code{4}lib JOURNAL

Issue 26, 2014-10-21 ISSN 1940-5758

Parsing and Matching Dates in VIAF

The Virtual International Authority File (OCLC Online Computer Library Center 2013) http://viaf.org is built from dozens of authority files with tens of millions of names in more than 150 million authority and bibliographic records expressed in multiple languages, scripts and formats. One of the main tasks in VIAF is to bring together personal names which may have various dates associated with them, such as birth, death or when they were active. These dates can be quite complicated with ranges, approximations, BCE dates, different scripts, and even different calendars. Analysis of the nearly 400,000 unique date strings in VIAF led us to a parsing technique that relies on only a few basic patterns for them. Our goal is to correctly interpret at least 99% of all the dates we find in each of VIAF's authority files and to use the dates to facilitate matches between authority records.

Python source code for the process described here is available at https://github.com/OCLC-Developer-Network/viaf-dates.

by Jenny A. Toves and Thomas B. Hickey

Background

VIAF merges records from various national and trans-national authority files into clusters. These authority files provide identifiers and often standard names for people, titles, series and concepts. There are many challenges in matching across the files, including recognizing and parsing dates associated with personal names. Bibliographic dates can be surprisingly complicated and over the years libraries have come up with a mélange of approaches to capture the aspects of dates that they are interested in.[1][2] Dates can appear in a number of different fields, including name headings, separate date fields, and within notes. VIAF needs to recognize the presence of dates and convert them into a uniform format that it can use when matching and comparing records about people.

How VIAF Uses Dates

While contemporary files such as ORCID (http://orcid.org/) place great reliance on email addresses and institutional associations, library files have in general eschewed such information and rely heavily on birth and death dates to disambiguate people. VIAF tries to match records that describe the same entity in multiple files, and dates associated with people are important pieces of information when matching names across authority files. Each VIAF cluster that represents a person, maintains a date range (described below in Storing and Comparing Dates). If VIAF can identify that two names are associated in some way (for example the forms are similar or have matching cross references) and share both a birth and death year, this is considered a very strong indication of a match, second only to more specific information such as matching unique identifiers. While on the face of it matching dates would seem straight forward, there are some subtleties even after the dates have been parsed.

Most of the dates VIAF needs are specifically coded as dates, so recognizing them is not difficult. However VIAF also needs to pick dates out of unstructured text, such as notes in authority records. For this we use some fairly simple regular expressions [3] to recognize and parse these dates. Appendix 3 has examples of these.

Here are examples of some of the main variants we see:

- Ranges. Typically used to indicate birth and death
  - 1903-1993

- Flourished dates (when the person was active). May or may not be a range.
  - 1130 fl.
- Circa dates. Approximate dates
  - ca. 1507-1584
- Other approximations
  - 4./5. Stol.
  - 10..-11..
- Showing month and day
  - 1942 June 24-
  - 1943 ún. 23.-?
- Different scripts
  - ٢٧٣م
- Different calendars
  - 604-672
- Of course, these can be combined in various ways
  - 14..-1472
  - 14??-
  - –ca 1475

Here are some ways we see December 3, 1949 entered:

- 1949 (December 3)-
- 1949 Dec. 3-
- 1949 3 déc.-
- 1949 December 3-
- 1949 (Dec. 3)-
- 03.12.1949-

We find 27,000 ways to say what we interpret as flourished around 1850. Here are the patterns that occur more than a thousand times (the first character is the MARC subfield code):

- f18..-19..
- d18??-

- f18..-18..
- d18..-19..
- f18..-18..?
- f17..-18..
- f18..-18..
- d18..-18..,
- d18..-18..
- d18..-….
- fca 18–
- d17..-18..
- f18..-19..?
- f17..-18..?
- f18..-19..
- f17..-18..

Although our previous system was able to share much of the code used to parse dates from various authority files we became aware that we had no way to make global changes in how dates were handled. One change we wanted to make was how we handled flourished dates, and realized we needed to touch the dozens of routines that handle the details of each authority file.

