**RDA Phase 3: Mining authority 670 fields for birth and death dates**

Gary L. Strawn

September 1, 2014

**Background**

Over the past few years, efforts to use computer programs to adapt information present in authority records for new purposes have met with considerable success. So far, this work has chiefly built on information present in parts of the authority record that have a clearly defined structure. For example, it has proven quite possible for a program to construct a satisfactory 046 field from subfield $d of the in an authority record for a simple personal name; and a program can identify the immediate hierarchical superior for a corporate body when some version of the hierarchy is given as part of a 4XX field in the body's authority record.

The 670 fields of the MARC authority format contain information that is related in some manner to the entity represented by the authority record. In addition to the usual numeric subfields and subfield $u for a uniform resource identifier, the 670 field contains these textual subfields:

* Subfield $a identifies the source from which information was drawn
* Subfield $b contains information extracted from the source

Although there are what might be called best practices on some points of detail, there are (to our ongoing collective shame) no actual standards for the recording of information in either subfield $a or subfield $b of the 670 field. This means that every possible method for expressing a given kind of information is likely to be met with. This level of casualness in the recording of complex information is layered on top of the varying natures of the sources from which the information in the 670 is copied or distilled. On its face, all of this would suggest that mining 670 fields in the same way that more formally-presented data in the authority record can be manipulated, is not worth the time and effort required and is likely to produce unsatisfactory and misleading results. However, if the approach taken is to find the greatest possible amount of *reliable* information, and to leave alone information in which confidence cannot be placed (perhaps for later review by humans), it can be shown that valuable extracts can be made even from the authority 670 field.

This paper describes one attempt to pull selected bits of information from 670 fields in authority records; namely, it describes an attempt to build or supplement the authority 046 field in authority records for simple personal names by inspecting 670 fields for birth and death dates.

*Throughout these descriptions, assume that character comparisons are made in a case-insensitive manner and ignoring punctuation, unless specifically stated otherwise.*

**Outline of procedure**

This section describes in a brief manner the method the program[[1]](#footnote-1) uses to extract dates from 670 fields for use in the 046 field. Many points given in brief form in this section are expanded in following sections.

*Identification and preliminary inspection of authority records of interest*

To be a candidate for the work described in this paper:

* The authority record must represent a simple personal name (authority record contains a field with tag 100, first indicator not "3", and with no subfields in the 100 field other than subfields $a, $b, $c, $d and $q)
* The authority record must represent a unique personal name (008/32 must not contain code "b")
* If the 100 field contains subfield $c, it must not consist of or otherwise contain the word "spirit" or "mrs"
* If the authority record already contains an 046 field, the existing 046 field may contain subfields $f and $g, but not subfields $s, $t or $2. (The program ignores subfields $v and $u.) The dates in subfields $f and $g of any existing 046 field may not begin with a minus sign (indicating a B.C. date), or a zero (indicating a year in the first millennium of the current era). The length of the text in these subfields may be 4 (year only), 7 (year plus month, with hyphen at position 5) or 8 (year, month and day). If the authority record contains more than one 046 field, all of the 046 fields must pass all of these tests, and there can be no more than one occurrence of subfield $f and one occurrence of subfield $g in all of the 046 fields taken together.
* If the 100 field contains subfield $d, the program attempts to generate its own 046 field from 100 subfield $d using techniques developed during Phase 2 of the manipulation of the LC/NACO Authority File for use under RDA. (The program does this whether or not the authority record already contains an 046 field.) If such an 046 field can be generated, the program does not continue with the authority record if the program-generated 046 field contains subfields other than $f and $g.[[2]](#footnote-2) In addition, the text of subfields $f and $g may not begin with a minus sign or a zero, and the length of the text in 046 subfields $f and $g must be 4, 7 or 8.

The program performs additional steps before it gives consideration to the 670 fields, although these steps do not lead to the exclusion of the record from consideration:

* The program generates a table of names present in the 100, 400[[3]](#footnote-3) and 500 fields. The program does this because versions of each name in either direct or inverted order may be needed at various stages of the work. For each name the program also extracts as separate elements the surnames and forenames, and the contents of any subfields $c and $q
* If any of the words in any of the extracted names can also serve as the name of a month or its abbreviation (examples: June, August), the program builds a list of the numeric equivalents for those months. This list will help prevent the accidental misuse of a part of the person's name as the name of a month

*Inspection of the 670 fields[[4]](#footnote-4)*

The program processes in turn each 670 field in the authority record.

Preliminary considerations for 670 fields:

* A 670 field is only interesting if subfield $b is present[[5]](#footnote-5)
* The program sets aside those 670 fields that contain the text "NUCMC" in subfield $a. Although these fields contain much useful information, they have been found also to contain much information that can lead a program in a very wrong direction. Subfield $b of these fields may be used to confirm or expand on information found in other locations, but cannot be used as an independent source.[[6]](#footnote-6)
* The program identifies 670 fields that cite what might be termed "databases" by inspecting the text of subfield $a. This identification may inform the program's work at a later stage

Preliminary considerations for each 670 subfield $b that passes the preceding tests:

* With a few exceptions, subfield $b of the 670 field must contain an even number of opening and closing parentheses.[[7]](#footnote-7) (The program needs to exclude from consideration the explanatory texts that appear between parenthesized expressions in 670 $b, and it can only reliably do this if the 670field follows a standard formatting convention.) Each such parenthesized expression may be called a *major segment.*
* Whenever the parenthesis counter switches from 0 to 1, the program examines the excluded text that *precedes* the new major segment (that is, the text between parenthesized segments) for characteristics that identify the just-encountered major segment as being derived from CIP or other similar types of data. The program separately labels information from each major segment as to its "CIP" nature.

