The kalendarium Package

Andrew Smith*

Version v1.0 — Released A.D. XV Kal. Iul. MMDCCLXXI A.U.C.

Abstract

The kalendarium package provides several macros with which to print dates in classical Latin given days on the Julian or Gregorian calendars, using the same syntax used by ancient Roman authors. The format of these dates may be customised either in the package options or on a per-command basis; these options also allow for the generation of date strings according to different eras of the Classical period.

1 Prerequisites

kalendarium is written in LaTeX3, and thus requires xparse and the LaTeX3 kernel to be installed. Furthermore, in order to parse key-value package and command options, the package l3keys2e is also required.

2 Usage and configuration

Most simply, kalendarium may be imported with default formatting in the usual fashion, with `\usepackage{kalendarium}`. Global formatting options may be set on import via key-value pairs; for example, to display all dates with no abbreviation, numbers fully written out in Latin, and years calculated *ab urbe condita*, this import string may be used:

```
\usepackage[abbreviate=false,dayfmt=latin,era=auc]{kalendarium}
```

A complete list of formatting options may be found in Section 5.1.

3 The Roman calendar

The kalendarium package provides utilities to format dates according to its namesake, the ancient Roman *kalendarium*.[1] While in early Rome, the calendar was subject to great variation at the discretion of the high priests of the *religio Romana*, this package implements the more standardised version used after the Julian reforms of 46 BCE. In addition, it provides formatting options to display dates according to even more modern standards, such as writing years *anno Domini* instead of *ab urbe condita* and avoiding the *bissextum Februarium*. This section describes the basics of the *kalendarium* and some historical variations thereupon.

*Email: aws@awsmith.us

[1] This package does not relate in any way to the Catholic liturgical General Roman Calendar.

3.1 Roman months

Prior to the Roman Republic, a ten-month calendar ascribed to Romulus is reported to have been used. Since this calendar had already fallen out of use by the Classical period and it required the insertion of fifty-one intercalary days between and during months—sometimes methodically, but often at random—utilities for writing dates according to the Romulan calendar are not provided by this package.

Following the Tarquin expulsion, this legendary calendar was replaced with a twelve-month system with an intercalary month inserted between the *Terminalia* and *Regifugium* (23 and 24 February, a newly-added month).[2] While this iteration of the calendar resolved many of the issues present in the Romulan calendar, it still did not accurately portray the length of a year, having only 354 days per year, and the intercalary month was still somewhat unstable.

In 46 BCE, the victorious *pontifex maximus* Julius Caesar conducted further calendrical reforms. The thus-named Julian calendar distributed the days of the intercalary month amongst the other months of the year, instituting the month lengths still used today. Furthermore, to compensate for the fact that the length of a year cannot be described in a perfect integer number of days, Caesar's calendar inserted an intercalary day (*bissextum*) between the *Terminalia* and *Regifugium* every four years.[3] Thus, a somewhat modern calendar was achieved:

Month		Days in month		
		Romulan	Republican	Julian
March	*Martius*	31	31	31
April	*Aprilis*	30	29	30
May	*Maius*	31	31	31
June	*Iunius*	30	29	30
July	*Quintilis*	31	31	31
August	*Sextilis*	30	29	31
September	*September*	30	29	30
October	*October*	31	31	31
November	*November*	30	29	30
December	*December*	30	29	31
January	*Ianuarius*		29	31
February	*Februarius*		28	28 *or* 29
Intercalary	*Mercedonius*	51	22/23	*None*

Following the assassination of Julius Caesar in 44 BCE, Mark Antony renamed the month of *Quintilis* to *Iulius* (July) in honour of the late emperor. Furthermore, following the death of Augustus, the month *Sextilis* was renamed *Augustus* (August) according to a referendum in 8 BCE.[4]

As a further refinement, in the year 1582 CE, Pope Gregory XIII added additional criteria for the calculation of *bissexta*, or leap years: an additional day was to be added to the month of February if the year was divisible by 4, but not if it was divisible by 100, yet still if it was divisible by 400.[5] Adopted throughout nearly all of the

[2] Plutarch, *Parallel Lives* 18.
[3] Macrobius, *Saturnalia* I.14.
[4] Macrobius, *Saturnalia* I.12.
[5] Pope Gregory XIII, *Inter gravissimas*.

modern world, this Gregorian calendar is used in all Western nations; nevertheless, for historical compatibility, this package provides support for converting dates on both the Julian and Gregorian calendars.

3.2 Roman days

Each Roman month was ascribed three named days: the Kalends, the Nones, and the Ides. The Kalends (*Kalendae*, *Kal.*) fell on the first day of every month, without variation. The Ides (*Idus*, *Id.*) fell on the fifteenth day of "full" months—months which comprised 31 days on the Romulan calendar, thus March, May, July/Quintilis, and October—and the thirteenth day of the other "hollow" months. Finally, the Nones (*Nonae*, *Non.*) fell eight (nine inclusively) days before the Ides, on the seventh day of full months and the fifth day of hollow ones.

Roman days were counted inclusively, meaning that the current day was taken into consideration when calculating the number of days between two dates. Thus, a Roman would state that 3 January is not two days after 1 January, but three. This system of inclusive counting appears throughout the *Kalendarium*, in which dates are expressed in relation to the next closest Nones or Ides of the current month, or the Kalends of the following month. An exception occurs for the day immediately preceding one of these named days, which is referenced only as *pridie*. The conjugate term for the day following, *postridie*, does not appear in the Roman calendar.

3.3 Roman weeks

During the monarchic and republican periods, Roman weeks were delineated in eight-day cycles, called *nundina* (sg. *nundinum*) on account of their inclusive counting of calendar days. Under this nundinal system, the days of the week were assigned letters A–H, with public markets and festivals occurring on the *nundinae* and not again for the next eight days. These market-days acted as a form of weekend for Roman workers, with farmers breaking from their work to travel into the city, sell their wares, and buy provisions to last through the next *nundinum*.[6]

In the early imperial period, the nundinal cycle was replaced by the seven-day week (*hebdomas*), influenced by the influx of eastern religion and philosophy to the western Roman empire. While these hebdomadal days continued to be assigned letters A–G, they were also named after pagan gods and goddesses in the fashion of the Greeks:

Modern	Latin		Namesake
Sunday	*dies Solis*	☉	Sol (the Sun)
Monday	*dies Lunae*	☾	Luna (the Moon)
Tuesday	*dies Martis*	♂	Mars
Wednesday	*dies Mercurii*	☿	Mercury
Thursday	*dies Iovis*	♃	Jupiter (GEN. *Iovis*)
Friday	*dies Veneris*	♀	Venus (GEN. *Veneris*)
Saturday	*dies Saturni*	♄	Saturn

Like many other facets of the Roman calendar, these pagan references survived the Christianisation of Rome and continue to appear in the lexica of the modern Romance languages.

[6] Macrobius, *Saturnalia* I.16.

3.4 Roman years

Methods used to refer to specific years on the Roman calendar have varied throughout history, beginning with regnal references to the acting king, ordinary consuls, or Diocletian Indiction. While not ubiquitous in Classical texts, the *ab urbe condita* (AUC, "from the founding of the city (of Rome)") reckoning, which measures dates in terms of their distance from the legendary founding of Rome in 753 BCE counted inclusively, has been widely accepted amongst classicists. In this era, the year 753 BCE would be rendered 1 AUC, and 2018 CE would be 2771 AUC (2018 + 753).

In 525 CE, the monk Dionysius Exiguus devised the *anno Domini* (AD, "in the year of the Lord") era, which measures years non-inclusively after the birth of Jesus of Nazareth.[7] To represent dates before this event, the era *ante Christum* (AC, "before (the Nativity of) Christ") was popularised by theologian Dionysius Petavius in 1627 CE.[8] These eras are still frequently used in the modern Gregorian calendar, although the non-sectarian Common Era (CE) replacements have become somewhat common. Furthermore, the phrase *anno Domini* is sometimes shortened to only *anno* (A).