An interesting observation is that while the domain, that is how dates are represented, is almost limitless, the range of actual individual dates representing a single day is fairly small. Given two thousand years of publication there are only about 730,000 different dates of interest (2000 years x 365 days/year). Since most dates we encounter are after 1700, only about 115,000 dates will be encountered with any regularity and the range of the transformation from text to our stored dates is fairly small. Another observation is that while we characterized the domain as 'almost limitless', in fact, within date subfields we find 384,000 actual unique date strings in all the VIAF authority source authority records, even before normalization (this is larger than the 115,000 individual dates both because of variant representations and because many of these strings represent a date range). A table [4] of all these date strings, ranked by frequency is available on GitHub.

Identifying dates in authority and bibliographic records

VIAF is interested in various dates in both authority and bibliographic data. We tend to divide dates into four categories

- Publication dates
- Personal dates in a field with a specified date format
- Personal dates in free text notes
- Personal dates in a heading

The first two of these are fairly straight forward. While publication dates can be complicated, this data is considered of low relevance for matching names so we simplify it by just using the equivalent of the MARC-21 Date1 field.[5] This is a coded field, but even that can have a number of variant forms, such as 198U to indicate an approximate date in the 1980's. For the purpose of VIAF we avoid publication dates that are indicated as questionable or are date ranges and

discard dates that are not all digits, along with a few patterns, such as '0000' and '9999', which do not look like valid dates.

When a date is in a field such as the MARC-21 046 [6], parsing is also quite straight-forward since the ISO 8601 standard on the Representation of Dates and times [7] is specified.

Picking up dates from free text notes fields can be a challenge, but prior to 2009, MARC-21 authority records did not have a structured way of entering date, month and days associated with people, so this information is often embedded in notes and needs to be found. With the many languages and scripts in VIAF source authority records, identifying the type of date in free text can be difficult, e.g. whether it is a publication date, birth, death, or some other type of date. Again, we simplify the process by looking for key words and then picking up only the most straightforward of the dates.

While all these dates are important we have put the most effort into parsing the dates contained in the personal headings themselves. A typical heading looks like:

```
‡a  Shakespeare, William, ‡d  1564-1616

‡a  Shakespeare ‡b  William ‡f  1564-1616
```

Where the ‡d (MARC-21) and ‡f (UNIMARC) indicate the date subfield with a hyphen indicating a birth-death range. Our approach to parsing these dates is the topic of the rest of the paper.

Storing and Comparing Dates

Each cluster in VIAF representing a person has a date range stored as two dates (min and max) consisting of a year, month and day. Associated with the date range is an indication of what they represent. We have three values:

- Lived – The dates are birth and/or death dates
- Flourished – The dates show when the person was active
- Circa – The dates are approximate

Each section of the date (Year-Month-Day) is set to 0 if not available. The dates themselves are kept in the standard Western calendar (often called the Gregorian calendar), using negative dates for dates before 1 A.D./C.E., turning 1 B.C.E. into -1. While best practice when using negative numbers for B.C.E. dates is to use astronomical conventions [8] [1] which use a year 0, we do not, maintaining consistency with the common practice of not having a year 0 in B.C.E dates and reserving 0 to indicate no data.

Our most basic test for comparing date ranges is to see whether two sets of date ranges are compatible:

- If the both the min and max years are zero in both dates, there is no conflict
- For min dates after 1400:
  - One max date cannot be less than the other min date
  - Max dates must be within 120 years of the other min date when present
- If either pair is marked Flourished
  - Min and max dates must be within 100 years of each other if present
- If either pair is marked Circa
  - Min and max dates must be within 10 years of each other if present

- Otherwise both are Lived
    - Min and max dates must be within 3 years of each other if present
    - If both have months specified, they must match
    - If both have days specified, they must match

The 3 year range was arrived at pragmatically (we started at 0 and worked up). For various reasons dates associated with people tend to vary by a few years, and allowing some variation in them helps bring records together.

When comparing dates within a single authority file (e.g. to look for duplicates) the fully parsed dates essentially have to match exactly to not conflict.

When only one of the min/max dates is present for comparing we get a single-date match which is much weaker than the earlier mentioned double-date match.

Mapping Dates to Patterns

Our original approach grew from parsing dates in two authority files to eventually three dozen. Although our processing for each file shared code, each had its own dedicated parsing routines. Driven by necessity, we evolved the current system that relies on mapping dates to patterns shared across all authority files that are then used to guide the parsing.

