Date Parsing Code

The basis of the parser is to support the paradigm defined within ISO-8601:2019 and EDTF (the Extended Date Time Format) through a number of common date expresssions.

ISO-8601:2019

ISO 8601:2004 has been superseded by ISO 8601:2019 which now contains two parts. Part 1 specifies basic rules. Part 2 specifies extensions, including community profiles to specify conformance levels for use of which rules.

Fundamentally from the perspective of activities such as search where data comparison is required, it represents a paradigm shift. The W3C, OASIS and their ilk view of dates are really timestamps assuming a common reference written in a consistent manner following a specific ISO-8601 expression cookie-cut. This is wholly ill-suited to task. The new 8601 revision now views dates and times as having an implicit (or explicit) accuracy, precision and readability just as the other standards of weights and measures. It also includes vagueness, uncertainty, wildcards and missing information.

Part 1 Basic Rules
The main changes of interest are representing positive leap seconds as [23:59:60Z] and disallowing the value “24” for hour, so that there is no “end of day” overlap with the beginning of the day [00:00.00Z]. The extended format date times have separators (YYYY-MM-DDThh:mm:ss.i). The basic format date times are without separators, except that basic times always start with “T”. Year is ordinarily an ordinal between 0000 and 9999, however expanded representations allow larger and negative years by mutual agreement. A time may include a decimal fraction (seperated using comma or period) in the lowest order time scale component.

A complete representation of duration can be expressed as a combination of units with designators starting with “P” : PYYMMDDTHHMMSS . By mutual agreement the basic and extended formats prefixed with “P” can be used.

In Part 2 Extensions many extensions are provided. EDTF functionality has now been integrated into ISO 8601-2019. For level 2, It adds the ablity to express years with exponents, and to specify number of significant digits; divisions of a year into seasons, quarters, semestrals and quadrimesters; and the ability to specify multiple dates with {braces}, one of a set with [square brackers], or a time interval with an unknown (blank) or open (double dot ..) start or end.

The code fragement does not support the full ISO-8601:2019 including extensions but does include large bits as well as its own extensions—many from the involvement of it’s main author in the creation of the EDTF standardized that formed the 2019 revision of the 8601 standard.
Of significant is the algorithm for matching dates (and times). While timestamps can be easily extended to assumed precision (and accuracy can be rationalized out of the equation) for general date and time expressions this is not possible. Observing that the range of timezones alone throughout the word surpasses the 24 hours in a day means that without an expression of time against a standard or geospatial location two expressions of date with an expression of day may refer to the same instant despite being on different days. The Line Islands (part of Kiribati) is UTC+14. The timezone of Baker

and Howland Islands is UTC-12. Wall clocks in Kiritimati and Howland always read 2 hours apart but on two distant separate days—they are 26 hours apart!

A common error is ETL pipelines is to try to resolve date expressions as-if they were timestamps. All too often they get resolved to the local time where the pipeline runs. A date such as 22 August 2021 would get encoded according to the local time, e.g. 20:19 (Munich) or resolved in UTC as 18:20. As one can see, however, an expression of 20210822T18:20:20Z for an event that gets processed on a machine in Tokyo using the same algorithm would get 20210823T03:20:20Z. As timestamps they clearly don't match. Even as a range, the one processed in Munich gets 2021-08-22 while the Tokyo process gets 2021-08-23. The process of "cleaning" the data in the pipeline did not clean anything but added noise and error. What date did the US land on the moon? Ask any schoolchild? 20. July 1969. 20. Juli 1969, 20:17:40 UTC. But that was 21th July in Tokyo! The moral of this story is that we can't add information we don't have. If we don't know or are uncertain of the time we can't just fill in. ISO-8601:2019 goes even a step further as it even allows for dates to have uncertain or even unknown values—values that may be corrected in the future or filled-in once additional information is available.

ISO-8601:2019 Part-2 extensions:

In ISO-8601:2019 extensions there are two levels defined: 1 and 2.

Approximate (~)

An estimate that is assumed to be possibly correct, or close to correct, where "closeness" may be dependent on specific application.

Uncertain (?)

We are not sure of the value of the variable (in our case date or time). Uncertainty is independent of precision. The source of the information may itself not be reliable, or we may face several values and not enough information to discern between them. For example we may be uncertain as to the year, or month, or day of an event etc.

The character '%' (per-cent) is used to mean both uncertain and approximate.

Level 1: '?,' '~', or '%' may only occur at the end of the date string, and it applies to the entire date.
Level 2: '?', '~', and '%' may be used to qualify a portion of the date

Unspecified (X) – in earlier versions of EDTF the character 'u'

The value is not stated. The point in time may be unspecified because it did not occur yet, because it is classified, unknown or for any other reason.