3.5 Latin date syntax

The syntax used in Latin date expressions is rather peculiar, albeit almost entirely formulaic. Unlike in many modern languages, the names of months were not considered to be nouns, but were adjectives, requiring agreement with their antecedents; additionally, certain phrases were fossilised to the point that they began to be used in expressions that would otherwise be considered ungrammatical.

Dates on the named days of each month, which were always plural, were expressed independently using the ablative case of time when ("on the…"). Thus, they would appear as *Kalendis Februariis*, *Nonis Septembribus*, *Idibus Iuniis*, etc. While this syntax was common for all date expressions in early Latin, Classical authors often opted for another approach for non-named days.

Dates that did not fall on the Kalends, Nones, or Ides were expressed in terms of their relation to the next named day. Thus, a date after the Kalends of a month but before the Nones would be named by the number of days, counted inclusively, until the Nones; a date after the Nones but before the Ides, by the days until the Ides; and a date after the Ides, by the days until the Kalends of the following month. These numbers of days were often preceded by the expression *ante diem* (A.D.) and followed by their respective named day and month in the accusative case, plural. An exception to this format occurred for days immediately preceding a named day, which were expressed as *pridie* (*prid.*), then the named day and month in the accusative plural.

A further exception occurred for the month of February in leap years. Leap days were considered to be a single day under Roman law, and were treated as such when counting days until the next Kalends. When a leap day was inserted into the month, it was known as the *bissextum* and written *ante diem bis VI Kalendas Martias*; the normal day five days prior to the end of the month was still written *ante diem VI Kalendas Martias*.

Counted numbers of days were often written with Roman numerals, as above; however, they were sometimes written out as full ordinal numbers in the accusative case, masculine singular, in agreement with *diem*. For further elucidation, some months are written in full on the following page.

[7] Dionysius Exiguus, Easter tables.
[8] Dionysius Petavius (Denis Péteau), *De doctrina temporum*.

	January		February		*Full*	*Hollow*
Day	Full Latin	Abbreviated	Regular year	Leap year	March	April
1	Kalendis Ianuariis	Kal. Ian.	Kal. Feb.	Kal. Feb.	Kal. Mar.	Kal. Apr.
2	ante diem quartum Nonas Ianuarias	A.D. IV Non. Ian.	A.D. IV Non. Feb.	A.D. IV Non. Feb.	A.D. VI Non. Mar.	A.D. IV Non. Apr.
3	ante diem tertium Nonas Ianuarias	A.D. III Non. Ian.	A.D. III Non. Feb.	A.D. III Non. Feb.	A.D. V Non. Mar.	A.D. III Non. Apr.
4	pridie Nonas Ianuarias	prid. Non. Ian.	prid. Non. Feb.	prid. Non. Feb.	A.D. IV Non. Mar.	prid. Non. Apr.
5	Nonis Ianuariis	Non. Ian.	Non. Feb.	Non. Feb.	A.D. III Non. Mar.	Non. Apr.
6	ante diem octavum Idus Ianuarias	A.D. VIII Id. Ian.	A.D. VIII Id. Feb.	A.D. VIII Id. Feb.	prid. Non. Mar.	A.D. VIII Id. Apr.
7	ante diem septimum Idus Ianuarias	A.D. VII Id. Ian.	A.D. VII Id. Feb.	A.D. VII Id. Feb.	Non. Mar.	A.D. VII Id. Apr.
8	ante diem sextum Idus Ianuarias	A.D. VI Id. Ian.	A.D. VI Id. Feb.	A.D. VI Id. Feb.	A.D. VIII Id. Mar.	A.D. VI Id. Apr.
9	ante diem quintum Idus Ianuarias	A.D. V Id. Ian.	A.D. V Id. Feb.	A.D. V Id. Feb.	A.D. VII Id. Mar.	A.D. V Id. Apr.
10	ante diem quartum Idus Ianuarias	A.D. IV Id. Ian.	A.D. IV Id. Feb.	A.D. IV Id. Feb.	A.D. VI Id. Mar.	A.D. IV Id. Apr.
11	ante diem tertium Idus Ianuarias	A.D. III Id. Ian.	A.D. III Id. Feb.	A.D. III Id. Feb.	A.D. V Id. Mar.	A.D. III Id. Apr.
12	pridie Idus Ianuarias	prid. Id. Ian.	prid. Id. Feb.	prid. Id. Feb.	A.D. IV Id. Mar.	prid. Id. Apr.
13	Idibus Ianuariis	Id. Ian.	Id. Feb.	Id. Feb.	A.D. III Id. Mar.	Id. Apr.
14	ante diem undevicesimum Kalendas Februarias	A.D. XIX Kal. Feb.	A.D. XVI Kal. Mar.	A.D. XVI Kal. Mar.	prid. Id. Mar.	A.D. XVIII Kal. Mai.
15	ante diem duodevicesimum Kalendas Februarias	A.D. XVIII Kal. Feb.	A.D. XV Kal. Mar.	A.D. XV Kal. Mar.	Id. Mar.	A.D. XVII Kal. Mai.
16	ante diem septimum decimum Kalendas Februarias	A.D. XVII Kal. Feb.	A.D. XIV Kal. Mar.	A.D. XIV Kal. Mar.	A.D. XVII Kal. Apr.	A.D. XVI Kal. Mai.
17	ante diem sextum decimum Kalendas Februarias	A.D. XVI Kal. Feb.	A.D. XIII Kal. Mar.	A.D. XIII Kal. Mar.	A.D. XVI Kal. Apr.	A.D. XV Kal. Mai.
18	ante diem quintum decimum Kalendas Februarias	A.D. XV Kal. Feb.	A.D. XII Kal. Mar.	A.D. XII Kal. Mar.	A.D. XV Kal. Apr.	A.D. XIV Kal. Mai.
19	ante diem quartum decimum Kalendas Februarias	A.D. XIV Kal. Feb.	A.D. XI Kal. Mar.	A.D. XI Kal. Mar.	A.D. XIV Kal. Apr.	A.D. XIII Kal. Mai.
20	ante diem tertium decimum Kalendas Februarias	A.D. XIII Kal. Feb.	A.D. X Kal. Mar.	A.D. X Kal. Mar.	A.D. XIII Kal. Apr.	A.D. XII Kal. Mai.
21	ante diem duodecimum Kalendas Februarias	A.D. XII Kal. Feb.	A.D. IX Kal. Mar.	A.D. IX Kal. Mar.	A.D. XII Kal. Apr.	A.D. XI Kal. Mai.
22	ante diem undecimum Kalendas Februarias	A.D. XI Kal. Feb.	A.D. VIII Kal. Mar.	A.D. VIII Kal. Mar.	A.D. XI Kal. Apr.	A.D. X Kal. Mai.
23	ante diem decimum Kalendas Februarias	A.D. X Kal. Feb.	A.D. VII Kal. Mar.	A.D. VII Kal. Mar.	A.D. X Kal. Apr.	A.D. IX Kal. Mai.
24	ante diem nonum Kalendas Februarias	A.D. IX Kal. Feb.	A.D. VI Kal. Mar.	A.D. bis VI Kal. Mar.	A.D. IX Kal. Apr.	A.D. VIII Kal. Mai.
25	ante diem octavum Kalendas Februarias	A.D. VIII Kal. Feb.	A.D. V Kal. Mar.	A.D. VI Kal. Mar.	A.D. VIII Kal. Apr.	A.D. VII Kal. Mai.
26	ante diem septimum Kalendas Februarias	A.D. VII Kal. Feb.	A.D. IV Kal. Mar.	A.D. V Kal. Mar.	A.D. VII Kal. Apr.	A.D. VI Kal. Mai.
27	ante diem sextum Kalendas Februarias	A.D. VI Kal. Feb.	A.D. III Kal. Mar.	A.D. IV Kal. Mar.	A.D. VI Kal. Apr.	A.D. V Kal. Mai.
28	ante diem quintum Kalendas Februarias	A.D. V Kal. Feb.	prid. Kal. Mar.	A.D. III Kal. Mar.	A.D. V Kal. Apr.	A.D. IV Kal. Mai.
29	ante diem quartum Kalendas Februarias	A.D. IV Kal. Feb.		prid. Kal. Mar.	A.D. IV Kal. Apr.	A.D. III Kal. Mai.
30	ante diem tertium Kalendas Februarias	A.D. III Kal. Feb.			A.D. III Kal. Apr.	prid. Kal. Mai.
31	pridie Kalendas Februarias	prid. Kal. Feb.			prid. Kal. Apr.	