The program divides each major segment successfully extracted according to the above procedure at each internal semicolon[[8]](#footnote-8) and considers each semicolon-delimited segment of a major segment as a separate entity, in this manner:

* If the text of the segment contains any indication that it represents dates in the Jewish calendar or various Islamic calendars, the program modifies the text of the segment so that the dates in these calendars, and their Gregorian equivalents, cannot be considered by the program.[[9]](#footnote-9)
* If the text of the segment contains "b." or "d." (or an equivalent) that is followed by a recognizable date, the program adds that date to a list of candidate birth or death dates. (This procedure is described in more detail below.)
* If nothing could be extracted from a 670 segment based on "b." or "d." or equivalent labels, and if the segment contains any numerals, the program tries a number of additional possibilities. (These additional possibilities are described below.)
* The program notes for use in its reports any expression in the form "aged <x> on <date>" (and variants), and any expression in the form "died <date> at age <x>" (and variants). This information may be of use to a reviewer when no other dates can be extracted by the program.
* If any kind of date was extracted from a 670 segment, the program inspects the text of the 670 segment to ensure that the context in which the date appears is reliable. This inspection may indicate that dates extracted from this one segment are unreliable, or it may indicate that all dates taken from the entire 670 field are unreliable.

After inspecting all of the segments in one 670 field:

* The program attempts to resolve any ambiguities in the interpretation of dates given as digits ("5/7/1945"); if the program cannot resolve the ambiguity, it sets aside the month and day portions of the extracted dates. (The matter of ambiguous dates is discussed below.)
* If both birth and death dates are extracted from a single 670 field: if the year portions of the dates are not separated by an arbitrary number of years,[[10]](#footnote-10) the program discards all of the dates.

After inspecting all of the 670 fields in one authority record:

* The program attempts to resolve any remaining ambiguities in the interpretation of dates given as digits, as sometimes information in one 670 field can shed light on the interpretation of information in another 670 field.
* The program inspects the year portions of each extracted date for consistency with other dates of the same type (birth or death). In case of conflict, the program retains the date found in the more reliable context; if the dates come from contexts of equal weight, the program discards parts or all of both.
* The program selects for use in the 046 field the most reliable remaining birth date (if any) with the fullest information, and the most reliable remaining death date (if any) with the fullest information.

The program produces the following bits of data that can be used to inform further work on the authority record:

* A list of all of the birth and death dates extracted from the 670 fields, each with a cluster of associated elements (for example: source location, reliability, and the presence of a program-supplied century)
* An identification of the best birth date and the best death date (as determined by the program)
* Texts to be used for 046 subfields $f and $g, if different from any such subfields already present in the record (these reflect the program-determined best dates, filtered by comparison with any existing 046 field)
* An indication that the authority 1XX field containing only a birth or death date in subfield $d could be modified by the addition of (respectively) a death date or birth date.
* If no solid date is available, an indication that the authority record appears to contain a construction such as "aged 68 in 1862" that might be used by a person reviewing the results to construct an 046 field according to EDTF conventions.

The following sections provide additional detail for certain parts of this work. Understand throughout that the descriptions given here, detailed though they may seem, are still only summaries of the work being performed.

**Scanning the 670 field for 'b., 'd.' and equivalents**

Use these markers to find points in a segment that might indicate the presence of a date in the immediately-following string:

|  |  |
| --- | --- |
| **Marker** | **Comments** |
| \* | May be followed by a date of birth |
| #assassinated[[11]](#footnote-11) | May be followed by date of death |
| #b. | Case-sensitive. May be followed by date of birth |
| #born#birth date is #birth date#birth year#dob | Maybe be followed by date of birth |
| #buried | May be followed by date near the date of death. Retain only the year portion of such a date |
| #committed suicide | May be followed by date of death |
| #d.-d. | Case-sensitive. May be followed by date of death[[12]](#footnote-12) |
| #died#dies#death date#death#date of death#decede#decedee#executed | May be followed by date of death |
| #geb. | May be followed by date of birth; but ignore if "um" is present |
| #gest. | May be followed by date of death; but ignore if "um" is present |
| #killed | May be followed by date of death |
| #murdered#murio | May be followed by date of death; if "murio", replace all occurrences of "en" and "el" with a space |
| #né#née | Accent required. May be followed by date of birth |
| #nacio#nacido | May be followed by date of birth; replace all occurrences of "en" and "el" with a space |
| #passed away | May be followed by death of death |
| #starb | May be followed by date of death; but ignore if "um" is present |

If any of these tests identifies a 670 field segment as possibly containing something of interest, treat the text following the marker in the following manner:

* Normalize the text; if this leaves fewer than 3 characters, do nothing with the text
* If the text begins with one of these words, discard the word: "EBD", "EBE", "EBDA", "EBENDA";[[13]](#footnote-13) if this leaves fewer than 3 characters, do nothing with the text
* If the text begins with the word "IN", discard the word. If the text fragment contains another occurrence of the word "IN" examine the text preceding that second "IN"; if this text does not contain any numerals, discard the second "IN" and all preceding text; if this leaves fewer than 3 characters, do nothing with the text
* If the text begins with the word "ON" or "AM", discard the word; if this leaves fewer than 3 characters, do nothing with the text
* If the text contains the word "ON" or "AM" internally, examine the text to the left of this word. If this text does not contain any numerals, discard "ON" or "AM" and the preceding text; if this leaves fewer than 3 characters, do nothing with the text
* If the text begins with the word "DEN", replace any additional occurrences of the word "DEN" with a space

Having done this preparatory work, consider the first character in the remaining 670 segment.

If the first character in the segment is an alphabetic character:

* If the character isthe start of a date in the Japanese era system, use that information to identify the year (if the first character in a Japanese era date is alphabetic, this means that day and month are not present)
* If the character is the start of the name of a month or its abbreviation, construct a date consisting of month plus year, or month, day and year
* If the initial alphabetic character is none of these but if text initially used to identify a putative date consists of more than a single letter ("born" but not "b."), discard any part of the string that appears to indicate the following presence of another date. (For example, if expecting a birth date, discard the word "died" and anything that follows.) If the remaining text contains a four-digit year, look on either side of that year for month and day information, and construct the date.
* Use the presence of hyphens and slashes to locate dates presented in formats such as mm/dd/yy or dd-mm-yy.

If the first character in the segment is a numeral:

* If the numeral is followed (optionally) by another numeral and a designation for a Japanese era, construct a date consisting of month plus year or month, day and year.
* The numeral may be followed by text recognizable as the name of or abbreviation for a month, followed by a year
* Otherwise, attempt to recognize dates in forms such as dd/mm/yy, mm-dd-yy, mm/yy and so on.

**Additional ways to identify useful dates**

One of the most reliable ways to find a useful date involves the inspection of text following markers such as "b." and "d," as described above. However, when this condition is not available, or when the test fails for some reason, there are yet other techniques of varying reliability that can identify dates in 670 subfield $b text.

* Can a year present in 100 $d or the 046 field be located in the 670, possibly adjacent to month and year information?
* Does the 670 segment contain two four-digit years separated by a hyphen?[[14]](#footnote-14)
* Does the 670 segment contain two full dates (month, day, year) separated by a hyphen?
* If the 670 field is recognized as a citation of a "database", does the segment contain a four-digit year? If so, month and day may be adjacent
* If a segment of a 670 field is a "CIP" segment, does the string contain a four-digit year? If so, month and day may be adjacent.
* Does the 670 segment contain a recognizable form of the name in the 100 field or one of the 400 fields (in direct or inverted order), followed by a date?
* Does the 670 segment consist solely of (day), month and year?

**Ambiguities, and their resolution**

When the month portion of a date is given as a number rather than as text, and a number representing the day of the month is also present, the date can present ambiguity as to its interpretation.[[15]](#footnote-15) This is compounded by the fact that different communities have different local practices for the relative positions of month and day; and even if information in a source is recorded exactly as found, this does not mean that all sources present the date information according to a single scheme.[[16]](#footnote-16)

Some clues within a single date are available to help with this problem.

* If one of the numbers is greater than twelve and the other is less than thirteen, the larger number must represent the day of the month and the smaller must represent the month. ("2/15/52" must always mean February 15, 1952.)
* If both of the numbers are less than thirteen and are equal, then although ambiguity is present the ambiguity has no real effect. ("6/6/1952" always comes out as June 6, 1952, regardless of which numeral is taken to be the month.) This doesn't really *resolve* the ambiguity, but it prevents the ambiguity from affecting the outcome.

If another date in the same 670 field is presented in the same manner, and is not itself ambiguous, the structure of the second date can be used to inform the interpretation of first date.

046: : $f 18941011 $g 19730428

670: : $a Her Yüan Ch'ang-ying tso p'in hsüan, 1985: $b t.p. (Yüan Ch'ang-ying) p. 277 (b. 10-11-1894; d. 4-28-73)

*The birth date is ambiguous (October 11? November 10?). This ambiguity can be resolved by noting that in the death date the number that must represent the month comes before the number that must represent the day.*

Similarly, if another 670 field presents what appears to be the same date in an unambiguous manner, that information can be used to disambiguate a date in another 670 field, with possible follow-on effects.

