield h
21.     if ($subfield->[0] eq "h") {
22.         # make sure there's a space before any ending /
23.         $title_part =~ s/(\S)($1$2$3/;
24.     } else {
25.         # remove [ and ]
26.         $title_part =~ s/[\[\]]///g;
27.     }
28. }
29. 
30.     # if this is a subfield c and a / is missing from the title
31.     if ($subfield->[0] eq "c" && $title =~ /\$/) {
32.         $title .= "/";
33.     }
34.     # if this is a subfield n or p
35.     if ($subfield->[0] eq "n" || $subfield->[0] eq "p") {
36.         # convert a pipe at the end of the title string so far to |
37.         $title =~ s/\|\|/;/;
38.         # convert pipes within the subfield n or p to commas
39.         $title_part =~ s/\|/;/g;
40.         $title_part = "," . $title_part;
41.     }
42.     $title .= $title_part;
43. }
44. }
45. # remove ||| (comes from transformation of ...)
46. $title =~ s/\|\|\|//g;
47. # remove everything between = and /;:| or eos
48. $title =~ s/=,(.*)(\$\|;:|;\|:\|$|\|$2/g;
49. # remove everything between / and |
50. $title =~ s/\|$.*?\|\|$1/g;
51. # remove everything between []s
52. $title =~ s/\[.*\]|//g;
53. # convert ; to |
54. ```
Table 1 shows, for a selection of titles, a comparison between the MARC 245, the output of the original algorithm, and the output of the revised algorithm.

<table>
<thead>
<tr>
<th>MARC 245</th>
<th>Original Output</th>
<th>Revised Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>=245 \  \ta3 symphonies[\sound recording] ;\tiThe rock = Der Fels = Le rocher /\tcSerge Rachmaninoff.</td>
<td>3 symphonies: The rock, Der Fels, Le rocher</td>
<td>3 symphonies; The rock</td>
</tr>
<tr>
<td>=245 \  \taFantasie in C major, op. 15[\sound recording] ;\tiWandererfantasie /\tcFranz Schubert. Fantasie in C major, op. 17 / Robert Schumann.</td>
<td>Fantasie in C major, op. 15: Wandererfantasie</td>
<td>Fantasie in C major, op. 15: Wandererfantasie; Fantasie in C major, op. 17</td>
</tr>
<tr>
<td>=245 \  \taConcerto no. 1 for piano &amp; orchestra, op. 15, C major[\sound recording] =\tiC-Dur = ut majeur ; Concerto no. 3 for piano &amp; orchestra, op. 37, C minor = c-moll = ut mineur / \tcLudwig van Beethoven.</td>
<td>Concerto no. 1 for piano &amp; orchestra, op. 15, C major: C-Dur, ut majeur ; Concerto no. 3 for piano &amp; orchestra, op. 37, C minor, c-moll, ut mineur</td>
<td>Concerto no. 1 for piano &amp; orchestra, op. 15, C major; Concerto no. 3 for piano &amp; orchestra, op. 37, C minor</td>
</tr>
<tr>
<td>=245 \  \taWeihnachtsoratorium[\sound recording] ; BWV 248.\tnKantate 1-3 =\tiChristmas oratorio / \tcJohann Sebastian Bach.</td>
<td>Weihnachtsoratorium; Kantate 1-3, Christmas oratorio</td>
<td>Weihnachtsoratorium; BWV 248, Kantate 1-3</td>
</tr>
<tr>
<td>=245 \  \taFantasien op. 116[\sound recording] ;\tiIntermezzi op. 117 ; Klavierstücke op. 118 &amp; 119 = Pieces for piano /\tcJohannes Brahms.</td>
<td>Fantasien op. 116: Fantasias ; Intermezzi op. 117 ; Klavierstücke op. 118 &amp; 119, Pieces for piano</td>
<td>Fantasien op. 116; Intermezzi op. 117; Klavierstücke op. 118 &amp; 119</td>
</tr>
<tr>
<td>=245 \  \taOverture to Candide[\sound recording] / \tcLeonard Bernstein ; [arr.] Grundman. George Washington Bridge / William Schuman. An outdoor overture / Aaron Copland. El Salon Mexico / Aaron Copland ; [arr.] Hindsley. Chester / William Schuman. La fies</td>
<td>Overture to Candide</td>
<td>Overture to Candide; George Washington Bridge; An outdoor overture; El Salon Mexico; Chester; La fies mexicana</td>
</tr>
<tr>
<td>MARC 245</td>
<td>Original Algorithm</td>
<td>Revised Algorithm</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Violin concerto no. 2: Symphony no. 4: Landscape; Let us now praise famous men; Elegy / Samuel Jones.</td>
<td>Symphony no. 3: The tricentennial / Lester Trumble.</td>
<td>The ballad of Baby Doe: opera in two acts / Pentimento</td>
</tr>
<tr>
<td>&quot;Pentimento&quot; [sound recording] / Ezra Laderman. Symphony no. 3: The tricentennial / Lester Trumble.</td>
<td>The protecting veil / for solo cello / Tavener</td>
<td>The protecting veil; Third suite for cello, op. 87; Thrinos: for solo cello</td>
</tr>
<tr>
<td>Orchesterstücke, Op. 12 Sz51 Orchestral pieces, Pièces pour orchestre; Konzert für orchestra, Sz116, Concerto for orchestra = Concerto pour orchestre / Béla Bartók.</td>
<td>String quartets op. 51; String quartet op. 67, &quot;American quartet&quot; op. 96</td>
<td>String quartets op. 51; String quartet op. 67, &quot;American quartet&quot; op. 96</td>
</tr>
<tr>
<td>&quot;String quartets op. 51&quot; [sound recording] / Johannes Brahms. &quot;American quartet&quot; op. 96 / Antonin Dvořák.</td>
<td>Partita no. 1, BWV 825; Englische suite no. 3, BWV 808; English suite, Suite anglaise; Französische suite no. 2, BWV 813; French suite, Suite française / Johann Sebastian Bach.</td>
<td>Partita no. 1, BWV 825; Englische suite no. 3, BWV 808; English suite, Suite anglaise; Französische suite no. 2, BWV 813; French suite, Suite française</td>
</tr>
<tr>
<td>Partita no. 1, BWV 825; Englische suite no. 3, BWV 808; English suite, Suite anglaise; Französische suite no. 2, BWV 813; French suite, Suite française / Johann Sebastian Bach.</td>
<td>Partita no. 1, BWV 825; Englische suite no. 3, BWV 808; English suite, Suite anglaise; Französische suite no. 2, BWV 813; French suite, Suite française</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison between MARC 245, output of the original algorithm, and output of the revised algorithm.