4 Basic commands

4.1 Writing a date

\KalDate ⋆ `\KalDate[⟨options⟩]{⟨year⟩}{⟨month⟩}{⟨day⟩}`
\kal_date:nnn ⋆ `\kal_date:nnn {⟨year⟩} {⟨month⟩} {⟨day⟩}`

This command renders the specified date according to the Roman calendar. Regardless of century, the ⟨*year*⟩ must be specified in full: `\KalDate{18}{6}{14}` will result in the year 18 CE, not 2018. If ⟨*options*⟩ are specified, they will apply only to the date rendered by their respective instance of this command.

Example

Here, we render the date 4 July 2018 with default formatting (no abbreviations, day written in Roman numerals, *anno Domini* era):

`\KalDate{2018}{7}{4}` ⇒ ante diem IV Nonas Iulias anno Domini MMXVIII

4.2 Writing a date from a string

\KalDateStr ⋆ `\KalDateStr[⟨options⟩]{⟨format⟩}{⟨delimiter⟩}{⟨date string⟩}`
\kal_date_string:nnn ⋆ `\kal_date_string:nnn {⟨format⟩} {⟨delimiter⟩} {⟨date string⟩}`
\kal_date_string:nno ⋆

While the previous command works well for predefined, manually-typed dates, dates generated by macros nearly always appear in the form of a single string, not three separate arguments. For such scenarios, **kalendarium** provides the `\KalDateStr` command, which converts a ⟨*date string*⟩ in the specified ⟨*format*⟩ with segments separated by the ⟨*delimiter*⟩ into a Roman date. The ⟨*format*⟩ should consist of a maximum of three characters—`Y` for the year, `M` for the month, and `D` for the day of the month—in the order in which they appear in the ⟨*date string*⟩.

Example

Here, we render the date 4 July 2018 from ISO format (`YYYY-MM-DD`) and standard United States format (`M/D/YYYY`) in the Roman calendar:

`\KalDateStr{YMD}{-}{2018-07-04}` ⇒
ante diem IV Nonas Iulias anno Domini MMXVIII
`\KalDateStr{MDY}{/}{7/4/2018}` ⇒
ante diem IV Nonas Iulias anno Domini MMXVIII

When using the standard LaTeX command `\KalDateStr`, the ⟨*date string*⟩ is expanded once to allow for the use of macros as input; this is equivalent to the LaTeX3 command `\kal_date_string:nno`.

Example

Here, we render the date on which the **kalendarium** style file was last generated according to the `\filedate` macro (⇒ `2018-06-17`) in the Roman calendar:

`\KalDateStr{YMD}{-}{\filedate}` ⇒
ante diem XV Kalendas Iulias anno Domini MMXVIII

4.3 Writing the current date

`\KalToday` ⋆ `\KalToday[⟨options⟩]`

This command renders the current date according to the Roman calendar, serving as the **kalendarium** equivalent of the `\today` macro.

> **Example**
>
> Here, we render the date on which this document was generated (July 5, 2018) in the Roman calendar:
>
> `\KalToday` ⇒ ante diem III Nonas Iulias anno Domini MMXVIII

4.4 Determining the weekday of a date

`\KalWeekday` ⋆ `\KalWeekday[⟨options⟩]{⟨year⟩}{⟨month⟩}{⟨day⟩}`
`\kal_weekday:nnn` ⋆ `\kal_weekday:n {⟨year⟩} {⟨month⟩} {⟨day⟩}`

This command determines the day of the week of a given day and displays its Latin name. Like the rest of **kalendarium**, it can handle dates on both the Julian and Gregorian calendars and takes the same set of ⟨*options*⟩ as other commands.

> **Example**
>
> Here, we determine the day of the week on which 4 July 2018 will fall:
>
> `dies \KalWeekday{2018}{7}{4}` ⇒ dies Mercurii
> `dies \KalWeekday[abbreviate=true]{2018}{7}{4}` ⇒ dies Merc.

5 Customisation

5.1 Format options

In order to provide the ability to render dates on the Roman calendar through various ages of the Classical period, **kalendarium** provides many options to customise the format in which dates are displayed. These options may be set in the global package options (`\usepackage` statement) or for individual commands via the ⟨*options*⟩ argument. Regardless of which means of configuration is used, options are set via comma-separated key-value pairs of the form `[`⟨*key*$_1$⟩`=`⟨*value*$_1$⟩`,`⟨*key*$_2$⟩`=`⟨*value*$_2$⟩`,...]`, in which ⟨*key*$_n$⟩ is any of the options in the left margin of this section and ⟨*value*$_n$⟩ is one of the accepted choices listed thereby. Values with a 🔔 blue bell to their left are the default values for their respective options.

`abbreviate`

This variable determines whether to use common abbreviations (A.D. for *ante diem*, Kal. for *Kalendas*, etc.) instead of full Latin phrases when displaying dates.

Accepted values

	`true`	Display dates in abbreviated format.
🔔	`false`	Display dates in full, unabbreviated Latin.

Example

Here, we render the date 4 July 2018, first unabbreviated, then with the `abbreviate` option set to `true`:

```
\KalDate[abbreviate=false]{2018}{7}{4} ⇒
```
ante diem IV Nonas Iulias anno Domini MMXVIII
```
\KalDate[abbreviate=true]{2018}{7}{4} ⇒
```
A.D. IV Non. Iul. A.D. MMXVIII
ante diem *Nonas* *Iulias* *anno Domini*

`antediem`

This variable determines whether to display the day portion of dates in the Classical *ante diem* style or in the Medieval ablative ordinal style.

Accepted values

🔔	`true`	Display days in Classical style, prefixed by *ante diem* (A.D.) and as an ordinal in the accusative case, when appropriate.
	`false`	Display days in Medieval style, with no prefix and as an ordinal in the ablative case, when appropriate.

Example

Here, we render the date 4 May 1886 in both the Classical and Medieval styles:

```
\KalDate[antediem=true]{1886}{5}{4} ⇒
```
ante diem IV Nonas Maias anno Domini MDCCCLXXXVI
```
\KalDate[antediem=false]{1886}{5}{4} ⇒
```
IV Nonas Maias anno Domini MDCCCLXXXVI

`bissextum`

This variable determines whether to display dates in February of leap years with a *bissextum* on 24 February or to continue the counting of days through the *Terminalia*.

Accepted values

🔔	`true`	Display 24 February as *a.d. bissextum Kalendas Martias* in leap years.
	`false`	Display 24 February as *a.d. septimum Kalendas Martias* in leap years.