Here are some of the pieces we need to identify:

- Type of date: flourished, circa, birth/death dates
- Era, e.g. A.D., B.C., B.C.E for Gregorian dates
- Reign and era dates for Japanese names
- Century indications
- Months
- Alternate calendars, e.g. Hijri (Arabic) dates
- Wild card characters
- Separating the min and max dates (e.g. birth and death)

Parsing a date subfield is accomplished by morphing the input data into a pattern that guides parsing. There are four basic steps:

- Normalization of the input data
- Splitting the input data into min (e.g. birth) date and max (e.g. death) date sections
    - If we have a Han (Japanese) date then resolve it via lookup table and we are done[9]
    - If all parts of the date are century dates then resolve them and we are done
- Resolving any wildcards
- Parsing the min/max date sections

As an example, “1949 December 3-“ is converted to the pattern “NNNN month NN”. This pattern is then matched to this regular expression:

```
NNNNmonthNN = re.compile('(\d\d\d\d) +([^0-9. ]*)\.? ?(\d\d?)')
```

Which is then used to pick out the year, month and day from the input string. Appendix 4 shows the 32 recognized patterns, and the code for all the patterns can be found in dateFld.py.[10]

Normalization starts with lowercasing the data and decomposing Unicode characters. Next some non-Latin digits and punctuation characters such as hyphens and question marks are converted to Latin equivalents. Digits are then converted to “N” and anything that looks like a month name is converted to “month”. We have built these tables up from examples encountered in dates, augmented by month names and abbreviations for various locales known to the Python date libraries. Here are some examples of the transformation:

- 1999 -> NNNN
- 1999 January 10 -> NNNN month NN
- 1947-1999 -> NNNN-NNNN
- 30 B.C. -> NN bc
- Circa 1920 -> circa NNNN

Splitting tries to find a single hyphen and use that point to split the date into min/max sections. If no hyphen exists, then look for evidence of this being a death date (ie. “d 1946”). If there is no hyphen and no evidence of a death date then it is assumed to be a birth date. If there are multiple hyphens, then we have a manually constructed table of 75 common patterns[11] that can be split. An example is “NNNN-NNNN-“. The second hyphen is probably a typo so we ignore it and split the date at the first hyphen. Here are some examples of the transformation:

- NNNN-NNNN -> (NNNN, NNNN)
- died NNNN -> (None, NNNN)
- NNth century-NNNN -> (NNth century, NNNN)
- NNNN -> (NNNN, None)

Resolving wildcards means finding things like “NNN?”, “NNN.” and “NNNN or N” and modifying the pattern, the input string and the datetype. An example is the string “NNN?” in a birth pattern with a date string of “197?”. The input data is changed to ‘1979’. The pattern is changed to ‘NNNN’ and the date type is set to ‘circa’. ‘circa’ allows ±10 years to match so this birth date will match any other birth date from 1969 to 1989. There are some patterns that we are unable to parse, usually because they are ambiguous. Appendix 1 has a list of these patterns.

Post parsing we apply Hijri and BC adjustments. Arabic dates are converted to their Latin form. If both min and max dates are BC (or if only the max date is BC), multiply min and max by -1.

The sample VIAF code in GitHub has a stub for the Hijri code. Since it has its own licenses it needs to be obtained separately.[12]

During this transformation, the dates undergo some ‘sanity’ checks, the most important being:

- Max >= min date
- Max date – min date <= 110 years
- Month values must be between 01 and 12 inclusive
- Day values must be appropriate for the given month

Testing

When we receive a new authority file to add to VIAF, one of our first tasks is to identify the dates in the records and decide how to parse them. To do this we extract all the dates found in the structured fields, count them and rank them. These strings can then be put through our existing date parsing routines and then visually compared to the input. Since we have the dates ranked by frequency we concentrate on the dates we will encounter the most, making sure that the interpretation looks correct and that we are handling at least 99% of the dates correctly. The date patterns tend to be clustered, so this task is not as difficult as it might seem.