**Analysis and Conclusion**

The revision process of my MARC title-extracting routines revolved around a significant fact and a significant insight. The fact: MARC is, at its heart, a data format built to contain catalog records; bibliographic items are described via the catalog records rather than directly via the structured MARC data. Understanding and internalizing this leads us to the insight: if we think of those catalog records as structured documents rather than as data records, then MARC has as much in common with a textual markup language (such as SGML or HTML) as it does
with what we might consider to be "structured data."

Thinking broadly, the fact that the MARC format focuses on catalog records rather than bibliographic items lends credibility to the idea that MARC has been increasingly relevant to library systems since the demise of the card catalog. Many in the library profession recognize that improving library tools requires the development of data formats that better support system functionality that modern library users need and expect. Extensive efforts to rectify the situation have been underway for over a decade, such as creating new models for bibliographic data, updating cataloging rules, and attempting to convert library data to new formats [7]. However, much bibliographic data is still locked away in MARC. Programmers who must write the code to interpret MARC have the daunting task of trying to understand a data format that is inherently alien to them. The aforementioned fact and insight can, perhaps, help make MARC seem slightly less alien to the programming mind—specifically upon examining the implications.

First, a MARC record does contain an explicit structure. It contains fields, subfields, and indicators, and our tools for processing MARC give us the ability to extract discrete, granular chunks of data just as we would from, e.g., a database. Concrete rules and semantics define the data that goes into a MARC record and dictate the use of the fields and subfields. When we must interpret, extract, or otherwise act upon the data, the documented rules and semantics can help guide our interpretation.

Second, data within a single MARC field often behaves like document markup. Unlike data occurring in a database, data expressed through document markup gains some degree of meaning based upon its context within the document and its relation to other data within the document. In MARC, punctuation that appears in one subfield can subtly change the meaning of data in a different subfield; changing subfield order can also subtly change the data's meaning.

Third, extracting bibliographic data from MARC is often not a simple matter just of interpreting the explicit structural information. In many cases—such as the case of the 245 field—we must recognize the existence of both an explicit and an implicit structure. MARC subfields comprise the explicit structure; cataloging rules define the implicit. The two are intertwined, and both must be interpreted. What appear to programmers as cataloging artifacts—odd patterns of punctuation and formatting within the data that are often dismissed as merely presentational—actually carry semantic meaning that is lost if the patterns are ignored. If we want to pull a book’s title out of the 245, we must understand that doing so requires us to interpret the "Title Statement" cataloging edifice—otherwise our methods will not work well.

This need to decipher data built upon cataloging rules programmatically is perhaps the biggest technical hurdle in the quest to convert library data into more machine-readable formats en masse. Cataloging has a long, complex history [8]. Multiple cataloging standards—or codes—have been developed throughout the years. Standards used concurrently with MARC—such as the Anglo-American Cataloguing Rules (AACR & AACR2)—have continued to evolve. Because MARC contains cataloging but does not enforce the use of any single cataloging code, it may not be possible to pinpoint exactly what set of cataloging rules any given record is based upon [9].

Furthermore, the act of cataloging—no matter the set of rules used—relies on human beings to enter data logically and consistently. Because modern
cataloging rules are complex, any MARC record set will naturally contain some
effects that could cause a program to misinterpret the data. The “cataloging” in
any given set of MARC records is therefore a moving target.