Example

Here, we render the date 24 February 2016, first with a *bissextum* then without:

```
\KalDate[bissextum=true]{2016}{2}{24} ⇒
    ante diem bis VI Kalendas Martias anno Domini MMXVI
\KalDate[bissextum=false]{2016}{2}{24} ⇒
    ante diem VII Kalendas Martias anno Domini MMXVI
```

Example

Here, we render the same dates, but with Latin ordinal numbers instead of Roman numerals:

```
\KalDate[bissextum=true,dayfmt=latin]{2016}{2}{24} ⇒
    ante diem bissextum Kalendas Martias anno Domini MMXVI
\KalDate[bissextum=false,dayfmt=latin]{2016}{2}{24} ⇒
    ante diem septimum Kalendas Martias anno Domini MMXVI
```

`dayfmt`

This variable determines the format in which to display the day portion of an *ante diem* date. If an invalid value is set, days will be displayed in Roman numerals, as is default.

Accepted values

🔔	`roman`	Display the day in Roman numerals.
	`latin`	Display the day in full Latin text. If the option `antediem` is set to `true`, the ordinal number will be in the accusative singular; otherwise, it will be in the ablative, in Medieval style.
	`arabic`	Display the day in Arabic numerals.

Example

Here, we render the date 27 November 1095 with all supported day formats:

```
\KalDate[dayfmt=roman]{1095}{11}{27} ⇒
    ante diem V Kalendas Decembres anno Domini MXCV
\KalDate[dayfmt=latin]{1095}{11}{27} ⇒
    ante diem quintum Kalendas Decembres anno Domini MXCV
\KalDate[dayfmt=latin,antediem=false]{1095}{11}{27} ⇒
    quinto Kalendas Decembres anno Domini MXCV
\KalDate[dayfmt=arabic]{1095}{11}{27} ⇒
    ante diem 5 Kalendas Decembres anno Domini MXCV
```

`era` This variable determines the era in which to display the specified year, or the prefix or suffix to be placed around the year number.

Accepted values

🔔 `ad`	Display years in *anno Domini* (A.D.) and *ante Christum* (A.C.) format.
`adshort`	Same as `ad`, except *anno Domini* is shortened to only *anno* (A.).
`auc`	Display years *ab urbe condita* (A.U.C.), the time since the legendary founding of the city of Rome in 753 BCE.
`secular`	Display years in a Latinised form of the secular Before Common Era and Common Era format, translated to *ante saeculum commune* (A.S.C.) and *saeculo communi* (S.C.), respectively.
`none`, `*`	Do not display the year when rendering dates.

Example

Here, we render the date 7 November 1917 in all supported eras:

```
\KalDate[era=auc]{1917}{11}{7} ⇒
    ante diem VII Idus Novembres MMDCLXX ab urbe condita
\KalDate[era=ad]{1917}{11}{7} ⇒
    ante diem VII Idus Novembres anno Domini MCMXVII
\KalDate[era=adshort]{1917}{11}{7} ⇒
    ante diem VII Idus Novembres anno MCMXVII
\KalDate[era=secular]{1917}{11}{7} ⇒
    ante diem VII Idus Novembres saeculo communi MCMXVII
\KalDate[era=none]{1917}{11}{7} ⇒
    ante diem VII Idus Novembres
```

`julian` This variable determines whether to determine leap years and weekdays according to the Julian or Gregorian calendar.

Accepted values

`true`	Perform date calculations according to the Julian calendar.
🔔 `false`	Perform date calculations according to the Gregorian calendar.

Example

Here, we render the date 24 February 1900 on both the Julian and Gregorian calendars. On the Julian calendar, 1900 is considered a leap year, and thus 24 February is a *bissextum*; however, on the Gregorian calendar, it is not.

```
\KalDate[julian=true]{1990}{2}{24} ⇒
    ante diem bis VI Kalendas Martias anno Domini MCMXC
\KalDate[julian=false]{1990}{2}{24} ⇒
    ante diem VI Kalendas Martias anno Domini MCMXC
```

`oldmonths`

This variable determines whether to display the Imperial names for the seventh and eighth months of the year, *Iulius* and *Augustus*, or the (pre-)Republican names, *Quintilis* and *Sextilis*.

Accepted values

	`true`	Display month names as in the Republican period and prior.
🔔	`false`	Display month names as in the Imperial period and after.

Example

Here, we render the date 4 July 2018, first with its Republican name *Quintilis*, then with its Imperial name *Iulius*:

```
\KalDate[oldmonths=true]{2018}{7}{4} ⇒
    ante diem IV Nonas Quintiles anno Domini MMXVIII
\KalDate[oldmonths=false]{2018}{7}{4} ⇒
    ante diem IV Nonas Iulias anno Domini MMXVIII
```

`periods`

This variable determines whether to display periods to indicate abbreviations for common calendrical phrases. If the `abbreviate` option is set to `false`, this option will have no effect on the date format.

Accepted values

🔔	`true`	Display periods to indicate abbreviated terms.
	`false`	Do not display any periods amidst abbreviations.

Example

Here, we render the date 25 December 1776, first with periods, then with the `periods` option set to `false`:

```
\KalDate[abbreviate=true]{1776}{12}{25} ⇒
    A.D. VIII Kal. Ian. A. MDCCLXXVI
\KalDate[abbreviate=true,periods=false]{1776}{12}{25} ⇒
    AD VIII Kal Ian A MDCCLXXVI
```

`yearfmt` This variable determines the format in which to display the numeric year of a date for which the option `era` is not set to `none`. If an invalid value is set, years will be displayed in Roman numerals, as is default.

Accepted values

`roman`	Display the year in Roman numerals.
`arabic`	Display the year in Arabic numerals.

Example

Here, we render the date 27 November 1095 with all supported year formats:

```
\KalDate[yearfmt=roman]{1095}{11}{27} ⇒
    ante diem V Kalendas Decembres anno Domini MXCV
\KalDate[yearfmt=arabic]{1095}{11}{27} ⇒
    ante diem V Kalendas Decembres anno Domini 1095
```

5.2 Formatting commands

In order to provide even more control over the formatting of Roman calendar dates, **kalendarium** provides several commands that may be overridden to change the visual appearance of its output. By default, all Roman numerals are output in lowercase, so these formatting commands must perform any desired case transformations in addition to styling operations.

`\KalAbbrFormat` ⋆ `\KalAbbrFormat{⟨abbreviation⟩}`

This command controls the display of all single-letter abbreviations within date expressions. If the option `abbreviate` is set to `false`, it will have no apparent effect on the output of any command.

Example

Here, we override `\KalAbbrFormat` to render all abbreviations in italics instead of the default small capitals:

```
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    A.D. IV Non. Iul. A. MMXVIII
\renewcommand{\KalAbbrFormat}[1]{\textit{#1}}
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    a.d. IV Non. Iul. a. MMXVIII
```

\KalDayFormat ⋆ `\KalDayFormat{⟨day⟩}`

This command controls the display of all day numbers within date expressions. By default, it only serves to capitalise Roman numerals.

Example

Here, we override `\KalDayFormat` to underline and capitalise all day numbers:

```
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    A.D. IV Non. Iul. A. MMXVIII
\renewcommand{\KalDayFormat}[1]{\underline{\MakeUppercase{#1}}}
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    A.D. IV Non. Iul. A. MMXVIII
```

\KalYearFormat ⋆ `\KalYearFormat{⟨year⟩}`

This command controls the display of all year numbers within date expressions. By default, it only serves to capitalise Roman numerals.