670: : $a Times online WWW site, October 18, 2007 $b (Laxmi Mall Singhvi; b. Nov. 9, 1931; d. Oct. 13, 2007; long-serving Indian High Commissioner in London who also achieved distinction as a jurist and constitutional expert)

\*670: : $a Information from 678 field in this record, converted 2014-05-03: $b (b. 11/9/31)

*The ambiguity in the date "11/9/31" in the second 670 field can be resolved by reference to "Nov. 9, 1931" in the first 670 field. The resolved date from the second 670 then becomes redundant with the date in the first 670.*

A date found in 670 subfield $a of the same field can be used, with care,[[17]](#footnote-17) to disambiguate a date in 670 subfield $b:

670: : $a Tel. call to pub. 11/30/79 $b (Lois Carol Winter, b. 10/11/44)

An ambiguous date in a CIP 670 field with certain characteristics can be determined to have been created in the United States, and so the date may be assumed to have been recorded with the month before the day.[[18]](#footnote-18)

CIP t.p. (Paul Allen Miller) pub. info (b. 11-07-59)

*The program translates the date as 19591107.*

CIP galley (b. London, 5/10/1891; Pentecostal preacher in Britain and N. America; d. 1966)

*The program translates the birth date as 18910510.*

A date in 670 subfield $a of a *subsequent* 670 field can be used, with substantial care, to disambiguate a date in 670 subfield $b. The date in the second 670 $a must be presented in the same format (using either slashes, or hyphens) as the ambiguous date in subfield $b, the date in subfield $a of the second 670 field must be within two weeks of the date of creation of the authority record, and this second 670 field must appear in the record *following* the 670 field with the ambiguous date. All of these tests taken together imply that the 670 field with the ambiguous date and the 670 field with the non-ambiguous date were both created by the same person at the time authority record itself was created.

*Date of creation in 008 field:* 920501

010: : $a n 92047995

100:1 : $a Stadelmann, Luis I. J.

670: : $a His Love and politics, c1992: $b CIP t.p. (Luis Stadelmann) galley (Luís I.J. Stadelmann, S.J.) data sheet (b. 1-11-35)

670: : $a LC data base, 05-01-92 $b (hdg.: Luis I.J. Stadelmann; usage: Luis I.J. Stadelmann)

*The ambiguity in the date "1-11-35" found in the first 670 cannot be resolved by information contained within that field. The second 670 field contains a date in the same format as the ambiguous date, and the date in the second 670 field corresponds well to the date on which the record was created (recorded in the 008 field). Although the date in the second 670 is itself ambiguous, its ambiguity can be resolved by reference to the date of record creation: "05" is the month portion of the date. Based on this constellation of characteristics, the ambiguity of the first date can be resolved: the date of birth is 11 January, 1935, not November 1, 1935.*

If the program cannot resolve the ambiguity in the interpretation of a date whose month portion is given as a numeral, the program will use only the year portion of the date in the 046 field. Although it would be possible to generate a report of all such cases, it is doubtful that any action could be taken on such a report without further research.

**Japanese (etc.) era dates, and similar considerations**

Dates for people from China and Japan are sometimes given in terms of the reigning emperor's era name plus a year. Such dates are sometimes preceded by a month, or a day and a month. Here are some examples:

Taishō 15

9/Taishō 2

8/6/Taishō 7

Minguo 8

Citations of dates in the Japanese system are quite common in authority records. The program contains a table of equivalents for Japanese era names for the Edo period and later. The only Chinese era name that seems to be in common use in authority records is Minguo, so the program recognizes only that name at present.

Happily, in all cases examined where two numbers precede the Japanese era name, the month is *always* given first; as far as can be determined these dates never present the ambiguity possible in dates given in other systems.

Dates in the Thai solar calendar can be converted into Gregorian dates by translating the month into its direct equivalent, and making an appropriate conversion for the year. The conversion of the year is a complicated matter because the date on which the year is said to begin has changed over time. The program does not attempt to convert Thai solar dates earlier than 2355.

010: : $a n 85314631

040: : $a DLC $b eng $c DLC $d OCoLC

046: : $f 19011205 $g 19610716

100:0 : $a Cha'um 'Inthusut, $d 1901-1961

400:1 : $a 'Inthusut, Cha'um, $d 1901-1961

670: : $a Pāthakathā bāng rư̄ang khōng Phon Trī Lūang Wičhitwāthakān læ Khunying Wičhitwæthakæn, 1961: $b t.p. (Khun Mæ Cha'um 'Inthusut) p. 1, 2nd group (b. 5 Thanwākhom 2444) p. 4 (d. 16 Karakkadākhom 2504)

The program will also convert dates expressed in terms of the Thai solar calendar that are not explicitly identified as such, if sufficient support is present elsewhere in the record.

010: : $a n 82151756

040: : $a DLC $b eng $c DLC $d OCoLC

046: : $f 19061121 $g 19811118

100:0 : $a Čaran Sāranāk, $d 1906-1981

400:1 : $a Sāranāk, Čharan, $d 1906-1981

670: : $a 'Anusōn nai ngān phrarātchathān phlœng sop Nāi Čharan Sāranāk, ... 1981 (a.e.) $b t.p. (Čharan Sāranāk) guard sheet (b. 11/21/2449; d. 11/18/2523)

*Although the dates in the 670 field are not identified as being constructed according to the Thai lunar calendar, by assuming that they might be, and comparing the converted years to the years in the 100 field, the assumption can be confirmed and a complete 046 field constructed.*

