CFPS 67
(Call for Papers Submission number 67)

Proposal to extend the calendar style mechanism of CFPS 43 into an abstract formatting model

Submitted by: Smith, Richard

Created: 2013-04-30

URL: Most recent version: http://fhiso.org/files/cfp/cfps67.pdf
This version: http://fhiso.org/files/cfp/cfps67_v1-0.pdf

Description: CFPS 43's style mechanism is extended into abstract formatting model that would allow applications to format correctly dates written in many unknown calendar systems.

Keywords: dates, calendars, date style, formatting, calendar facets

Abstract

This paper develops the concept of a calendar style IDs proposed in CFPS 43 to encapsulate important presentational aspects of dates, and in the process renames them calendar facets. The need to support third-party facets, potentially in combination, requires a general vocabulary to be used in their definitions. A simple vocabulary is provided by a placing facets into classes, and two such classes are proposed: one relating to the presentation of years, and one for the rest of the date. Two new facets are proposed to serve as the defaults for these classes.

It is shows how facets can be used to develop an abstract and extensible model for formatting dates, thereby allowing applications to better handle unfamilar calendars. Two further classes and associated default facets are introduced to describe remaining aspects of formatting, and formal definitions are given for the four facets proposed in this paper. After several further examples of this framework in use, it is noted that a major use of facets will be to specify year numbering schemes that count from a different epoch. A general alogithm is given for handling arbitrary epochs.

1 Introduction

The discussion in CFPS 43 lead to proposal that dates should have optional calendar style IDs attached to them in order to record important characteristics about how a source recorded a date [1]. This paper uses the term *calendar facet* where CFPS 43 used clendar style ID. Several example facets were given: `lady-day` for the use of years reckoned from Lady Day (25 March); `roman` for the Roman reckoning of days relative to the Kalends, Nones and Ides; and some for regnal years. It was noted that certain styles were orthogonal: for example, the Roman reckoning of days could be used in conjunction with regnal years.

The set of style IDs needs to be extensible to allow applications to note important characteristics of dates that may not have been standardised. As an example, in its latter years, the Ottoman Empire used a variant of the Julian calendar known as the *Rumi calendar* in which years were counted from Muhammad's emigration (or *Hijra*) from Mecca to Medina in 622 AD, with March as the first month of the year. For the purpose of the present discussion, this paper assumes that the FHISO has not standardised this calendar, but that a particular application wants to use it. For the reasons discussed in CFPS 43, the Rumi calendar should not be treated as a fundamentally different to the Julian calendar. Instead, the application would define an `anno-hegirae` facet, and then appropriately format Julian dates that are styled with it. Because the serialisation of the date would be as a standard Julian date with an *anno domini* year, other applications see a standard Julian date with an unknown facet, and can still format it and perform calculations with it (assuming they understand the Julian calendar).

If the vendor hopes that other applications will start to support the `anno-hegira` facet, its interaction with other facets must be defined. Can it be used with `lady-day`? And supposing `roman` has been developed as a third-party extension, how can the Rumi calendar vendor make an informed decision on whether `anno-hegirea` is compatible with `roman` when the creators of the two facets may not be familiar with the other one? A vocabulary is needed to define which combinations of styles are valid.

2 Facet classes

Certain facets relate only to the presentation of the year, while others relate only to the way in which the day and month are presented. This paper proposes that each facet is associated with a *facet class*, and that only one facet of a given class is permitted. Two classes are proposed here. One is called `year` and relates to the style of year used; the other is called `recurring` that relates to the part of a date that recurs every year (i.e. the day and month). Two further classes will be proposed later in this paper; they are included in the table below for convenience of reference.

Class	Default facet	Example facets
`root`	`year-and-recurring`	
`year`	`common-era`	`lady-day`, `anno-hegirae`
`recurring`	`day-in-month`	`roman`
`month-name`	`western-month`	`icelandic-month`, `hebrew-month`

New facets are also defined, one for each class, that will serve as the *default facet* of each class in the Julian and Gregorian calendars. Other calendars will define their own default facets, but it is likely that these defaults will see significant reuse. The set of default facets associated with a calendar is known as its *default style*. When no facet from a given class is supplied explicitly, the calendar's default facet for that class is used.