Since we now have a uniform approach to dates across the various authority files that contribute to VIAF, we can also consolidate all the dates we have found across all of them. This lets us spot patterns that, while fairly rare in individual files, across all of VIAF are large enough to merit attention. Appendix 2 shows the most common patterns along with an indication of how often they are encountered.

Limitations

We recently added Japanese 'reign' dates to VIAF. While there are many other forms of dates in the world [13] [14], we are comfortable that we are covering the vast majority of dates we encounter.

Works Cited

[1] Library of Congress. "Extended Date/Time Format (EDTF) 1.0 Submission." Library of Congress. January 13, 2012. http://www.loc.gov/standards/datetime/pre-submission.html (accessed 11 11, 2013).

[2] Dublin Core Metadata Initiative. DCMI Date Working Group. http://dublincore.org/groups/date (accessed 04 24, 2013).

[3] Wikimedia. "Regular expression." English Wikipedia. http://en.wikipedia.org/wiki/Regular_expression (accessed 11 11, 2013).

[4] https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/datefields.txt

[5] Network Development and MARC Standards. "MARC 21 Format for Bibliographic Data: 008:." Network Development and MARC Standards. 09 01, 2011. http://www.loc.gov/marc/bibliographic/bd008a.html (accessed 04 26, 2013).

[6] Network Development and MARC Standards Office, Library of Congress. "MARC 21 Format for Authority Data: 046: Special Coded Dates." 04 05, 2011. http://www.loc.gov/marc/changes-rda-046.html (accessed 05 25, 2013).

[7] WikiMedia. "ISO_8601." Wikipedia. 04 24, 2013. http://en.wikipedia.org/wiki/ISO_8601 (accessed 04 26, 2013).

[8] Espenak, Fred. "Year Dating Conventions." NASA Eclipse Web Site. February 25, 2008. http://eclipse.gsfc.nasa.gov/SEhelp/dates.html (accessed 02 10, 2014).

[9] https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/handates.py

[10] https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/dateFld.py

[11] https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/overrides.py

[12] Alsadi<alsadi@gmail.com>, Muayyad Saleh. "hijra/hijra.py." http://github.com/ojuba-org. 2006-2008. https://github.com/ojuba-org/hijra/blob/master/hijra.py (accessed 08 26, 2014).

[13] Reingold, Edward M., and Nachum Dershowitz. Calendrical calculations. Cambridge, UK: Cambridge University Press, 2001.

[14] Blinn, Peter. "Today's date in over 400 more-or-less obscure foreign languages." Curious Notions. 2014. http://www.curiousnotions.com/todays-date.asp (accessed 03 18, 2014).

About the Authors

Jenny Toves is a Software Architect in OCLC Research. In addition to VIAF, her current interests include the large scale FRBR clustering of bibliographic records, which is central to OCLC's linked data program. One aspect of this work is the algorithmic creation of authority records by mining the bibliographic data. She has an undergraduate degree in Computer Information Systems and a M.S. in Computer Science.

Thomas Hickey is Chief Scientist at OCLC where he helped found OCLC Research. Current interests include metadata creation and editing systems, authority control, parallel systems for bibliographic processing, and information retrieval and display. In addition to implementing VIAF, his group looks into exploring Web access to metadata, identification of FRBR works and expressions in WorldCat, the algorithmic creation of authorities, and the characterization of collections. He has an undergraduate degree in Physics and a Ph.D. in Library and Information Science.

Appendix 1

Unhandled date patterns with multiple hyphens.

These patterns are not handled because the meaning is ambiguous or because we haven't seen it before. This data is monitored to look for new overrides to add to overrides.py (https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/overrides.py).

UNHANDLED	ca NN- -	2330
UNHANDLED	ca NN – -	110
UNHANDLED	NN- -	69
UNHANDLED	NN-..-	38
UNHANDLED	N–	36
UNHANDLED	ca NN –	34
UNHANDLED	NN-ה המאה/NN-ה	30
UNHANDLED	-caNNN of -caNNN	17
UNHANDLED	NN – -	17
UNHANDLED	-NNN or -NNN	16

Appendix 2

This table summarizes the 40 most common date patterns covering 99.4% of the dates VIAF encounters. See https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/appendices.py for code to generate this table.