Level 1: 'X' may be substituted for the right-most digits, e.g. day, day and month, …
Level 2: 'X' may be used as a replacement for any character in the string.

Before and/or after (..)

Level 1: Not available

Date Parsing Code

Level 2: ..” (two dots) may be used before a date to denote “before or on the date” or after a date to denote “on this date or after”.

Extended Year Formats (beyond year 9999)

‘Y’ may be used at the beginning of the date string to signify that the date is a year, when (and only when) the year exceeds four digits, i.e. for years later than 9999 or earlier than -9999.

Significant Digits (available only in Level-2)

A year is followed by ‘S’, followed by a positive integer, the integer indicates the number of significant digits for the expressed year.

Example: “1950S2” := Some year between 1900 and 1999 , estimated to be 1950.

Seasons (Year Divisions)

Level 1: The values 21, 22, 23, 24 may be used to signify ' Spring', 'Summer', 'Autumn', 'Winter', respectively. Example: 2020-21 for Spring 2020

These values are from the perspective of the “reporter” and don’t respect their location. But since Summer is the Southern Hemisphere maps to Winter in the Northern we have in Level-2 additionally:

25-28 = Spring - Northern Hemisphere, Summer- Northern Hemisphere, Autumn - Northern Hemisphere, Winter - Northern Hemisphere
29-32 = Spring – Southern Hemisphere, Summer– Southern Hemisphere, Autumn – Southern Hemisphere, Winter - Southern Hemisphere
33-36 = Quarter 1, Quarter 2, Quarter 3, Quarter 4 (3 months each)
37-39 = Quadrimester 1, Quadrimester 2, Quadrimester 3 (4 months each)

Decade (available in Level-2 only)

Format: YYY Example: 196 (1960-1969) Example: 196~ (approximate decade)

Note: the expression 196 expresses what is commonly called the “1960s”. It does not refer to an event within the interval defined from 1 Jan 1960 to 31 Dec 1969 but refers to a singular event viewed with decade precision.

Parser

The date parser is relatively complicated and less than straightfoward due to a great extent due to its ambition to handle the large range of date formats and semantics in widespread use. Internally the parser attempts to store dates in a manner consistent with ISO-8601:2019 (as well as some features of EDTF that did not make it into the ISO standard). Signifcant date formats that are parsed are, among others, RFC-2822 dates (and some derivates) as commonly used in electronic mails as well as “common” (and typically not well defined and/or ambiguous formats are found in many countries such as M/D/[YY]YY, DD/MM/[YY]YY, [YY]YY-MM-DD, DD.MM.[YY]YY etc.

Date Parsing Code

- The date parser for long date names understand only the following languages: English, French, German (and Austrian variants), Spanish, Italian and Polish. Adding additional languages is straightfoward: see date.cxx
- Valid dates included are also the national numerical only (non-ISO) standards using the '-'. '.' and '/' styles of notation. Dates are either intrinsically resolved or should they be ambiguous using the locale. In a date specified as 03/28/2019, its clear that the 28 refers to day of the month, while an expression such as 03-03-23 is ambiguous. Sometimes the the use of a dash, slash or period has a semantic difference but sometimes they are used interchangeably. In Sweden for instance the '-' notation is generally used as Year Month Day (e.g. 99-12-31) while in the US it is written as Month Day Year and in much of the European continent its Day Month Year. In Britain both Month Day Year and from the end of the 19th Century Day Month Year are, aligning with the Continent, commonly encountered. In the UK, in fact, all of the following forms 31/12/99, 31.12.99, 31.xii.99 and 1999-12-31 are encountered. With years in YYYY notation or with NN > 31 its clear that it can't be day of the month just as NN > 12 can't be the number of the month.
- In two digit YY specified years the year is resolved as following:

Current Year Last Two Digits	Two Digit Year Specified	Year RR Format Returns
0-49	0-49	Current Century
50-99	0-49	One Century after current
0-49	50-99	One Century before current
50-99	50-99	Current Century

Unix Model: Year starts at 70 for 1970

00-68	2000-2068
69-99	1969-1999

- The parser understands precision of day, month, year
- The parser understands BC and BCE dates, e.g. "12th century b.c."

The parser supports number of special reserved terms for dates such as "Today", "Now", "Yesterday", "Last Week", "Past Month", "14 days ago" etc. Inputs such as 2/10/2010 are ambiguous and should they be desired added as local formats (See next section).