**Not all useful dates will be found**

Although the program is able to identify *most* of the relevant dates in the 670 fields of an authority record, it cannot find all of them. Many tempting avenues of exploration cannot be followed, because of the likelihood that irrelevant dates would be swept up along with the good ones. A review of authority records whose 670 $b texts contain numerals but from which dates were not extracted will turn up occasional candidates for use in an 046 field. Here are a few examples:

010: : |a n 82012341 |z n 85329076

100:1 : |a Fonseca Martel, César

670: : |a His Comunidad y producción en la agricultura andina, 1988: |b t.p. (César Fonseca) p. 212 (1935-1986, Llata, Huáuco, Perú; profesor y jefe del Departamento de Ciencias Sociales en la Universidad Nacional del Centro)

*"1935-1986" may well be the dates of birth and death, but the context does not clearly identify them as such.*

010: : |a n 82249177

100:1 : |a Vidal, Cecília

670: : |a Xavier Nogués, 1873-1941, 2010: |b t.p. (Cecília Vidal Maynou) p. 6 (Barcelona, 1940)

*"1940" may be the date of birth, but the context does not clearly identify it as such.*

010: : |a n 83132702

100:10: |a Göbel, Thomas

670: : |a "Zum Erstaunen bin ich da", c1998: |b p. 5 (Thomas Göbel) p. 11 (70 on Feb. 11, 1998)

*If the statement in 670 $b is taken literally, the date of birth can be calculated, but the program makes no attempt to do so.*

010: : |a n 82203085

100:1 : |a Coven, Stephen

670: : |a NUC pre-1956 imprints |b (Coven, Stephen, fl. 1668)

*The date in the 670 field could be used in 046 subfield $s; this program does not generate subfields $s or $t.*

010: : |a n 88121854

100:1 : |a Brass, Helmut

670: : |a Numerical integration 111, 1988: |b CIP t.p. (H. Brass) data sheet (Brass, Helmut; b. 2-22-1934)

670: : |a Quadrature theory, c2011: |b ECIP t.p. (Helmut Brass) data view (b. Feb. 22, 1936)

*The year portions of the two dates contradict each other; the program is forced to discard both.*

010: : $a nb 99181079

035: : $a (OCoLC)oca05363226

040: : $a Uk $b eng $c Uk $d TxU-Mu

100:1 : $a Lipesker, S.

400:1 : $a Lipesker, Santos

670: : $a Quiero bailar boogie, 1945

670: : $a Una carta para Italia, c1948: $b caption (letra y música de Santos Lipesker y Reinaldo Yiso)

670: : $a Club de Tango WWW site, May 26, 2004: $b efemérides tangueras (Junio 30 1978. Falleció Santos Lipesker, bandoneonista que actuara con Pedro Maffia y tuvo su orquesta, André y su conjunto. Había nacido el 10 de octubre de 1918)

*The program will be able to interpret "nacido el 10 de octubre de 1918" as the birth date. Because of the manner in which the death date is presented elsewhere in the 670 field, the program will not able to find it.*

010: : $a n 85051184

100:1 : $a Salvi, Nicola

670: : $a Oxford art online, July 23, 208 $b (Salvi, Nicola; b Rome, 6 Aug 1697; d Rome, 9 Feb 1751)

*The lack of full stops after "b" and "d" prevents the program from recognizing the birth and death dates.*

**Appendix: Information that the class makes available to programs that contain it**

Although programmers are not the primary audience for this document, non-programmers may be interested in the range of information made available by the code module that does the work of parsing 670 fields in an authority record for birth and death dates.[[19]](#footnote-19)

