
icalendar

Release 3.9.1.dev0

June 21, 2015

1 Roadmap	3
2 Changes in version 3.0	5
2.1 API Change	5
2.2 Timezone support	5
3 About this fork which is not a fork anymore	7
4 Test Coverage Report	9
5 Contents	11
5.1 About	11
5.2 Installing iCalendar	11
5.3 iCalendar package	11
5.4 More documentation	16
5.5 API Reference	16
5.6 icalendar contributors	19
5.7 License	21
Python Module Index	23

The `icalendar` package is a parser/generator of iCalendar files for use with Python.

Homepage <http://icalendar.readthedocs.org>

Code <http://github.com/collective/icalendar>

Mailing list <http://github.com/collective/icalendar/issues>

Dependencies `setuptools` and since version 3.0 we depend on `pytz`.

Compatible with Python 2.6, 2.7 and 3.3+

License BSD

Roadmap

- 4.0: API refactoring

Changes in version 3.0

2.1 API Change

Since version 3.0 the icalendar de/serialization API is unified to use only `to_ical` (for writing an ical string from the internal representation) and `from_ical` (for parsing an ical string into the internal representation).

`to_ical` is now used instead of the methods `ical`, `string`, `as_string` and instead of string casting via `__str__` and `str`.

`from_ical` is now used instead of `from_string`.

This change is a requirement for future Python 3 compatibility. Please update your code to reflect to the new API.

2.2 Timezone support

Timezones are now fully supported in icalendar for serialization and deserialization. We use the `pytz` library for timezone components of datetime instances. The timezone identifiers must be valid `pytz` respectively Olson database timezone identifiers. This can be a problem for 'GMT' identifiers, which are not defined in the Olson database.

Instead of our own UTC tzinfo implementation we use `pytz` UTC tzinfo object now.

About this fork which is not a fork anymore

The aim of this fork (not fork anymore, read further) was to bring this package up to date with latest icalendar RFC specification as part of [plone.app.event](#) project which goal is to bring recurrent events to Plone.

After some thoughts we (Plone developers involved with [plone.app.event](#)) send a suggestion to icalendar-dev@codespeak.net to take over maintaining of icalendar. Nobody objected and since version 2.2 we are back to development.

Test Coverage Report

Output from coverage test:

Name	Stmts	Miss	Cover
src/icalendar/__init__	4	0	100%
src/icalendar/cal	312	7	98%
src/icalendar/caselessdict	65	7	89%
src/icalendar/compat	1	0	100%
src/icalendar/parser	193	5	97%
src/icalendar/parser_tools	20	0	100%
src/icalendar/prop	540	62	89%
src/icalendar/timezone_cache	1	0	100%
src/icalendar/tools	16	0	100%
TOTAL	1152	81	93%

5.1 About

Max M had often needed to parse and generate iCalendar files. Finally he got tired of writing ad-hoc tools. This package is his attempt at making an iCalendar package for Python. The inspiration has come from the email package in the standard lib, which he thinks is pretty simple, yet efficient and powerful.

At the time of writing this, last version was released more than 2 years ago. Since then many things have changed. For one, RFC 2445 was updated by RFC 5545 which makes this package. So in some sense this package became outdated.

5.2 Installing iCalendar

To install the icalendar package, use:

```
python setup.py install
```

If installation is successful, you will be able to import the iCalendar package, like this:

```
>>> import icalendar
```

5.2.1 Building the documentation locally

To build the documentation follow these steps:

```
$ git clone https://github.com/collective/icalendar.git
$ cd icalendar
$ virtualenv-2.7 .
$ source bin/activate
$ pip install -r requirements_docs.txt
$ cd docs
$ make html
```

You can now open the output from `_build/html/index.html`. To build the presentation-version use `make presentation` instead of `make html`. You can open the presentation at `presentation/index.html`.

5.3 iCalendar package

This package is used for parsing and generating iCalendar files following the standard in RFC 2445.

It should be fully compliant, but it is possible to generate and parse invalid files if you really want to.

5.3.1 File structure

An iCalendar file is a text file (utf-8) with a special format. Basically it consists of content lines.

Each content line defines a property that has 3 parts (name, parameters, values). Parameters are optional.