Relative dates controlled vocabulary

"Today" := the LOCAL date (of the server) without time (day precision).
"Yester[day|week|month|year]" := the past X (X precision) from the point of view of the local date/time where X is one of day, week, month or year (e.g. Yesterday is the day before today in day precision)

"Tomorrow" := the day after today (day precision)

Date Parsing Code

"This day" := Today
"This Month" := the current month (LOCAL) in month precision
"This year" :? the current year (LOCAL) in year precision.
Now or Present -- the date/time at the moment the word is parsed.
[Last|Past] [NNN] Sec[onds]|Min[utes]|H[ours]|Day[s]|Week[s]|Month[s]|Year[s]|Decade[s]|
Millennium [Ago|Past]
Last|Past&nsp;Sun[day]|Mon[day]|Tue[sday]|Wed[nesday]|Thu[rsday]|Fri[day]|Sat[urday]

Examples: "*14 hours ago*", "*6 days past*", "*Last year*", "*Past month*", "*200 minutes ago*", "*Last Monday*"

Note: "Yesterday" is date precision while "Past Day" is date/time precision. The two also might refer to two different days due to the time difference between local time (from the perspective of the server) and UTC/GMT: "Past Day" is based upon the date/time as per UTC/GMT while Today, Yesterday and Tomorrow are based upon "local" time.

Note also that "14 days ago" defines a date as well as method of comparison based upon day precision---thus something different from 335 hours ago or 2 weeks ago. Last year, resp. month etc., each define a precision of year, resp. month etc.

The prefix "*End of*" is interpreted as the end of the period.

The sequel in date ranges: NNNNs (as in "*1950s*") NNNN century (as in "*19th century*", "*19 Jh.*" etc) Past|Last [NNN] Seconds|Minutes|Hours|Days|Weeks|Months|Years|Decades|Millennium
Last|Past Sun[day]|Mon[day]|Tue[sday]|Wed[nesday]|Thu[rsday]|Fri[day]|Sat[urday]

Examples: "*Past 14 days*", "*Past 24 hours*", "*Past month*", "*Since last Friday*" NOTE: The prefix words "*Within*", "*during*" and "*the*" are skipped, e.g. "*Within the past day*" is reduced to "*Past day*".

Since means from then until now. Since last week means from the start of last week to now. "*Last week*, by contrast, means only the days of last week (Sunday through Sat.). Date ranges, in contrast to dates, are precise to time but start at the beginning and end at the end of their respective unit, e.g. last month starts at midnight of the first and ends just as midnight strikes on the last day of the month (in Oct. its the 31th day).

Local Extensions to handle "ambiguous" national formats

A number of local formats can be easily added via a "datemsk.ini" file. That file is quite simple and uses host-platform STRPTIME(3) function.

RFC-2822 Dates

Date and time occur in several email header fields. This section specifies the syntax for a full date and time specification.

```
date-time       =       [ day-of-week "," ] date FWS time [CFWS]
```

Date Parsing Code

```
day-of-week     =       ([FWS] day-name) / obs-day-of-week
day-name        =       "Mon" / "Tue" / "Wed" / "Thu" /
                        "Fri" / "Sat" / "Sun"
date            =       day month year
year            =       4*DIGIT / obs-year
month           =       (FWS month-name FWS) / obs-month
month-name      =       "Jan" / "Feb" / "Mar" / "Apr" /
                        "May" / "Jun" / "Jul" / "Aug" /
                        "Sep" / "Oct" / "Nov" / "Dec"
day             =       ([FWS] 1*2DIGIT) / obs-day
time            =       time-of-day FWS zone
time-of-day     =       hour ":" minute [ ":" second ]
hour            =       2DIGIT / obs-hour
minute          =       2DIGIT / obs-minute
second          =       2DIGIT / obs-second
zone            =       (( "+" / "-" ) 4DIGIT) / obs-zone
```

For month-name we support a number of other languages.

Precision

Precision is a measure of a range or interval within which the ‘true’ value exists. Precision is explicit in the date or date/time expression; if an event occurred in the year 1318, the precision is one year (it could occur at any time within this year). If we specify 1945-09-15, the precision is one day, etc. In EDTF we can extend this definition to a specify a decade or century precision using the X symbol - see discussion of masked precision above. In Level-2 of ISO-8601 we have an extention that explicitly expresses decade precision.

Date Comparisons/Matching

Central to the code is the problem: When do dates/times match?

Elements when searched are assumed to be of their intrinsic datatype when the query if of the same type. Searching a date field, for example, with a date implies a search using the date data type matching algorithms rather than as the individual terms. What constitutes a match depends upon the data type. Dates for example follow the rules of least precision comparison. A date query for 2001 matches, for example, 20010112. While >19 as a string is equivalent to 19* and matches 199 or even 19x as a numerical field all values > 19, e.g. 20, 21, 22, 23 etc.