Example

Here, we override `\KalYearFormat` to overline all year numbers and render them in small capitals:

```
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    A.D. IV Non. Iul. A. MMXVIII
\renewcommand{\KalYearFormat}[1]{$\overline{\mbox{\textsc{#1}}}$}
\KalDate[abbreviate=true,era=adshort]{2018}{7}{4} ⇒
    A.D. IV Non. Iul. A. MMXVIII
```

6 Implementation

6.1 Package declaration and dependencies

```
⟨*package⟩
\RequirePackage{xparse}
\RequirePackage{l3keys2e}
\ProvidesExplPackage
  {kalendarium}
  {2018-06-17}{1.0}
  {A package to display dates in the classical Roman style}
\ExplSyntaxOn
```

6.2 Package options

Process global options immediately upon loading the package, setting default values for any options not specified by the user.

```
\keys_define:nn {kalendarium}{
  abbreviate .bool_set:N = \l_kal_abbreviate_bool,
  abbreviate .initial:n  = { false },
  periods    .bool_set:N = \l_kal_periods_bool,
```

```
  periods    .initial:n  = { true },
  era        .tl_set:N   = \l_kal_era_tl,
  era        .initial:n  = { ad },
  julian     .bool_set:N = \l_kal_julian_bool,
  julian     .initial:n  = { false },
  dayfmt     .tl_set:N   = \l_kal_dayfmt_tl,
  dayfmt     .initial:n  = { roman },
  yearfmt    .tl_set:N   = \l_kal_yearfmt_tl,
  yearfmt    .initial:n  = { roman },
  antediem   .bool_set:N = \l_kal_antediem_bool,
  antediem   .initial:n  = { true },
  bissextum  .bool_set:N = \l_kal_bissextum_bool,
  bissextum  .initial:n  = { true },
  oldmonths  .bool_set:N = \l_kal_oldmonths_bool,
  oldmonths  .initial:n  = { false }
}
\ProcessKeysOptions {kalendarium}
```

6.3 Constant lists

In order to convert numeric dates to written Latin, lookup lists are allocated for each required conversion. Thus, not all genders, numbers, and cases of adjectives are listed, but only those that regularly appear in Roman date strings.

`\c_kal_day_acc_clist` Ordinal numbers 1–19 in the accusative singular masculine; the *bissextum*, while not the ordinal number 20, is allocated the twentieth position in this and other ordinal lists for the sake of convenience. Furthermore, the ordinals *primum* and *secundum* do not appear in date strings but are included nonetheless to facilitate index lookups.

```
\clist_new:N  \c_kal_day_acc_clist
\clist_set:Nn \c_kal_day_acc_clist {
  primum,secundum,tertium,quartum,quintum,sextum,septimum,octavum,nonum,decimum,
  undecimum,duodecimum,tertium\ decimum,quartum\ decimum,quintum\ decimum,
  sextum\ decimum,septimum\ decimum,duodevicesimum,undevicesimum,bissextum
}
```

(*End definition for* `\c_kal_day_acc_clist`.)

`\c_kal_day_abl_clist` Ordinal numbers 1–19 in the ablative singular masculine; used only for the Medieval day format that forgoes the prefix *ante diem*.

```
\clist_new:N  \c_kal_day_abl_clist
\clist_set:Nn \c_kal_day_abl_clist {
  primo,secundo,tertio,quarto,quinto,sexto,septimo,octavo,nono,decimo,
  undecimo,duodecimo,tertio\ decimo,quarto\ decimo,quinto\ decimo,
  sexto\ decimo,septimo\ decimo,duodevicesimo,undevicesimo,bissexto
}
```

(*End definition for* `\c_kal_day_abl_clist`.)

`\c_kal_weekday_gen_clist` Latin names of weekdays in the genitive singular case, intended to follow the word *dies*. Saturday is placed first due to the weekday indexing used in Zeller's congruence.

```
\clist_new:N  \c_kal_weekday_gen_clist
\clist_set:Nn \c_kal_weekday_gen_clist {
  Saturni,Solis,Lunae,Martis,Mercurii,Iovis,Veneris
}
```

(*End definition for* `\c_kal_weekday_gen_clist`.)

`\c_kal_weekday_abbr_clist` Abbreviated Latin names of weekdays; a simple truncation of values from `\c_kal_weekday_gen_clist` could not be performed due to the irregular lengths of the abbreviations for Tuesday and Wednesday.

```
\clist_new:N  \c_kal_weekday_abbr_clist
\clist_set:Nn \c_kal_weekday_abbr_clist {
  Sat,Sol,Lun,Mart,Merc,Iov,Ven
}
```

(*End definition for* `\c_kal_weekday_abbr_clist`.)

`\c_kal_month_acc_clist` `\c_kal_month_abl_clist` `\c_kal_month_abbr_clist` Full Latin names of months in the accusative and ablative plural feminine, along with abbreviations for the months. Note that *Quintilis* and *Sextilis* are considered the thirteenth and fourteenth months to facilitate list access; however, *Quintilis* is not considered the month after December when determining its Kalends reference. August is repeated after *Sextilis* to facilitate Kalends referencing. These issues could be solved in a much cleaner manner if LaTeX3 data structures were more than simple stacks.

```
\clist_new:N  \c_kal_month_acc_clist
\clist_set:Nn \c_kal_month_acc_clist {
  Ianuarias,Februarias,Martias,Apriles,Maias,Iunias,Iulias,Augustas,
  Septembres,Octobres,Novembres,Decembres,Quintiles,Sextiles,Augustas
}
\clist_new:N  \c_kal_month_abl_clist
\clist_set:Nn \c_kal_month_abl_clist {
  Ianuariis,Februariis,Martiis,Aprilibus,Maiis,Iuniis,Iuliis,
  Augustis,Septembribus,Octobribus,Novembribus,Decembribus,
  Quintilibus,Sextilibus,Augustis
}
\clist_new:N  \c_kal_month_abbr_clist
\clist_set:Nn \c_kal_month_abbr_clist {
  Ian,Feb,Mar,Apr,Mai,Iun,Iul,Aug,Sept,Oct,Nov,Dec,Quin,Sex,Aug
}
```

(*End definition for* `\c_kal_month_acc_clist`, `\c_kal_month_abl_clist`, *and* `\c_kal_month_abbr_clist`.)

`\c_kal_month_lengths_clist` Number of days in all months, with February to be handled later.

```
\clist_new:N  \c_kal_month_lengths_clist
\clist_set:Nn \c_kal_month_lengths_clist {
  31,28,31,30,31,30,31,31,30,31,30,31,31,31
}
```

(*End definition for* `\c_kal_month_lengths_clist`.)

`\c_kal_month_ides_clist` Day on which the Ides of each month fall; determined by whether months are "full" or "hollow" on the Romulan calendar.

```
\clist_new:N  \c_kal_month_ides_clist
\clist_set:Nn \c_kal_month_ides_clist {
  13,13,15,13,15,13,15,13,13,15,13,13,15,13
}
```

(*End definition for* `\c_kal_month_ides_clist`.)

6.4 Internal utility functions

\kal_period:n Conditionally renders a period in abbreviations depending on the value of the `periods` option; saves a moderate amount of typing.

```
\cs_new:Nn \kal_period:n {
  \bool_if:NT \l_kal_periods_bool {.}
}
```

(*End definition for* `\kal_period:n`.)

\kal_abbr:nn Conditionally renders full text (`#1`) or an abbreviation (`#2`) followed by a period depending on the value of the `abbreviate` option.

```
\cs_new:Npn \kal_abbr:nn #1#2 {
  \bool_if:NTF \l_kal_abbreviate_bool {#2 \kal_period:n{}} {#1}
}
```

(*End definition for* `\kal_abbr:nn`.)

\kal_month:nn Conditionally renders the full name of the month at index `#2` in list `#1` or an abbreviation depending on the value of the `abbreviate` option.

```
\cs_new:Npn \kal_month:nn #1#2 {
  \kal_abbr:nn
    { \clist_item:Nn {#1} {#2} }
    { \clist_item:Nn \c_kal_month_abbr_clist {#2} }
}
```

(*End definition for* `\kal_month:nn`.)