*General information*

* *AddBirthDateTo100D:* If this has the value True, subfield $d in the 100 field has a death date and no birth date; but another part of the authority record appears to contain a birth date. A program could use this information to add the birth date to the 100 field automatically; or a report could be generated for individual review.[[20]](#footnote-20)
* *AddDeathDateTo100D:* If this has the value True, subfield $d in the 100 field has a birth date and no death date; but another part of the authority record appears to contain a death date. A program could use this information to add the birth date to the 100 field automatically; or a report could be generated for individual review.
* *AgeConstructionPresent:* The authority record appears to have one or more text strings that suggest that it might be possible to calculate or estimate a person's birth and/or death dates. (Such a string in a 670 field might take a form such as "aged 67 in 1952".) The program's rules for detecting this condition are somewhat broad; no automated action could be based on this information, but a report could be generated for individual review.
* *AtLeastOne670AWithoutBContainsParentheses:* At least one 670 field in the record does not contain subfield $b but does contain opening and closing parentheses. If this property is True, it may be the case that one or more 670 fields contains subfield $b information, but the subfield $b code itself is not present.
* *AtLeastOne670BContainsNumerals:* At least one670 subfield $b in the authority record contains at least one numeral. If the program is not able to extract any date information from the 670 fields, if this has the value True a report could be generated for individual review. Such fields may contain dates presented in ways that the program cannot recognize.
* *AtLeastOne670IsBadlyFormed:* At least one 670 subfield $b in the authority record is, according to the program's rules, badly formed. The program expects that subfield $b of a 670 field will consist of parenthesized expressions separated by additional text. If 670 $b contains unequal numbers of left and right parentheses, the program must ignore the 670 field, and so will not be able to extract any date information that the 670 field might contain. This information might be used to generate a report that could be used by an operator to change the structure of 670 fields, extract date information from the 670 fields, or both.
* *BirthDateBestIndex:* If this is true, this is the value to supply as the *BirthDateIndex* property to expose information about the birth date that the program considers to be the best. This date is the source of (but is not necessarily the same as) the date suggested for use in 046 subfield $f.
* *BirthDatesConsistent:* If this is True, the birth dates that the program found are entirely consistent. If this is False but the *BirthYearsConsistent* property is True, the year portions of the birth dates are consistent across the 670 fields, but the month and/or day portions are not consistent.
* *BirthYearOccurrences:* A count of the number of times the program found what it believes to be the "correct" birth date in the 670 fields.
* *BirthYearsConsistent:* If this is True, the year portion of the birth dates that the program found are consistent across the 670 fields.
* *DeathDateBestIndex:* If this is true, this is the value to supply as the *DeathDateIndex* property to expose information about the death date that the program considers to be the best. This date is the source of (but is not necessarily the same as) the date suggested for use in 046 subfield $g.
* *DeathDateMayBeAvailable:* If this is True, the program found something in the 670 fields that might be a death date, but program was not able to construct a complete date. This information could be used to generate a report. (For example, a 670 field might say "died June 6" without giving the year, but it is possible that the year might be inferred from information present elsewhere in the authority record.)
* *DeathDatesConsistent:* If this is True, the death dates that the program found are entirely consistent across the 670 fields. If this is False but the *DeathYearsConsistent* property is True, the year portions of the death dates are consistent, but the month and/or day portions are not consistent.
* *DeathYearOccurrences:* A count of the number of times the program found what it believes to be the "correct" death date in the 670 fields.
* *DeathYearsConsistent:* If this is True, the year portion of the birth dates that the program found are consistent across the 670 fields.
* *FailureReason:* A number that indicates why the program was not able to generate any date information from the authority record.[[21]](#footnote-21)
* *FailureReasonText:* A displayable piece of text corresponding to *FailureReason* that indicates why the program was not able to generate any date information from the authority record.
* *Field670SkippedAltogether:* If this is True, the program found at least one 670 field in the record that it was compelled to ignore.[[22]](#footnote-22) This information might be used to generate a report for review.
* *Field670SkippedSegment:* If this is True, the program found at least one segment of a 670 field that it was compelled to ignore.[[23]](#footnote-23)
* *RecordIsACandidate:* If this is True, the program considered the record to be a candidate for closer inspection. If this is False, the program did not consider the record to be a candidate. (The program's rules for identifying candidates are listed elsewhere in this document.)
* *SubfieldFFor046:* The program's suggestion for text to use in subfield $f of the 046 field. If the 046 field already contains subfield $f, this text is intended to *replace* it (this will only happen if the program is able to add month or month/day to a year). If the record already contains 046 $f and the program's best suggestion for this is the same, the *SubfieldFFor046* property is empty.
* *SubfieldGFor046:* The program's suggestion for text to use in subfield $g of the 046 field. If the 046 field already contains subfield $g, this text is intended to *replace* it (this will only happen if the program is able to add month or month/day to a year). If the record already contains 046 $g and the program's best suggestion is the same, the *SubfieldGFor046* property is empty.

*General information about birth dates contained in the authority record*

As the code module reads through an authority record, it collects information about dates in an internal array, which it then makes available to the program that contains it. The program makes the following properties available for each birth date it finds.[[24]](#footnote-24)

* *BirthDateAmbiguity:* If a 670 field gives the month and day as numerals, the number given here represents the program's decision about the handling of the month and day.
* *BirthDateAmbiguityText:* A piece of display text corresponding to the value in the *BirthDateAmbiguity* property.
* *BirthDateAmbiguousPossibility1, BirthDateAmbiguousPossibility2:* If a 670 field gives the month and day as numerals and the module cannot resolve the ambiguity through other clues, the program presents only the year portion of the date as its finished date, and presents in these two properties the two possibilities. For example, if the program is unable to resolve the ambiguity in the date "8/10/1952" the program will present "19520810" and "19521008" in these two properties. No importance should be accorded to the program's choice of one date to be Possibility1 and the other date to be Possibility2.
* *BirthDateField670SegmentNumber:* A number indicating the segment of a 670 field from which the program extracted the date. The program's scheme for dividing a 670 field into segments based on parentheses, and then by semicolons, is described elsewhere in this document.
* *BirthDateField670SequenceNumber:* A number indicating the 670 field from which the program extracted the date. The program numbers first 670 field in the record number 1, the second as number 2, and so on.
* *BirthDateIgnoreMe:* A number indicating (if not zero) why the program chose to ignore a date that it found.
* *BirthDateIgnoreMeText:* A piece of display text corresponding to the number on the *BirthDateIgnoreMe* property.
* *BirthDateSource:* A number that indicates how the module identified the birth date.
* *BirthDateSourceTag:* The tag of the variable field from which the program extracted the date (possibilities: 046, 100, 670).
* *BirthDateSourceText:* A piece of display text corresponding to the number in the *BirthDateSource* property.
* *BirthDateSuppliedCentury:* Ifthis is True, the program supplied the century portion of the birth date.
* *BirthDateText:* The date extracted by the module from the authority record.