The `common-era` facet is used to represent the fact that the year is to be displayed as a year *anno domini* with years beginning on 1 January. The choice of facet name reflects the increasingly common practice in the historical literature to write CE in place of AD. What suffix, if any, is used in the source is not considered an important characteristic of the date. Similary, the suffix, if any, is actually used by the application to denote dates writen in this facet is up to the application.

The `day-in-month` facet denotes a date is specified using a day number together with a month. It is not considered an important characteristic of the source data whether the day was given before or after the month, or whether the month was spelt out in full, abbreviated, or given as a number.

Subject to the requirement of using only one facet from each class (and to the restriction, discussed in §5.4, that certain facets may be restricted to certain calendars), any combination of facets is valid, even though some combinations, such as `anno-hegirae` with `roman`, may be unlikely to appear in practice. Applications might support the display of dates with arbitrary combinations of facets by formatting the recurring (i.e. day and month) part of a date separately to the year and concatenating them. If an unknown facet is used, it will be ignored, probably resulting in the default facet being used, though an application might wish to indicate somehow that an unknown facet was ignored.

3 Abstract formatting model

In this section, the system of facet classes is used to describe an abstract model by which an application can format dates written in an arbitrary calendar, including calendars not known to the application. To do this, the application tries to determine the set of default facets for the calendar. (How the application does this for an unknown calendar is beyond the scope of this paper. One possibility, alluded to at the end of CFPS 43, is that the calendar ID is somehow associated with a URI which can be fetched to yield information about the calendar. Another possibility is to allow documents to contain this information, perhaps in a header declaring the calendar.)

For the purpose of this specification, LOOKUP(c, d) is a function that returns the facet of class c applying to the date d. This will either be a known facet listed in the date's style, or one of the calendar's default facets. (A future revision of CFPS 38 may provide a way of specifying default facets in a document header that override the calendar's defaults [2].) If such a facet cannot be found, for example if the calendar's default style cannot be determined or if one those facets is unknown, LOOKUP returns `null-facet`, which is defined below.

Similarly, FORMAT(f, d, s) is a function that returns the string s formatted in a manner specific to facet f in the context of formatting the date d. In an application written in an object-oriented language, FORMAT might be a virtual function on facets, called with parameters d, s and returning a string. Each facet will define the behaviour of formatting it, though lattitude is given for applications to customise the specified behaviour to allow for localisation, user preferences, context, and the display capabilities of the application. Finally, INVOKE(c, d, s) is shorthand for FORMAT(LOOKUP$(c, d), d, s$).

The formatted version of a date, d, is INVOKE(`root`, d, d). It is suggested that, if a date is expressed in a calendar other than the document's default calendar, the calendar is noted if there is scope of confusion. (In general, there is likely to be scope for confusion between two calendars having the same default style.)

4 Facet definitions

4.1 `null-facet`

This facet is referred to as `null-facet` in this specification, but in many respects it is not a real facet as it has no associated class and cannot therefore be specified explicitly, whether on a date or as part of a calendar's default style. Support for this facet is mandatory.

Format(`null-facet`, d, s) is only called as a last resort when unable to format the string s. All it can do is to return the string s, perhaps marked up in some application-defined way to indicate that it has not been understood. The facet might apply different formatting depending on context in which it was called.

4.2 `year-and-recurring`

Most world calendars are likely to use a date representation consisting of a number representing the year followed by further components that represent subdivisions of a year. (This is not required and, for example, a Mayan calendar might use five componets, the first two of which are the number of *b'ak'tuns* and *k'atuns*, units of 400 and 20 *tuns* or years, respectively. Such a calendar will generally not be able to use this facet as its `root`-class facet. Nevertheless, it is anticipated that virtually all calendars will use this facet, and there will be little need for additional `root`-class facets.) For calendars with a year-like-quantity as the highest-order component, this facet provides a way of decoupling the formatting of the its year and recurring part, which are handled by the facet classes of those names.