A simple content line with only name and value could look like this:

```
BEGIN:VCALENDAR
```

A content line with parameters can look like this:

```
ATTENDEE;CN=Max Rasmussen;ROLE=REQ-PARTICIPANT:MAILTO:example@example.com
```

And the parts are:

```
Name:   ATTENDEE
Params: CN=Max Rasmussen;ROLE=REQ-PARTICIPANT
Value:  MAILTO:example@example.com
```

Long content lines are usually “folded” to less than 75 character, but the package takes care of that.

5.3.2 Overview

On a higher level iCalendar files consists of components. Components can have sub components.

The root component is the VCALENDAR:

```
BEGIN:VCALENDAR
... vcalendar properties ...
END:VCALENDAR
```

The most frequent subcomponent to a VCALENDAR is a VEVENT. They are nested like this:

```
BEGIN:VCALENDAR
... vcalendar properties ...
BEGIN:VEVENT
... vevent properties ...
END:VEVENT
  END:VCALENDAR
```

Inside the components there are properties with values. The values have special types. like integer, text, datetime etc. These values are encoded in a special text format in an iCalendar file.

There are methods for converting to and from these encodings in the package.

These are the most important imports:

```
>>> from icalendar import Calendar, Event
```

5.3.3 Components

Components are like (Case Insensitive) dicts. So if you want to set a property you do it like this. The calendar is a component:


```
>>> cal = Calendar()
>>> cal['dtstart'] = '20050404T080000'
>>> cal['summary'] = 'Python meeting about calendaring'
>>> for k,v in cal.items():
...     k,v
(u'DTSTART', '20050404T080000')
(u'SUMMARY', 'Python meeting about calendaring')
```

NOTE: the recommended way to add components to the calendar is to use create the subcomponent and add it via `Calendar.add!` The example above adds a string, but not a `vText` component.

You can generate a string for a file with the `to_ical()` method:

```
>>> cal.to_ical()
'BEGIN:VCALENDAR\r\nDTSTART:20050404T080000\r\nSUMMARY:Python meeting about calendaring\r\nEND:VCALENDAR'
```

The rendered view is easier to read:

```
BEGIN:VCALENDAR
DTSTART:20050404T080000
SUMMARY:Python meeting about calendaring
END:VCALENDAR
```

So, let's define a function so we can easily display `to_ical()` output:

```
>>> def display(cal):
...     return cal.to_ical().replace('\r\n', '\n').strip()
```

You can set multiple properties like this:

```
>>> cal = Calendar()
>>> cal['attendee'] = ['MAILTO:maxm@mxm.dk', 'MAILTO:test@example.com']
>>> print display(cal)
BEGIN:VCALENDAR
ATTENDEE:MAILTO:maxm@mxm.dk
ATTENDEE:MAILTO:test@example.com
END:VCALENDAR
```

If you don't want to care about whether a property value is a list or a single value, just use the `add()` method. It will automatically convert the property to a list of values if more than one value is added. Here is an example:

```
>>> cal = Calendar()
>>> cal.add('attendee', 'MAILTO:maxm@mxm.dk')
>>> cal.add('attendee', 'MAILTO:test@example.com')
>>> print display(cal)
BEGIN:VCALENDAR
ATTENDEE:MAILTO:maxm@mxm.dk
ATTENDEE:MAILTO:test@example.com
END:VCALENDAR
```

Note: this version doesn't check for compliance, so you should look in the RFC 2445 spec for legal properties for each component, or look in the `icalendar/calendar.py` file, where it is at least defined for each component.

5.3.4 Subcomponents

Any component can have subcomponents. Eg. inside a calendar there can be events. They can be arbitrarily nested. First by making a new component:

```
>>> event = Event()
>>> event['uid'] = '42'
>>> event['dtstart'] = '20050404T080000'
```

And then appending it to a “parent”:

```
>>> cal.add_component(event)
>>> print display(cal)
BEGIN:VCALENDAR
ATTENDEE:MAILTO:maxm@mxm.dk
ATTENDEE:MAILTO:test@example.com
BEGIN:VEVENT
DTSTART:20050404T080000
UID:42
END:VEVENT
END:VCALENDAR
```

Subcomponents are appended to the subcomponents property on the component:

```
>>> cal.subcomponents
[VEVENT({'DTSTART': '20050404T080000', 'UID': '42'})]
```

5.3.5 Value types

Property values are utf-8 encoded strings.