*General information about death dates contained in the authority record*

As the code module reads through an authority record, it collects information about dates in an internal array, which it then makes available to the program that contains it. The program makes the following properties available for each birth date it finds.[[25]](#footnote-25)

* *DeathDateAmbiguity:* If a 670 field gives the month and day as numerals, the number given here represents the program's decision about the handling of the month and day.
* *DeathDateAmbiguityText:* A piece of display text corresponding to the value in the *DeathDateAmbiguity* property.
* *DeathDateAmbiguousPossibility1, DeathDateAmbiguousPossibility2:* If a 670 field gives the month and day as numerals and the module cannot resolve the ambiguity through other clues, the program presents only the year portion of the date as its finished date, and presents in these two properties the two possibilities. For example, if the program is unable to resolve the ambiguity in the date "8/10/1952" the program will present "19520810" and "19521008" in these two properties. No importance should be accorded to the program's choice of one date to be Possibility1 and the other date to be Possibility2.
* *DeathDateAmbiguityText:* A piece of text intended for public display that corresponds to the number given in *DeathDateAmbiguity*
* *DeathDateField670SegmentNumber:* A number indicating the segment of a 670 field from which the program extracted the date. The program's scheme for dividing a 670 field into major segments based on parentheses, and then by semicolons, is described elsewhere in this document.
* *DeathDateField670SequenceNumber:* A number indicating the 670 field from which the program extracted the date. The program numbers first 670 field in the record number 1, the second as number 2, and so on.
* *DeathDateIgnoreMe:* A number indicating (if not zero) why the program chose to ignore a date that it found.
* *DeathDateIgnoreMeText:* A piece of display text corresponding to the number on the *DeathDateIgnoreMe* property.
* *DeathDateSource:* A number that indicates how the module identified the death date.
* *DeathDateSourceTag:* The tag of the variable field from which the program extracted the date (possibilities: 046, 100, 670).
* *DeathDateSourceText:* A piece of display text corresponding to the number in the *DeathDateSource* property.
* *DeathDateSuppliedCentury:* Ifthis is True, the program supplied the century portion of the death date.
* *DeathDateText:* The date extracted by the module from the authority record.
1. Speaking more correctly, what is here called a *program* is actually a *class* that must be instantiated within some larger package; the larger package incorporates the class module that does the work described here. This distinction will only mean something to programmer-type people, and will be ignored in this document unless the distinction is critical to a point under discussion. [↑](#footnote-ref-1)
2. Among other things, this means that the presence of subfield $2 in the machine-constructed 046 field disqualifies the authority record from consideration for the work described in this document. [↑](#footnote-ref-2)
3. The program ignores 400 fields for vernacular scripts, 400 fields that are suppressed, and 400 fields for former headings. [↑](#footnote-ref-3)
4. This description presumes that older-style 678 fields have already been converted into 670 fields. The reformulation of 678 fields into 670 fields is a separate part of the RDA Phase 3 project, and is described in a separate document. [↑](#footnote-ref-4)
5. Some 670 fields consisting of subfield $a may actually contain useful information: "The life of K. Smith, Aug. 15 1896-July 5, 1962, together with a listing of her descendants, 1965". However, the inspection of 670 subfield $a for useful date information has proven to lead to many unfortunate results, and will not be pursued as part of this project. This does not, however, mean that 670 subfield $a plays no part in the work described in this document. The program prepares a report of each 670 field with no subfield $b whose subfield $a contains parentheses; these fields may need to have the $b code added. [↑](#footnote-ref-5)
6. This is not intended in any way to be a criticism directed at the NUCMC folks. The difficulty is that there is often so much rich date-shaped information in these fields in addition to dates of immediate interest (for example: birth and death dates of a person's relatives) that it has proven impossible to tell a program how to figure out which dates to latch onto and which to ignore. [↑](#footnote-ref-6)
7. The program's method is to inspect the text of subfield $b from left to right, incrementing a counter for each opening parenthesis encountered, and decrementing the same counter for each closing parenthesis. Any time the parenthesis counter arrives at zero after having been above zero, the program has come to the end of a major segment. (This technique handily allows for nestedparentheses.) There are of course exceptions. If at any closing parenthesis the counter is *below* zero, and if the remainder of the 670 field does not contain any more opening parentheses, the program uses the text beginning from the putative opening parenthesis as a major segment. If at the end of the examination of subfield $b the parenthesis counter is 1, in a similar manner the program takes text from the beginning of the putative opening parenthesis as the text of a major segment. The program generates a report of 670 subfield $b that do not contain acceptable parentheses. In many cases these are older 670 fields with the author of a resource in subfield $a and the title of the resource in subfield $b; in such cases, the $b code should be removed, placing all of the text in subfield $a. [↑](#footnote-ref-7)