Pattern	Occurences	Example	Parsed date	Date type	% of total
NNNN	19563491	d1947-	[1947, '', '']	lived	94.19%
.	372270	d(1947-).	[0, '', '']	lived	95.99%
NNN	179488	d900-talet	[900, '', '']	lived	96.85%
NNNN?	145978	f1950?-….	[1950, '', '']	circa	97.55%
NNth century	126436	d20th century	[1900, '', '']	flourished	98.16%
ca. NN. jh.	47126	dca. 20. Jh.	[1900, '', '']	flourished	98.39%
…	43903	d1977-…	[0, '', '']	lived	98.60%
ca. NN./NN. jh.	17848	dca. 20./21. Jh.	[1900, '', '']	flourished	98.69%
NNth cent.	16349	d17th cent.	[1600, '', '']	flourished	98.76%

Pattern	Occurences	Example	Parsed date	Date type	% of total
NN	15008	d19-….	[19, '', '']	lived	98.84%
ca.	11056	dca. Gegenwart	[0, '', '']	circa	98.89%
NNNN month NN	10839	d1921 October 30-	[1921, '10', '30']	lived	98.94%
?	10562	d?-….	[0, '', '']	lived	98.99%
NN. jh.	8852	d20. Jh.	[1900, '', '']	flourished	99.04%
NN./NN. jh.	6758	d20./21. Jh.	[1900, '', '']	flourished	99.07%
NNth cent	6620	d19th cent	[1800, '', '']	flourished	99.10%
N…	6342	d18..-1…	[1850, '', '']	flourished	99.13%
ca. N. jh.	5796	dca. 6. Jh.	[500, '', '']	flourished	99.16%
NNe e.	5136	d18e E.	[1700, '', '']	flourished	99.18%
NNNN month N	4886	d1956 November 7-	[1956, '11', '07']	lived	99.21%
ca. N. h. NN. jh.	4783	dca. 2. H. 20. Jh.	[1900, '', '']	flourished	99.23%
ca. NN.jh.	4016	dca. 20.Jh.	[1900, '', '']	flourished	99.25%
NN. stol.	3964	d19. stol.	[1800, '', '']	flourished	99.27%
Nth century	3787	dactive 9th century	[800, '', '']	flourished	99.29%
NN.NN.NNNN	3437	d09.06.1703-	[1703, '06', '09']	lived	99.30%
N. jh. v. chr.	3213	d3. Jh. v. Chr.	[-300, '', '']	flourished	99.32%
?.	2836	d(1892-?).	[0, '', '']	lived	99.33%
NNe eeuw	2722	d18e eeuw	[1700, '', '']	flourished	99.34%
N. jh. n. chr.	2563	d5. Jh. n. Chr.	[400, '', '']	flourished	99.36%
ca NN- -	2330	fca 18- -	[0, '', '']	circa	99.37%
ca. N. hälfte NN. jh.	2250	dca. 2. Hälfte 17. Jh.	[1600, '', '']	flourished	99.38%
sec. xvi	2225	dsec. XVI	[1500, '', '']	flourished	99.39%
ca. ende NN. jh./anfang NN. jh.	2105	dca. Ende 20. Jh./Anfang 21. Jh.	[1900, '', '']	flourished	99.40%
NNth/NNth cent.	2041	d17th/18th cent.	[1600, '', '']	flourished	99.41%
sec. xvii	1904	dsec. XVII	[1600, '', '']	flourished	99.42%
N	1813	d1-1	[0, '', '']	lived	99.43%
NNth/NNth cent	1797	d17th/18th cent	[1600, '', '']	flourished	99.44%
ca. N./N. jh.	1729	dca. 5./6. Jh.	[400, '', '']	flourished	99.44%
NNNN ?	1636	f1577 ?-1650	[1577, '', '']	circa	99.45%
ca. NN./NN.jh.	1415	dca. 20./21.Jh.	[1900, '', '']	flourished	99.46%