\kal_ante_diem:nn Renders the day portion (`#1`) of dates that would be preceded by *ante diem* in the Classical period. If `#2` is equal to `1`, the full Latin ordinal number for the day is rendered in the accusative case; otherwise, the number is rendered as-is.

```
\cs_new:Npn \kal_ante_diem:nn #1#2 {
  \bool_if:nTF {\l_kal_antediem_bool}
    {
      \kal_abbr:nn {ante\ diem} {\KalAbbrFormat{a \kal_period:n{} d}}
      \ \int_compare:nNnTF {#2} = {1}
          { \clist_item:Nn \c_kal_day_acc_clist {#1} }
          { #1 }
    }
```

If the user has instead chosen to use the Medieval day syntax (no *ante diem*), render the number in the ablative case.

```
    {
      \int_compare:nNnTF {#2} = {1}
        { \clist_item:Nn \c_kal_day_abl_clist {#1} }
        { #1 }
    }
}
```

(*End definition for* `\kal_ante_diem:nn`.)

\kal_day:n Renders the full name of the day `#1` days before the next named day, inclusive. If it is the day directly before (2 days inclusive), render *pridie*.

```
\cs_new:Npn \kal_day:n #1 {
  \int_compare:nNnTF {#1} = {2}
    { \kal_abbr:nn {pridie} {prid} }
```

Otherwise, render the appropriate *ante diem* number in the format set by the user in the option `dayfmt`.

```
    {
      \str_case_x:nnF { \l_kal_dayfmt_tl }
        {
          { latin }  { \kal_ante_diem:nn {#1} {1} }
          { roman }  { \kal_ante_diem:nn {\KalDayFormat{
                                            \int_to_roman:n {#1}}} {0} }
          { arabic } { \kal_ante_diem:nn {\int_eval:n {#1}} {0} }
        }
```

If an invalid day format was set, use Roman numerals.

```
        { \kal_ante_diem:nn {\KalDayFormat{\int_to_roman:n {#1}}} {0} }
    }
}
```

(*End definition for* `\kal_day:n`.)

`\kal_bissextum_day:n` Renders the names of days (`#1`) in February in years for which *bissexta* (leap days) are inserted. If the specified day is before 24 February, it can be rendered normally.

```
\cs_new:Npn \kal_bissextum_day:n #1 {
  \int_compare:nNnTF {#1} < {24}
    { \kal_day:n { \l_kal_month_length_int - #1 + 1 } }
```

If the day is after 24 February, we must add another day to the distance from the Kalends to compensate for the *bissextum*.

```
    {
      \int_compare:nNnTF {#1} > {24}
        { \kal_day:n { \l_kal_month_length_int - #1 + 2 } }
```

If the specified day is the *bissextum* itself, render its unique string in the format requested by the user, or in Roman numerals if the format is invalid.

```
        {
          \str_case_x:nnF { \l_kal_dayfmt_tl }
            {
              { latin }  { \kal_ante_diem:nn {20} {1} }
              { roman }  { \kal_ante_diem:nn {bis\ \KalDayFormat{vi} } {0} }
              { arabic } { \kal_ante_diem:nn {bis\ \KalDayFormat{6} } {0} }
            }
            { \kal_ante_diem:nn {bis\ \KalDayFormat{vi} } {0} }
        }
    }
}
```

(*End definition for* `\kal_bissextum_day:n`.)

`\kal_year:n` Renders the year `#1` in the format set in the option *yearfmt*, defaulting to Roman numerals if an invalid value was set.

```
\cs_new:Npn \kal_year:n #1 {
  \KalYearFormat{
    \str_case_x:nnF { \l_kal_yearfmt_tl }
      {
        { roman }  { \int_to_roman:n {#1} }
        { arabic } { \int_eval:n {#1} }
      }
```

```
      { \int_to_roman:n {#1} }
  }
}
```

(*End definition for* `\kal_year:n`.)

6.5 Date rendering

`\l_kal_month_idx_int`
`\l_kal_month_length_int`
`\l_kal_month_ides_int`

Begin by defining some scratch variables for the index of the month to pull from the constant lists, the length of the specified month, and the day on which the Ides of the month fall.

```
\int_new:N \l_kal_month_idx_int
\int_new:N \l_kal_month_length_int
\int_new:N \l_kal_month_ides_int
```

(*End definition for* `\l_kal_month_idx_int`, `\l_kal_month_length_int`, *and* `\l_kal_month_ides_int`.)

`\kal_date:nnn`

We can now start the process of rendering a date string.

```
\cs_new:Npn \kal_date:nnn #1#2#3 {
```

Archaic months. The first quantity we must establish is the index of the month `#2` to use from the constant lists. If the user has requested the archaic months *Quintilis* and *Sextilis* via the `oldmonths` option, target those values at indices 13 and 14.

```
  \bool_if:nTF {\l_kal_oldmonths_bool}
    {
      \int_case:nnF {#2}
        {
          {7} { \int_set:Nn \l_kal_month_idx_int {13} }
          {8} { \int_set:Nn \l_kal_month_idx_int {14} }
        }
```

Otherwise, retrieve values from the index specified.

```
        { \int_set:Nn \l_kal_month_idx_int {#2} }
    }
    { \int_set:Nn \l_kal_month_idx_int {#2} }
```

Leap years. Now we must deal with leap years. Since this package targets both the Julian and Gregorian calendars, we shall need to handle both at once. Begin by checking the criteria for a leap month common to both calendars: the month must be February, and the year must be evenly divisible by 4.

```
  \int_compare:nNnTF {\l_kal_month_idx_int} = {2}
    {
      \int_compare:nNnTF { \int_mod:nn {#1} {4} } = {0}
        {
```

If the user has chosen to use the Julian calendar or the specified year satisfies the Gregorian criterion of not being evenly divisible by 100, the month will be 29 days long.

```
          \bool_if:nTF
            {
              \l_kal_julian_bool ||
              \int_compare_p:n { \int_mod:nn {#1} {100} != 0 }
            }
            { \int_set:Nn \l_kal_month_length_int {29} }
```

If the Gregorian calendar is in use and the year is evenly divisible by 100, the month will have 28 days, except in the case that the year is also evenly divisible by 400, when it will have 29 days.

```
            {
              \int_compare:nNnTF { \int_mod:nn {#1} {400} } = {0}
                { \int_set:Nn \l_kal_month_length_int {29} }
                { \int_set:Nn \l_kal_month_length_int {28} }
            }
        }
```

If none of the leap year criteria are satisfied, February will have 28 days.

```
        { \int_set:Nn \l_kal_month_length_int {28} }
    }
```

Finally, if the specified month is not February, look up its length in the list of month durations.

```
    {
      \int_set:Nn \l_kal_month_length_int
        { \clist_item:Nn \c_kal_month_lengths_clist {\l_kal_month_idx_int} }
    }
```

Naming the day. Now that we have figured out how long our month is, we can render the name of the day `#3`. We shall begin by determining on which day of the month the Ides will fall, as many of the other days pivot thereupon.

```
  \int_set:Nn \l_kal_month_ides_int
    { \clist_item:Nn \c_kal_month_ides_clist {\l_kal_month_idx_int} }
```

We shall handle the named days themselves first: the Kalends on the first day of the month, the Nones eight (nine inclusively) days before the Ides, and the Ides.

```
  \int_case:nnTF {#3}
    {
      { 1 }                          { \kal_abbr:nn {Kalendis} {Kal} }
      { \l_kal_month_ides_int - 8 } { \kal_abbr:nn {Nonis}    {Non} }
      { \l_kal_month_ides_int }      { \kal_abbr:nn {Idibus}   {Id}  }
    }
```