8. In the past, it was at many institutions a convention that individual data elements plucked from a given location within a data source were delimited by semicolons. This handy convention is still in play, but in the last year or two it has become increasingly common for operators blindly to copy a block of text from an online resource directly into 670 subfield $b without any reformatting or editing. Such information is often presented as full sentences rather than as semicolon-delimited bits of information. It has proven impossible for the program described in this document to divide such a text block into sentences, and to treat each sentence as a separate entity. (The 670 $b text may contain many full stops followed by space and an uppercase letter that do not indicate a sentence break.) One consequence of this inability is that the program often considers the entire 670 $b to be a single segment; this means in turn that the program often sets aside all of the information contained in the segment because of characteristics that are, on inspection by a trained operator, not related to the date of interest. [↑](#footnote-ref-8)
9. It can be demonstrated that such information can be used successfully to generate 046 fields in many cases. However, problems occur often enough in this work that it seems best to refer such 670 fields to an operator for individual handling. This restriction does not apply to dates recorded in terms of a Japanese regnal era, or to dates recorded in terms of the Thai solar calendar. [↑](#footnote-ref-9)
10. The current version of the program requires an interval of 25 years between birth and death; this can easily be adjusted if necessary. [↑](#footnote-ref-10)
11. In this list, "#" indicates a mandatory blank space. [↑](#footnote-ref-11)
12. If "#d." is the beginning of a construction containing "d. o. b.", "d. of b." or variants, treat any following date as the date of birth. [↑](#footnote-ref-12)
13. With a designation such as this, a German reference source may indicate that a person was born in a given city on a given date, and died on a given date *in the same city.* [↑](#footnote-ref-13)
14. At present, the program only applies this test if the 670 segment contains nothing other thantwo four-digit years and a hyphen. [↑](#footnote-ref-14)
15. For example, a date given as "3/5/1963" might mean either the March 5 or May 3 of 1963. The accepted practice for constructing the 670 field is to use textual names of months, not their numeric equivalent; but dates in many authority records, especially older ones, contain months represented as numerals. A date transcribed as numerals may represent the data as represented in the source, or it may represent the translation by the cataloger of unambiguous information into an ambiguous form. There is no ambiguity if the month is represented as a *roman* numeral: "3.XI.1937" can only represent November 3, 1937. [↑](#footnote-ref-15)
16. For an example of a case where it appears that information in a source was interpreted incorrectly by the person constructing the authority record, consider n 81065103, where the birth date is given as 01-07-52 in one place and 07-01-52 in another. One of these is likely to be an injudicious re-interpretation of a date given in another form; but which one? But was the error made by the cataloger, or present in the original source? In such a case, the program discards the month and day and uses just the year. [↑](#footnote-ref-16)
17. The program does not look for a date in 670 subfield $a if the 670 $a text contains the string "678" and "data", "information" or "info." (as in "Information from 678 field, converted 3/24/2014"). [↑](#footnote-ref-17)
18. The field must begin with one of several recognized "CIP" markers, and the full text of the statement that precedes the segment containing the ambiguous date must likewise be one of several recognized markers ("data sheet", "galley" and so on). [↑](#footnote-ref-18)
19. Programmers: This appendix does not constitute instructions in the use of the code object that parses authority records. The items listed here are read-only properties of the code object that examines 670 fields. [↑](#footnote-ref-19)
20. Although it would be possible for a program automatically to add the birth date to 100 $d, such a course is *not recommended,* especially so if the authority record already contains an 046 field without subfield $f. In at least a few such cases the birth date that the program has identified is not in fact a reliable date; the cataloger constructing the record carefully omitted 046 subfield $f for a good reason. It is better if cases where it appears that a birth date might be added to 100 $d be reviewed by competent operators. [↑](#footnote-ref-20)
21. Reasons include: information already present in the 046 or 100 field blocks the program from continuing; the record is for an undifferentiated personal name, or a family name; the record is for a "spirit"; the record is for more than just a personal name (name/title, for example); the birth/death years presented in the 046 and 100 fields disagree; the birth year is not less than the death year; the birth year and the death year are too far apart. [↑](#footnote-ref-21)
22. Reasons include: presence of "B.C." or "B.C.E.", and indications that a date somewhere in the segment might be represented in an alternate calendar, such as "julian", "gregorian", "H.", "lunar calendar" and "old style". [↑](#footnote-ref-22)
23. The program only inspects the context of a segment of a 670 field if it has extracted something that appears to be a date. Reasons include: presence of an indication that a date belongs to another person ("daughter", "Jr."); presence of an indication that a date is not exact ("about", "before", "ca."); indication that a date is not a birth or death date ("baptized", "buried") and many other reasons. [↑](#footnote-ref-23)
24. Programmers: The number of birth dates found is available as the *BirthDateCount* property. If this value is greater than 0, set the *BirthDateIndex* property to values from 0 through *BirthDateCount* -1 to expose the information pertaining to each birth date in turn. If the *BirthDateBest* property is greater than -1, it contains the value for the *BirthDateIndex* property that points to the date that the program has decided is the "best" birth date it found in the record. [↑](#footnote-ref-24)
25. Programmers: The number of death dates found is available as the DeathDateCount property. If this value is greater than 0, set the DeathDateIndex property to values from 0 through DeathDateCount-1 to expose the information pertaining to each birth date in turn. If the *DeathDateBest* property is greater than -1, it contains the value for the *DeathDateIndex* property that points to the date that the program has decided is the "best" death date it found in the record. [↑](#footnote-ref-25)