This is impractical if you want to use the data for further computation. Eg. the datetime format looks like this: ‘20050404T080000’. But the package makes it simple to Parse and generate iCalendar formatted strings.

Basically you can make the add() method do the thinking, or you can do it yourself.

To add a datetime value, you can use Python’s built in datetime types, and then set the encode parameter to true, and it will convert to the type defined in the spec:

```
>>> from datetime import datetime
>>> cal.add('dtstart', datetime(2005,4,4,8,0,0))
>>> cal['dtstart'].to_ical()
'20050404T080000'
```

If that doesn’t work satisfactorily for some reason, you can also do it manually.

In ‘icalendar.prop’, all the iCalendar data types are defined. Each type has a class that can parse and encode the type.

So if you want to do it manually:

```
>>> from icalendar import vDatetime
>>> now = datetime(2005,4,4,8,0,0)
>>> vDatetime(now).to_ical()
'20050404T080000'
```

So the drill is to initialise the object with a python built in type, and then call the “to_ical()” method on the object. That will return an ical encoded string.

You can do it the other way around too. To parse an encoded string, just call the “from_ical()” method, and it will return an instance of the corresponding Python type:

```
>>> vDatetime.from_ical('20050404T080000')
datetime.datetime(2005, 4, 4, 8, 0)

>>> dt = vDatetime.from_ical('20050404T080000Z')
```

```
>>> repr(dt)[:62]
'datetime.datetime(2005, 4, 4, 8, 0, tzinfo=<UTC>)'
```

You can also choose to use the `decoded()` method, which will return a decoded value directly:

```
>>> cal = Calendar()
>>> cal.add('dtstart', datetime(2005,4,4,8,0,0))
>>> cal['dtstart'].to_ical()
'20050404T080000'
>>> cal.decoded('dtstart')
datetime.datetime(2005, 4, 4, 8, 0)
```

5.3.6 Property parameters

Property parameters are automatically added, depending on the input value. For example, for date/time related properties, the value type and timezone identifier (if applicable) are automatically added here:

```
>>> event = Event()
>>> event.add('dtstart', datetime(2010, 10, 10, 10, 0, 0,
...                               tzinfo=pytz.timezone("Europe/Vienna")))

>>> lines = event.to_ical().splitlines()
>>> self.assertTrue(
...     b"DTSTART;TZID=Europe/Vienna;VALUE=DATE-TIME:20101010T100000"
...     in lines)
```

You can also add arbitrary property parameters by passing a parameters dictionary to the add method like so:

```
>>> event = Event()
>>> event.add('X-TEST-PROP', 'tryout.',
...          parameters={'prop1': 'val1', 'prop2': 'val2'})
>>> lines = event.to_ical().splitlines()
>>> self.assertTrue(b"X-TEST-PROP;PROP1=val1;PROP2=val2:tryout." in lines)
```

5.3.7 Example

Here is an example generating a complete iCal calendar file with a single event that can be loaded into the Mozilla calendar

Init the calendar:

```
>>> cal = Calendar()
>>> from datetime import datetime
```

Some properties are required to be compliant:

```
>>> cal.add('prodid', '-//My calendar product//mxm.dk//')
>>> cal.add('version', '2.0')
```

We need at least one subcomponent for a calendar to be compliant:

```
>>> import pytz
>>> event = Event()
>>> event.add('summary', 'Python meeting about calendaring')
>>> event.add('dtstart', datetime(2005,4,4,8,0,0,tzinfo=pytz.utc))
>>> event.add('dtend', datetime(2005,4,4,10,0,0,tzinfo=pytz.utc))
>>> event.add('dtstamp', datetime(2005,4,4,0,10,0,tzinfo=pytz.utc))
```

A property with parameters. Notice that they are an attribute on the value:

```
>>> from icalendar import vCalAddress, vText
>>> organizer = vCalAddress('MAILTO:noone@example.com')
```