If the specified day is one of these named days, render the month in the ablative plural.

```
    { \ \kal_month:nn {\c_kal_month_abl_clist} {\l_kal_month_idx_int} }
```

If the specified day does not have a name ascribed to it, we must determine the next named day thereafter. We shall walk forward through the month to find this day, beginning with the days before the Nones, which take the Latin form *ante diem… Nonas*. Note that 7 days are subtracted from the Ides in this calculation as a further consequence of inclusive counting.

```
    {
      \int_compare:nNnTF {#3} < { \l_kal_month_ides_int - 8 }
        {
          \kal_day:n { \l_kal_month_ides_int - #3 - 7 }
          \ \kal_abbr:nn {Nonas} {Non}
          \ \kal_month:nn {\c_kal_month_acc_clist} {\l_kal_month_idx_int}
        }
```

If the day is not before the Nones, it may be before the Ides and would take the Latin form *ante diem… Idus*.

```
        {
          \int_compare:nNnTF {#3} < {\l_kal_month_ides_int}
            {
              \kal_day:n { \l_kal_month_ides_int - #3 + 1 }
              \ \kal_abbr:nn {Idus} {Id}
              \ \kal_month:nn {\c_kal_month_acc_clist} {\l_kal_month_idx_int}
            }
```

If the day is before neither the Nones nor the Ides, it must reference the Kalends of the following month, taking the Latin form *ante diem… Kalendas*. If it has been requested by the user, the *bissextum* falls in this date span, so the month of February must be handled separately from all other dates.

```
            {
              \bool_if:nTF
                {
                  \int_compare_p:nNn {\l_kal_month_idx_int} = {2} &&
                  \l_kal_bissextum_bool
                }
                { \kal_bissextum_day:n {#3} }
```

Outside of special Februaries, these dates may be handled normally. Note that two days are added to the specified day in calculation as a result of inclusive counting.

```
                {
                  \kal_day:n { \l_kal_month_length_int - #3 + 2 }
                }
                \ \kal_abbr:nn {Kalendas} {Kal}
```

We must also handle the fact that the month following December is January, so the Kalends references of December dates will be in the first month. For all other months, reference the next month.

```
                \ \int_compare:nNnTF {\l_kal_month_idx_int} = {12}
                  { \kal_month:nn {\c_kal_month_acc_clist} {1} }
                  {
                    \kal_month:nn {\c_kal_month_acc_clist}
                                  {\l_kal_month_idx_int + 1}
                  }
            }
        }
    }
```

Rendering the year. The only portion of the date that remains to be rendered is the year, in the format specified in the `era` option. If the era is not one of the ones listed below, the year is not displayed; the value `none` is suggested to the user to avoid unintended side-effects of utilising other values.

```
  \str_case_x:nn { \l_kal_era_tl }
    {
```

The first date format we shall handle is the *ab urbe condita* (AUC) era, which measures years relative to the legendary founding of Rome in 753 BCE. As opposed to the other supported eras, *ab urbe condita* has one peculiar quirk: while the *anno Domini* and Common Era formats omit the year zero, *ab urbe condita* continued through this year. Thus, while the numeric difference between 1 BCE and 1 CE is two years, the difference

between these years *ab urbe condita* is only one year. To solve this problem, we can use different addends for dates before and after "0 CE".

```
      { auc } {
        \ \KalYearFormat{
          \int_compare:nNnTF {#1} < {0}
            { \kal_year:n { #1 + 754 } }
            { \kal_year:n { #1 + 753 } }
        }
        \ \kal_abbr:nn {ab\ urbe\ condita}
            {\KalAbbrFormat{a \kal_period:n{} u \kal_period:n{} c}}
        }
```

The *anno Domini* era is significantly simpler, as its years are congruent with those used as input for this function. If a date falls after "0 CE", it is classified in the *anno Domini* (AD) era; if it falls before, it is *ante Christum* (AC).

```
      { ad } {
        \int_compare:nNnTF {#1} < {0}
          {
            \ \KalYearFormat{\kal_year:n {\int_abs:n {#1}}}
            \ \kal_abbr:nn {ante\ Christum}
                           {\KalAbbrFormat{a \kal_period:n{} c}}
          }
          {
            \ \kal_abbr:nn {anno\ Domini}
                           {\KalAbbrFormat{a \kal_period:n{} d}}
            \ \KalYearFormat{\kal_year:n {#1}}
          }
        }
```

The *anno*, or "short AD," pseudo-era is nearly identical, only replacing the phrase *anno Domini* with the shorter *anno* (A).

```
      { adshort } {
        \int_compare:nNnTF {#1} < {0}
          {
            \ \KalYearFormat{\kal_year:n {\int_abs:n {#1}}}
            \ \kal_abbr:nn {ante\ Christum}
                           {\KalAbbrFormat{a \kal_period:n{} c}}
          }
          {
            \ \kal_abbr:nn {anno} {\KalAbbrFormat{a}}
            \ \KalYearFormat{\kal_year:n {#1}}
          }
        }
```

Finally, the Common Era (*saeculum commune*) shares the same year numbers as the *anno Domini* era, but uses different verbiage.

```
      { secular } {
        \int_compare:nNnTF {#1} < {0}
          {
            \ \KalYearFormat{\kal_year:n {\int_abs:n {#1}}}
            \ \kal_abbr:nn {ante\ saeculum\ commune}
                {\KalAbbrFormat{a \kal_period:n{} s \kal_period:n{} c}}
          }
          {
            \ \kal_abbr:nn {saeculo\ communi}
```

```
                              {\KalAbbrFormat{s \kal_period:n{} c}}
            \ \KalYearFormat{\kal_year:n {#1}}
          }
        }
    }
}
```

(*End definition for* `\kal_date:nnn`.)

6.6 Rendering dates from strings

`\l_kal_date_year_int` `\l_kal_date_month_int` `\l_kal_date_day_int` `\l_kal_date_split_seq` `\l_kal_date_idx_int`

Begin by defining some scratch variables for the year, month, and day of the specified date; the date sequence after being split at the specified delimiters; and a loop index.

```
\int_new:N \l_kal_date_year_int
\int_new:N \l_kal_date_month_int
\int_new:N \l_kal_date_day_int
\seq_new:N \l_kal_date_split_seq
\int_new:N \l_kal_date_idx_int
```

(*End definition for* `\l_kal_date_year_int` *and others.*)

`\kal_date_string:nnn`

To parse a date string (`#3`), we must first determine the order in which the year, month, and day are arranged in the format string (`#1`). We begin by splitting the date string at the specified delimiter (`#2`).

```
\cs_new:Npn \kal_date_string:nnn #1#2#3 {
  \seq_set_split:Nnn \l_kal_date_split_seq {#2} {#3}
```

We now loop through the tokens in the format string, resolving each to one component of a fully-formed date. For each successful match, we set the corresponding variable to the element of the split sequence at the same index.

```
  \int_set:Nn \l_kal_date_idx_int {1}
  \int_do_until:nNnn {\l_kal_date_idx_int} = {4}
    {
      \str_case_x:nn {\tl_item:nn {#1} {\l_kal_date_idx_int}}
        {
          { Y } { \int_gset:Nn \l_kal_date_year_int
                  { \seq_item:Nn \l_kal_date_split_seq {\l_kal_date_idx_int} } }
          { M } { \int_gset:Nn \l_kal_date_month_int
                  { \seq_item:Nn \l_kal_date_split_seq {\l_kal_date_idx_int} } }
          { D } { \int_gset:Nn \l_kal_date_day_int
                  { \seq_item:Nn \l_kal_date_split_seq {\l_kal_date_idx_int} } }
        }
      \int_incr:N \l_kal_date_idx_int
    }
```

Finally, we render the date normally from these resolved components.

```
  \kal_date:nnn {\l_kal_date_year_int}
                {\l_kal_date_month_int}
                {\l_kal_date_day_int}
}
```

To allow this command to be used with macro arguments, we shall create a variant that expands the date string once.

```
\cs_generate_variant:Nn \kal_date_string:nnn { nno }
```

(*End definition for* `\kal_date_string:nnn`.)