Format(`year-and-recurring`, d, s) splits the string s on its first '`-`' to get two parts (y, r) neither of which contains the '`-`' that separated the parts, although r may contain other instances of '`-`'. If s contains no '`-`' then $y = s$ and r is undefined. Leading 0s may be trimmed from y. The facet then formats y using Invoke(`year`, d, y), and r is formatted with Invoke(`recurring`, d, r) if r is defined. The formatted versions of y and r are concatenated in an application-defined order, perhaps with additional spacing, punctuation or markup. The result of the concatenation is returned.

4.3 `common-era`

This facet provides a simple formatting of a year, with counting starting at from 1 AD. The present definition assumes the year is serialised counting from 1 AD (as is the case in with proposals for the Gregorian [3] and Julian calendars [4]).

$\textsc{Format}($ `common-era`$, d, s\,)$ does little more than return s. It may append or prepend an abbreviation like 'AD' or 'CE'. If the application is localised for language that uses non-Arabic numeral (e.g. Bengali), then the application may need to transliterate the year number into the appropriate numerals.

4.4 `day-in-month`

This facet provides formatting of the recurring part of a date in the common case that it represented by a month and day, and the day is to be formatted as a number.

$\textsc{Format}($ `day-in-month`$, d, s\,)$ splits the string s on its last '`-`' to get two parts (m, a) neither of which contains the '`-`' that separated the parts, although m may (but generally won't) contain other instances of '`-`'. Leading `0`s may be trimmed from s and, if it's defined, a. If s has no '`-`' then $m = s$ and a is undefined. Unless the application requires a numeric representation of the month, it formats m using $\textsc{Invoke}($ `month-name`$, d, m\,)$. If a is defined, the application may do additional formatting such as appending an ordinal suffix (e.g. -st, -nd, -rd or -th in English) or transliteration. The formatted versions of m and a are concatenated in an application-defined order, perhaps with additional spacing, punctuation or markup, and the result of the concatenation is returned.

4.5 `western-month`

This facet formats the month names used by calendars derived from the ancient Roman calendar — January, February, March, April, May, June, July, August, September, October, November and December — and their variants in other languages. (This paper does not propose separate `month-name`-class facets for each language.)

$\textsc{Format}($ `western-month`$, d, s\,)$ treats s as an integer month number and usees it to look up the corresponding month name. The application may translate the month name into the user's language, and might abbreviate or format the month in an application-defined manner. If s is not an integer, the facet may show an error or delegate formatting to `null-facet`.

5 Examples

5.1 The Swedish calendar of 1700–1712

This calendar is the same as the Julian calendar except that 1700 was not a leap year and 1712 was a 'doubly-leap' year in which February had 30 days. For that twelve year peiod, Sweden was one day ahead of those parts of Europe that were still on the Julian calendar, and ten behind those that were already on the Gregorian calendar. It is assumed that this calendar will have same default style as the Gregorian or Julian calendar. How should the Swedish date `1712-02-12` be formatted?

The `root`-class facet is `year-and-recurring` which splits the date into the year, `1712`, and recurring part, `02-30`. The `year`-class facet is `common-year` which might produce '1712 CE'. The `recurring`-class facet is `day-in-month` which splits the recurring part into a month, 02, and a day, 30. The `day-in-month` looks up month 2 and returns 'Feb'. The `day-in-month` concatenates the formatted day and month to give '30 Feb', to which `year-and-recurring` appends the year resulting in '30 Feb 1712 CE'. At no point does any part of the process try to validate the date, so it never notices that '30 Feb' is a rather unusual date, even though it is valid in this particular year of this particular calendar. (Validation of a date can only be done with detailed knowledge of the calendar, which in this example the application does not have.)

5.2 Years starting on Lady Day

The example of years starting on Lady Day, 25 March, was used quite extensively in CFPS 43 which suggested defining a `lady-day` facet. It would be of the `year` class.

In this example, FORMAT(`lady-day`, d, s) would need to inspect the full date, d, to determine whether it was before or after 25 March, or whether it was a reduced representation spanning the year change. This could be done by constructing the month representation, m, by appending '`-03`' to s, which is the year. If $d = m$ or $d = s$ then the facet is formatting a reduced representation spanning the year change and needs to return a dual-year for $s - 1$ and s. If it's not a reduced representation, then the first day of the year, f, is constructed by appending '`-03-25`' to s. If $d < f$, then the either a dual-year ($s - 1$ and s) is needed or $s - 1$ can be suffixed with '(OS)', as the application vendor prefers. Otherwise, the year is s.