Automatic encoding is not yet implemented for parameter values, so you must use the ‘v*’ types you can import from the icalendar package (they’re defined in icalendar.prop):

```
>>> organizer.params['cn'] = vText('Max Rasmussen')
>>> organizer.params['role'] = vText('CHAIR')
>>> event['organizer'] = organizer
>>> event['location'] = vText('Odense, Denmark')

>>> event['uid'] = '20050115T101010/27346262376@mxm.dk'
>>> event.add('priority', 5)

>>> attendee = vCalAddress('MAILTO:maxm@example.com')
>>> attendee.params['cn'] = vText('Max Rasmussen')
>>> attendee.params['ROLE'] = vText('REQ-PARTICIPANT')
>>> event.add('attendee', attendee, encode=0)

>>> attendee = vCalAddress('MAILTO:the-dude@example.com')
>>> attendee.params['cn'] = vText('The Dude')
>>> attendee.params['ROLE'] = vText('REQ-PARTICIPANT')
>>> event.add('attendee', attendee, encode=0)
```

Add the event to the calendar:

```
>>> cal.add_component(event)
```

Write to disk:

```
>>> import tempfile, os
>>> directory = tempfile.mkdtemp()
>>> f = open(os.path.join(directory, 'example.ics'), 'wb')
>>> f.write(cal.to_ical())
>>> f.close()
```

5.4 More documentation

Have a look at the tests of this package to get more examples. All modules and classes docstrings, which document how they work.

5.5 API Reference

5.5.1 icalendar.cal

Calendar is a dictionary like Python object that can render itself as VCAL files according to rfc2445.

These are the defined components.

```
class icalendar.cal.Calendar (*args, **kwargs)
    This is the base object for an iCalendar file.
```

class `icalendar.cal.Component` (*args, **kwargs)

Component is the base object for calendar, Event and the other components defined in RFC 2445. normally you will not use this class directly, but rather one of the subclasses.

add (*name, value, parameters=None, encode=1*)

Add a property.

Parameters

- **name** (*string*) – Name of the property.
- **value** (*Python native type or icalendar property type.*) – Value of the property. Either of a basic Python type of any of the icalendar's own property types.
- **parameters** (*Dictionary*) – Property parameter dictionary for the value. Only available, if encode is set to True.
- **encode** (*Boolean*) – True, if the value should be encoded to one of icalendar's own property types (Fallback is "vText") or False, if not.

Returns None

add_component (*component*)

Add a subcomponent to this component.

content_line (*name, value, sorted=True*)

Returns property as content line.

content_lines (*sorted=True*)

Converts the Component and subcomponents into content lines.

decoded (*name, default=[]*)

Returns decoded value of property.

classmethod from_ical (*st, multiple=False*)

Populates the component recursively from a string.

get_inline (*name, decode=1*)

Returns a list of values (split on comma).

property_items (*recursive=True, sorted=True*)

Returns properties in this component and subcomponents as: [(name, value), ...]

set_inline (*name, values, encode=1*)

Converts a list of values into comma separated string and sets value to that.

to_ical (*sorted=True*)

Parameters sorted – Whether parameters and properties should be lexicographically sorted.

walk (*name=None*)

Recursively traverses component and subcomponents. Returns sequence of same. If name is passed, only components with name will be returned.

class `icalendar.cal.ComponentFactory` (*args, **kwargs)

All components defined in rfc 2445 are registered in this factory class. To get a component you can use it like this.

5.5.2 icalendar.prop

This module contains the parser/generators (or coders/encoders if you prefer) for the classes/datatypes that are used in iCalendar:

4.2 Defined property parameters are:

ALTREP, CN, CUTYPE, DELEGATED-FROM, DELEGATED-TO, DIR, ENCODING, FMTTYPE, FB-TYPE, LANGUAGE, MEMBER, PARTSTAT, RANGE, RELATED, RELTYPE, ROLE, RSVP, SENT-BY, TZID, VALUE

4.3 Defined value data types are:

BINARY, BOOLEAN, CAL-ADDRESS, DATE, DATE-TIME, DURATION, FLOAT, INTEGER, PERIOD, RECUR, TEXT, TIME, URI, UTC-OFFSET

iCalendar properties has values. The values are strongly typed. This module defines these types, calling `val.to_ical()` on them, Will render them as defined in rfc2445.

If you pass any of these classes a Python primitive, you will have an object that can render itself as iCalendar formatted date.