6.7 Determining weekdays of dates

\l_kal_weekday_idx_int
\l_kal_weekday_month_int
\l_kal_weekday_year_int
\l_kal_weekday_century_int

Begin by defining some scratch variables for the month, year, and zero-based century of the specified date, and a variable in which to store the weekday.

```
\int_new:N \l_kal_weekday_idx_int
\int_new:N \l_kal_weekday_month_int
\int_new:N \l_kal_weekday_year_int
\int_new:N \l_kal_weekday_century_int
```

(*End definition for* `\l_kal_weekday_idx_int` *and others.*)

\kal_weekday:nnn

To determine the day of the week of an arbitrary date on either the Julian or Gregorian calendars, we shall use an implementation of Zeller's congruence. For the Gregorian calendar, this arithmetic algorithm is as follows:

$$w = \left(d + \left\lfloor \frac{13\,(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor + 5J \right) \mod 7$$

- w is the index of the day of the week, with Saturday at 0, Sunday at 1, Monday at 2, and so forth
- d is the day of the month
- m is the month, where all months follow their normal enumerations (March at 3, April at 4, etc.) except January and February, which are assigned indices 13 and 14, respectively
- K is the year of the date within its century $(y \mod 100)$
- J is the zero-based century of the year, $\left\lfloor \frac{y}{100} \right\rfloor$.

A similar formula exists for Julian dates:

$$w = \left(d + \left\lfloor \frac{13\,(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + 5 + 6J \right) \mod 7$$

We begin our implementation by determining the index of the specified month. Since we already need to add 12 to the indices of January and February, and the term m only appears in the quantity $(m+1)$, we shall combine these additions from the start.

```
\cs_new:Npn \kal_weekday:nnn #1#2#3 {
  \int_compare:nNnTF {#2} < {3}
    { \int_set:Nn \l_kal_weekday_month_int { #2 + 13 } }
    { \int_set:Nn \l_kal_weekday_month_int { #2 + 1 } }
```

We now determine the year of the century of the specified date, along with its zero-based century, according to the equations above.

```
  \int_set:Nn \l_kal_weekday_year_int { \int_mod:nn {#1} {100} }
  \int_set:Nn \l_kal_weekday_century_int { \fp_eval:n { floor(#1 / 100) } }
```

Next, we shall implement the portions of the algorithm that are shared between the Julian and Gregorian calendars, calculating all like terms between the two.

```
  \int_set:Nn \l_kal_weekday_idx_int {#3}
  \int_add:Nn \l_kal_weekday_idx_int
                { \fp_eval:n { floor(13 * \l_kal_weekday_month_int / 5) } }
  \int_add:Nn \l_kal_weekday_idx_int { \l_kal_weekday_year_int }
  \int_add:Nn \l_kal_weekday_idx_int
                { \fp_eval:n { floor(\l_kal_weekday_year_int / 4) } }
```

At this point, the algorithms diverge. If the date has been specified as Julian, we shall use the $\cdots + 5 + 6J$ termination.

```
  \bool_if:nTF {\l_kal_julian_bool}
    { \int_add:Nn \l_kal_weekday_idx_int { 5 + 6 * \l_kal_weekday_century_int } }
```

Otherwise, we shall use the more complex Gregorian termination.

```
    {
      \int_add:Nn \l_kal_weekday_idx_int { \fp_eval:n
          { 5 * \l_kal_weekday_century_int + floor(\l_kal_weekday_century_int / 4) }
        }
    }
```

Finally, we perform the modulus operation to reach an index value for the day of the week. As Zeller's algorithm was designed for 0-indexed programming languages, we must add 1 to the index to compensate for LaTeX3's 1-based indexing.

```
  \int_set:Nn \l_kal_weekday_idx_int { \int_mod:nn {\l_kal_weekday_idx_int} {7} }
  \int_incr:N \l_kal_weekday_idx_int
```

We can now fetch the appropriate form of the weekday from the lookup lists.

```
  \kal_abbr:nn
    { \clist_item:Nn \c_kal_weekday_gen_clist { \l_kal_weekday_idx_int } }
    { \clist_item:Nn \c_kal_weekday_abbr_clist { \l_kal_weekday_idx_int } }
}
```

(*End definition for* `\kal_weekday:nnn`.)

6.8 User-facing macros

`\KalDate` Render a Roman calendar date for the specified year (`#2`), month (`#3`), and day (`#4`) with the specified options (`#1`).

```
\DeclareDocumentCommand{\KalDate}{o m m m}
  {
    \IfValueTF{#1}
      {
        \group_begin:
          \keys_set:nn {kalendarium} {#1}
          \kal_date:nnn {#2} {#3} {#4}
        \group_end:
      }
      { \kal_date:nnn {#2} {#3} {#4} }
  }
```

(*End definition for* `\KalDate`. *This function is documented on page 7.*)

`\KalDateStr` Convert the specified date string (`#4`), with components in the order of the format string (`#2`) and delimited by a consistent value (`#3`), to a Roman calendar date with the specified options (`#1`). To avoid user confusion, the date string is always expanded.

```
\DeclareDocumentCommand{\KalDateStr}{o m m m}
  {
    \IfValueTF{#1}
      {
        \group_begin:
          \keys_set:nn {kalendarium} {#1}
          \kal_date_string:nno {#2} {#3} {#4}
```

```
      \group_end:
    }
    { \kal_date_string:nno {#2} {#3} {#4} }
}
```

(*End definition for* `\KalDateStr`*. This function is documented on page 7.*)

`\KalToday` Render the Roman calendar date for the current date with the specified options (`#1`).

```
\DeclareDocumentCommand{\KalToday}{o}
  {
    \IfValueTF{#1}
      {
        \group_begin:
          \keys_set:nn {kalendarium} {#1}
          \kal_date:nnn {\the\year} {\the\month} {\the\day}
        \group_end:
      }
      { \kal_date:nnn {\the\year} {\the\month} {\the\day} }
  }
```

(*End definition for* `\KalToday`*. This function is documented on page 8.*)

`\KalWeekday` Determine the Latin day of the week for the specified year (`#2`), month (`#3`), and day (`#4`) with the specified options (`#1`).

```
\DeclareDocumentCommand{\KalWeekday}{o m m m}
  {
    \IfValueTF{#1}
      {
        \group_begin:
          \keys_set:nn {kalendarium} {#1}
          \kal_weekday:nnn {#2} {#3} {#4}
        \group_end:
      }
      { \kal_weekday:nnn {#2} {#3} {#4} }
  }

\ExplSyntaxOff
```

(*End definition for* `\KalWeekday`*. This function is documented on page 8.*)

`\KalAbbrFormat` `\KalDayFormat` `\KalYearFormat` Commands to control the formatting of abbreviations, days, and years, respectively, in Roman calendar dates. The user is encouraged to override these commands to fit their particular æsthetic goals.

```
\newcommand{\KalAbbrFormat}[1]{\textsc{#1}}
\newcommand{\KalDayFormat}[1]{\MakeUppercase{#1}}
\newcommand{\KalYearFormat}[1]{\MakeUppercase{#1}}
```

(*End definition for* `\KalAbbrFormat`*,* `\KalDayFormat`*, and* `\KalYearFormat`*. These functions are documented on page 13.*)

⟨/package⟩