Appendix 3

Regular expressions used to find birth and death dates in free text note fields in MARC authority records.

```
## DD. MM. YYYY.

# ('', 'narozen', '31', '01', '1962')

birth3 = re.compile('(^| )(narozen) (\d{1,2}) ?(\d{1,2}) ?(\d{4})')

death3 = re.compile('(^| )(zemrel) (\d{1,2}) ?(\d{1,2}) ?(\d{4})')
```

```
 8
 9  # dec 21, 1903
10
11  # ('', 'b.', 'jan 31 1962', 'jan ', '31 ', '1962')
12
13  birth1a = re.compile('(^| |\()(b\.|n\.) ((\D{3,9} )?([0-9]{1,2},? )?([0-9]{4}))')
14
15  death1a = re.compile('(^| |\()(m\.|d\.) ((\D{3,9} )?([0-9]{1,2},? )?([0-9]{4}))')
16
17  birth1b = re.compile('(^| )(born|ne|nee|birth|dob) (([a-z]{3,9} )?([0-9]{1,2},? )?([0-9]{4
18
19  death1b = re.compile('(^| )(died) (([a-z]{3,9} )?([0-9]{1,2},? )?([0-9]{4}))')
20
21  # 21 dec, 1903
22
23  # ('', 'b.', '31 jan 1962', '31', 'jan', '1962')
24
25  birth2a = re.compile('(^| |\()(b\.|n\.) (([0-9]{1,2}) (\D{3,9}) ([0-9]{4}))')
26
27  death2a = re.compile('(^| |\()(m\.|d\.) (([0-9]{1,2}) (\D{3,9}) ([0-9]{4}))')
28
29  birth2b = re.compile('(^| )(born|ne|nee|birth|dob) (([0-9]{1,2}) ([a-z]{3,9}) ([0-9]{4}))'
30
31  death2b = re.compile('(^| )(died) (([0-9]{1,2}) ([a-z]{3,9}) ([0-9]{4}))')
```

Appendix 4

There are 32 known patterns which map to 10 regular expressions that are used to parse the dates. The month table (monthLookup in https://github.com/OCLC-Developer-Network/viaf-dates/blob/master/dateFld.py) is used to convert month names to 01-12.

```
 1  knownDatePatterns = {
 2
 3  'NNNN month NN' : parseNNNNmonthNN,
 4
 5  'NNNN monthNN' : parseNNNNmonthNN,
 6
 7  'NNNN month N' : parseNNNNmonthNN,
 8
 9  'NNNN monthN' : parseNNNNmonthNN,
10
11  'NNNN month' : parseNNNNmonth,
12
13  'NN month NNNN' : parseNNmonthNNNN,
14
15  'NN month NNN' : parseNNmonthNNNN,
16
17  'N month NNNN' : parseNNmonthNNNN,
18
19  'month NN NNNN' : parsemonthNN_NNNN,
20
21  'month N NNNN' : parsemonthNN_NNNN,
22
23  'NNNN NN month' : parseNNNN_NNmonth,
24
25  'NNNN N month' : parseNNNN_NNmonth,
26
27  'NNNN NN NN' : parseNNNN_NN_NN,
28
29  'NNNN-NN-NN' : parseNNNN_NN_NN,
30
31  'NNNN/NN/NN' : parseNNNN_NN_NN,
32
33  'NNNN-NN-NN-' : parseNNNN_NN_NN,
34
```

```
35  'NNNN N N' : parseNNNN_NN_NN,
36
37  'NNNN/N/N' : parseNNNN_NN_NN,
38
39  'NNNN/N/NN' : parseNNNN_NN_NN,
40
41  'NNNN/NN/N' : parseNNNN_NN_NN,
42
43  'NN.NN.NNNN' : parseNN_NN_NNNN,
44
45  'NN-NN-NNNN' : parseNN_NN_NNNN,
46
47  'NN/NN/NNNN' : parseNN_NN_NNNN,
48
49  'NN.N.NNNN' : parseNN_NN_NNNN,
50
51  'N.NN.NNNN' : parseNN_NN_NNNN,
52
53  'N.N.NNNN' : parseNN_NN_NNNN,
54
55  'NN.NNNN' : parseNN_NNNN,
56
57  'NNNNNNNN' : parseNNNNNNNN,
58
59  'NNNN' : parseNNNN,
60
61  'NNN' : parseNNNN,
62
63  'NN' : parseNNNN,
64
65  'N' : parseNNNN,
66
67  }
```

Subscribe to comments: For this article | For all articles