Property Value Data Types starts with a 'v'. they all have an `to_ical()` and `from_ical()` method. The `to_ical()` method generates a text string in the iCalendar format. The `from_ical()` method can parse this format and return a primitive Python datatype. So it should always be true that:

```
x == vDataType.from_ical(vDataType(x).to_ical())
```

These types are mainly used for parsing and file generation. But you can set them directly.

class `icalendar.prop.FixedOffset` (*offset, name*)
Fixed offset in minutes east from UTC.

class `icalendar.prop.LocalTimezone`
Timezone of the machine where the code is running.

class `icalendar.prop.TypesFactory` (**args, **kwargs*)
All Value types defined in rfc 2445 are registered in this factory class.

The value and parameter names don't overlap. So one factory is enough for both kinds.

for_property (*name*)
Returns a the default type for a property or parameter

from_ical (*name, value*)
Decodes a named property or parameter value from an icalendar encoded string to a primitive python type.

to_ical (*name, value*)
Encodes a named value from a primitive python type to an icalendar encoded string.

class `icalendar.prop.vBinary` (*obj*)
Binary property values are base 64 encoded.

class `icalendar.prop.vBoolean`
Returns specific string according to state.

class `icalendar.prop.vCalAddress`
This just returns an unquoted string.

class `icalendar.prop.vDDDLists` (*dt_list*)
A list of vDDDTypes values.

class `icalendar.prop.vDDDTypes` (*dt*)
A combined Datetime, Date or Duration parser/generator. Their format cannot be confused, and often values can be of either types. So this is practical.

class `icalendar.prop.vDate` (*dt*)
Render and generates iCalendar date format.

- class** `icalendar.prop.vDatetime` (*dt*)
Render and generates icalendar datetime format.
- `vDatetime` is timezone aware and uses the `pytz` library, an implementation of the Olson database in Python. When a `vDatetime` object is created from an ical string, you can pass a valid `pytz` timezone identifier. When a `vDatetime` object is created from a python datetime object, it uses the `tzinfo` component, if present. Otherwise an timezone-naive object is created. Be aware that there are certain limitations with timezone naive DATE-TIME components in the icalendar standard.
- class** `icalendar.prop.vDuration` (*td*)
Subclass of `timedelta` that renders itself in the iCalendar DURATION format.
- class** `icalendar.prop.vFloat`
Just a float.
- class** `icalendar.prop.vFrequency`
A simple class that catches illegal values.
- class** `icalendar.prop.vGeo` (*geo*)
A special type that is only indirectly defined in the rfc.
- class** `icalendar.prop.vInline`
This is an especially dumb class that just holds raw unparsed text and has parameters. Conversion of inline values are handled by the Component class, so no further processing is needed.
- class** `icalendar.prop.vInt`
Just an int.
- class** `icalendar.prop.vPeriod` (*per*)
A precise period of time.
- class** `icalendar.prop.vRecur` (**args, **kwargs*)
Recurrence definition.
- class** `icalendar.prop.vText`
Simple text.
- class** `icalendar.prop.vTime` (**args*)
Render and generates iCalendar time format.
- class** `icalendar.prop.vUTCOffset` (*td*)
Renders itself as a utc offset.
- class** `icalendar.prop.vUri`
Uniform resource identifier is basically just an unquoted string.
- class** `icalendar.prop.vWeekday`
This returns an unquoted weekday abbreviation.

5.6 icalendar contributors

- Johannes Raggam <johannes@raggam.co.at> (Maintainer)
- Max M <maxm@mxm.dk> (Original author)
- Andreas Zeidler <az@zitc.de>
- Andrey Nikolaev <nikolaeff@gmail.com>
- Barak Michener <me@barakmich.com>
- Christian Geier <contact@lostpackets.de>