To give some concrete examples, the `lady-day` facet might format the year of `1710-03` as '1709/10', the year of `1710-03-13` as '1709/10' or '1709 (OS)', and the year of `1710-03-29` as '1710'.

5.3 The Old Icelandic calendar

This calendar had twelve months of 30 days, with an intercalary period called *Sumarauki* inserted after the seventh month. This either had 4 or 11 days, depending on whether it was a leap year. The representation would probably treat this as an extra month effectively giving 13 months. Such a calendar would need a custom `icelandic-month` facet of class `month-name`; very likely a general-purpose genealogy application would not support this facet.

The day and month ‘2 Sólmánuðr’ might be represented in some year with the recurring part `07-02`. The `day-in-month` facet cannot find a known facet of the `month-name` class and so falls back to using `null-facet`. That might return the month giving ‘2 7’ for the recurring part, which is not ideal. But the application is free to do better. It might be able to format the month number differently to indicate that it is unstyled: ‘2 `"07"`’. Or perhaps if `null-facet` determines that it is acting as a `month-name`-class facet, it can convert the month to a Roman numeral: ‘2 `VII`’.

5.4 The Hebrew calendar

The Hebrew calendar has twelve or thirteen months in the year, depending on whether it is a leap year. The name of the twelfth month of the year depends on whether it is a leap year or not: in a normal year it is called *Adar* (אֲדָר), but in leap year it is *Adar I* (אֲדָר א׳) and is followed by *Adar II* (אֲדָר ב׳). This is an example where the facet of the `month-name` class needs to access the full date (the d in each call to FORMAT) to determine whether it is a leap year. It can only do this if it understands the calendar. A `hebrew-month` facet therefore cannot be used with an arbitrary calendar.

5.5 Roman reckoning of days

The `roman` facet discussed in CFPS 43 is of the `recurring` class as it affects the formatting of days. As with the previous example of `hebrew-month`, the `roman` facet needs access to the full date to determine whether it is a leap year. This is necessary in order to know whether `02-25` should be *a.d. VI Kal Mar* (as it is in a leap year) or *a.d. VII Kal Mar* (in non-leap years). This means that the Roman reckoning of days can only be applied to calendars that the facet understands. When ths facet invokes the `month-name`-class facet, the month number it uses will not necessarily be the one found in the date. For dates between the Ides and Kalends, it will be one more than the month number in the representation (modulo 12), as in the example just given.

6 Arbitrary epochs

A major use of `year`-class facets is anticipated to be in defining an year-numbering scheme and stating when in the year the year increments. Such a facet is referred to as an *epoch facet*. The `common-era` facet is an epoch facet, as are the example `lady-day` and `anno-hegirae` facets. (The regnal year example in CFPS 43 is a `year`-class facet that is not an epoch facet.)

Facet name	Epoch	Usual suffix
`common-era`	0001-01-01	CE
`lady-day`	0001-03-25	CE (OS)
`anno-hegirae`	0585-03-01	AH

The *epoch* is defined as the representation of the first day in the year numbered 1. Once the epoch facet is applied to particular date, the epoch is interpretted as a date in that calendar. This allows the `lady-day` facet to represent years counted from 25 March, irrespective of whether it is applied to a Julian or Gregorian date (or, indeed, a Swedish one). If the epoch is not a valid date in the calendar in question, then the facet cannot be applied to that calendar.

The epoch for `anno-hegirae`, as used by the Rumi calendar, is listed as 585 AD, even though it is nominally a number of years since 622 AD. This is because, for the first six hundred years, counting was done using a lunar calendar with years that were, on average, rather longer than the Julian year. By the thirteenth century AD, the difference between the years *anno domini* and *anno hegirae* was 584 years, meaning that 1 AH of the proleptic Rumi calendar started during 585 AD. The `03-01` simply reflects that the year began on 1 March.