In my particular example, the records that I used had been cataloged mostly
according to some version of the AACR2 standard [10], which somewhat
mitigated the effects of this problem. Even so, I believe that the approach I have
described offers a potential avenue for handling record sets where this is not the
case. Because the approach is one of reverse engineering based on bottom-up,
pattern-matching techniques, it should allow for greater flexibility in
accommodating inconsistencies (e.g., when the cataloging does not follow the
cataloging rules). It also should require no knowledge of exactly which cataloging
standards were used in creating the MARC data; in theory, it only requires a
programmer to be able to recognize the sets of consistent patterns and variations
on those patterns that appear within the data. Future work would need to test
these hypotheses on record sets that exhibit increasingly more complex and
varied cataloging in order to help determine how to refine and adapt the
approach. Although outside my direct realm of expertise, exploring the use of
machine-learning algorithms and data analytics to help with pattern identification
would be another logical pathway for developing the approach further.

References

Library of Congress.

British Broadcasting Corporation. (2008, August 5). The accelerator of the

Coyle, K. (2005). Catalogs, card—And other anachronisms. The Journal of
Academic Librarianship 31(1), 60-62.

Cataloging rules for the 20th century. D-Lib Magazine (13)1/2. Retrieved from:
http://dlib.org/dlib/january07/coyle/01coyle.html

language [Electronic version]. Proceedings of the 1974 ACM SIGFIDET
Workshop on Data Description, Access, and Control, 249-264. New York, NY:
Association for Computing Machinery. Retrieved from:

Chen, P. P. (1976, March). The Entity-Relationship model—Toward a unified
view of data [Electronic version]. ACM Transactions on Database Systems,


Furrie, B., & Database Development Department of the Follett Software
http://www.loc.gov/marc/umb/


Notes

[1] Just a quick note of clarification: throughout the paper, I specifically have in mind the MARC 21 format for bibliographic data, even though I refer to it only as “MARC”.

[2] The Functional Requirements for Bibliographic Records data model is a perfect example of a “modern data model” for bibliographic data. This model was developed in the late 1990s. See http://www.ifla.org/en/publications/functional-requirements-for-bibliographic-records for more information.

[3] This sentiment is so widespread in the library profession that any citations I could provide to back up this statement would be both woefully incomplete and largely unnecessary—any cataloger would validate my assertion. You can, however, view a good illustration of this point—for example—in the comments following the Language Log blog posting about the “Metadata Train Wreck” that is Google Books: http://languagelog.ldc.upenn.edu/nll/?p=1701.

[4] As with the preceding statement about librarians’ attitudes toward cataloging, I cannot really give citations to back up this claim with 100% certainty and objectivity—as far as I can tell, programmers’ attitudes toward cataloging have never been studied. I can give examples, however. Roy Tennant’s 2002 piece entitled “MARC Must Die” (referenced elsewhere in this paper) provides a good look at some of MARC’s problems from a modern technological viewpoint.


There is also the statement about working with MARC data purportedly made by Google engineer Leonid Taycher that “the first thing he had to learn was that the ‘Machine Readable’ part of the MARC acronym was a lie” (from http://go-to-hellman.blogspot.com/2010/01/google-exposes-book-metadata-privates.html).

Finally, the NGC4lib, or Next Generation Cataloging for Libraries, listserv (view
archives at http://serials.informotions.com/ngc4lib/) provides a wealth of documented discussions between programmers and catalogers about the utility of cataloging data.

[5] Clearly this is my opinion and is based mainly on anecdotal evidence. I again submit that the NGC4lib archives contain numerous examples backing up my point that catalogers and programmers often speak past one another.

[6] Within the XML community exists an analogous concept—that of “document-centric” versus “data-centric” XML. (For a concise explanation, see http://techessence.info/node/51.) Thinking of the document/data division as a continuum rather than a binary either/or (and applying it beyond just the XML format), MARC would fall somewhere in the middle. It is fielded and thus behaves like data, but the contents of many of those fields behave more like documents (where the subfields provide mark-up).

[7] FRBR is the primary example of a “new” model for bibliographic data. The work happening on Resource Description and Access (RDA) is the primary example of cataloging rules being updated. There have been a number of recent efforts to publish library cataloging data in linked data formats on the Web: for example, http://www.viaf.org and http://id.loc.gov/.

[8] Chapter three (pages 37-52) of The Organization of Information by Arlene G. Taylor contains a succinct chronology of the historical developments that lead to modern cataloging. Cataloging as we know it is largely a product of the 19th and early 20th centuries.

[9] Position 18 of the MARC leader contains a code that can help you determine what descriptive cataloging standard was used in the creation of the record. It can indicate AACR2, International Standard Bibliographic Description (ISBD), non-ISBD, or an unknown cataloging standard. Still, knowing you have a “non-ISBD” or “unknown” standard is not terribly helpful, and it does not tell you what version of AACR2 or ISBD were used.

[10] Because the records were cataloged according to AACR2, the formatting of the MARC 245s (e.g., the punctuation) conformed to ISBD. Although the approach that I took did not require it, I could have consulted the ISBD documentation to help me make sense of the data.