- Christophe de Vienne <cdevienne@gmail.com>
- Dai MIKURUBE <dmikurube@acm.org>
- Dan Stovall <dbstovall@gmail.com>
- Eric Hanchrow <erich@cozi.com>
- Eric Wieser <wieser.eric@gmail.com>
- Erik Simmler <tgecho@gmail.com>
- George V. Reilly <george@reilly.org>
- Jannis Leidel <jannis@leidel.info>
- Jeroen van Meeuwen (Kolab Systems) <vanmeeuwen@kolabsys.com>
- Jordan Kiang <jordan@cozi.com>
- Klaus Klein <kleink+github@kleink.org>
- Laurent Lasudry <lasudry@gmail.com>
- Lennart Regebro <lregebro@nuxeo.com>
- Léo S <leo@naeka.fr>
- Marc Egli <frog32@me.com>
- Markus Unterwaditzer <markus@unterwaditzer.net>
- Martijn Faassen <faassen@infrae.com>
- Martin Melin <git@martinmelin.com>
- Michael Smith <msmith@fluendo.com>
- Mikael Frykholm <mikael@frykholm.com>
- Olivier Grisel <ogrisel@nuxeo.com>
- Pavel Repin <prepin@gmail.com>
- Pedro Ferreira <jose.pedro.ferreira@cern.ch>
- Rembane <andeke@gmail.com>
- Robert Niederreiter <rnix@squarewave.at>
- Rok Garbas <rok@garbas.si>
- Ronan Dunklau <ronan@dunklau.fr>
- Russ <russ@rw.id.au>
- Sidnei da Silva <sidnei@enfoldsystems.com>
- Stanislav Ochotnicky <sochotnicky@redhat.com>
- Stefan Schwarzer <sschwarzer@sschwarzer.net>
- Thomas Bruederli <thomas@roundcube.net>
- Thomas Weißschuh <thomas@t-8ch.de>
- Victor Varvayuk <victor.varvariuc@gmail.com>
- Wichert Akkerman <wichert@wiggy.net>
- cillianderoiste <cillian.deroiste@gmail.com>

- fitnr <fitnr@fakeisthenewreal>
- hajdbo <boris@hajduk.org>
- ilya <ilya@boltnev-pc.(none)>
- spanktar <spanky@kapanka.com>
- tgecho <tgecho@gmail.com>
- tisto <tisto@plone.org>

Find out who contributed:

```
$ git shortlog -s -e
```

5.7 License

Copyright (c) 2012-2013, Plone Foundation All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

i

`icalendar.cal`, 16
`icalendar.prop`, 17

A

add() (icalendar.cal.Component method), 17
add_component() (icalendar.cal.Component method), 17

C

Calendar (class in icalendar.cal), 16
Component (class in icalendar.cal), 16
ComponentFactory (class in icalendar.cal), 17
content_line() (icalendar.cal.Component method), 17
content_lines() (icalendar.cal.Component method), 17

D

decoded() (icalendar.cal.Component method), 17

F

FixedOffset (class in icalendar.prop), 18
for_property() (icalendar.prop.TypesFactory method), 18
from_ical() (icalendar.cal.Component class method), 17
from_ical() (icalendar.prop.TypesFactory method), 18

G

get_inline() (icalendar.cal.Component method), 17

I

icalendar.cal (module), 16
icalendar.prop (module), 17

L

LocalTimezone (class in icalendar.prop), 18

P

property_items() (icalendar.cal.Component method), 17

S

set_inline() (icalendar.cal.Component method), 17

T

to_ical() (icalendar.cal.Component method), 17
to_ical() (icalendar.prop.TypesFactory method), 18

TypesFactory (class in icalendar.prop), 18

V

vBinary (class in icalendar.prop), 18
vBoolean (class in icalendar.prop), 18
vCalAddress (class in icalendar.prop), 18
vDate (class in icalendar.prop), 18
vDatetime (class in icalendar.prop), 18
vDDDLists (class in icalendar.prop), 18
vDDDTypes (class in icalendar.prop), 18
vDuration (class in icalendar.prop), 19
vFloat (class in icalendar.prop), 19
vFrequency (class in icalendar.prop), 19
vGeo (class in icalendar.prop), 19
vInline (class in icalendar.prop), 19
vInt (class in icalendar.prop), 19
vPeriod (class in icalendar.prop), 19
vRecur (class in icalendar.prop), 19
vText (class in icalendar.prop), 19
vTime (class in icalendar.prop), 19
vUri (class in icalendar.prop), 19
vUTCOffset (class in icalendar.prop), 19
vWeekday (class in icalendar.prop), 19

W

walk() (icalendar.cal.Component method), 17