An algorithm is given below for calculating the year with respect to an arbitrary epoch in an arbitrary calendar. The `common-era` and `lady-day` facets should behave the same if they are formatted using this algorithm as with the special-case algorithms in §4.3 and §5.2, respectively. It is hoped that if applications use a general epoch algorithm, they can format dates using arbitrary epoch facets, providing they are able to discover the associated epoch and are provided with a hint as to an appropriate suffix to use.

At present this mechanism does not cope with epochs that start before the year 0000 in the calendar representation. This is relevant to several common calendars including the Hebrew calendar which expresses years *anno mundi*, with an epoch in 3761 BC This paper does not consider how to handle such epochs. However it is noted that if the representation of dates is extended to somehow allow negative years, this will be solved.

6.1 Algorithm

If f is an epoch facet with an epoch of e, then $\textsc{Format}(f, d, s)$ is defined as follows. If $d < e$ then the date being formatted predates the epoch, and the application may show an error, if not it displays the date in some application-defined manner. The epoch year and epoch's recurring part, (y, r), are extracted from e by splitting it at the first '`-`'.

It is first necessary to test whether the full date, d, is a reduced representation spanning the start of a year. For every instance of a '`-`' in r, the substring preceding the '`-`' is considered. Call this substring n. A string, m, is formed by the concatenation of s, '`-`' and n. If at any point $d = m$, the date is reduced representation spanning the start of a year. If so, then a dual-year format is needed for $(s - y, s - y + 1)$. Otherwise, continue on to the next '`-`' in r until there are no more.

The dual-year format of (x, y) is typically done by taking x, and following it by a separator such as '`/`'. If x and y share a common leading substring, that substring may be removed from y before y is appended to the result.

If it's not a reduced representation, then the first day of the year, f, is constructed by concatenating s, '`-`' and r. If $d < f$, then the year to be formatted is $s - y$; otherwise it is $s - y + 1$.

Whether it was a single or a dual year, a suffix (such as the usual one provided in the table) may be appended or prepended, and any transliteration of numerals required for localisation can be done.

7 Concluding remarks

The proposals in this paper can be considered as three separate proposals, each dependent on the earlier ones. The first was the introduction of the `year` and `recurring` facet classes (and the associated `common-era` and `day-in` facets) as a vocabulary for use in defining third-party facets. With just this part, the facets are little more than opaque tags each of which needs application support.

The second proposal was the use of facet and class mechansim into a model for the formatting of dates in arbitrary calendars. This completes the separation of the formatting of a calendar from its mechanics (i.e. the details of how many days are in which months, and so on). There is no good reason why an application should need to understand about the unqiue status of Sweden's 30 Feb 1712 just so that it can format it.

The third part was the mechanism for defining epoch facets. An application supporting this is not only able to handle some unfamiliar calendars, but also some unfamiliar facets. This paper has not discussed how an application should go about discovering the basic details of unknown calendars or facets, but the use of QNames as calendar IDs (and presumably facet IDs) suggested in CFPS 37 results in a URI being associated with every calendar and facet [5]. An application could use this URI to fetch information about unknown calendars and facets.

It is easy to see how many `month-name`-class facets are little more than lookup tables for month names, and that with such an discovery protocol in place, unknown facets of this type could be handled similarly to unknown epoch facets.

References

[1] Richard Smith, 2013, *Proposal to add style to the wholly-numeric representation of dates in CFPS 13* (CFPS 43), `http://fhiso.org/files/cfp/cfps43.pdf`

[2] Richard Smith, 2013, *Proposal for compound calendars to resolve a difficulty with default calendars* (CFPS 38), `http://fhiso.org/files/cfp/cfps38.pdf`

[3] Tony Proctor, 2013, *Proposal to Accommodate Gregorian Dates using a Modified ISO 8601* (CFPS 17), `http://fhiso.org/files/cfp/cfps17.pdf`

[4] Richard Smith, 2013, *Proposal to support the Julian calendar similarly to CFPS 17* (CFPS 44), `http://fhiso.org/files/cfp/cfps44.pdf`

[5] Richard Smith, 2013, *Proposal for a scalable extensibility mechanism* (CFPS 37), `http://fhiso.org/files/cfp/cfps37.pdf`