|  | Frigcal Developer Guide |
| --- | --- |

This is frigcal's Developer Guide. It provides implementation details and chronicles the changes made in frigcal releases. It was originally part of the main and sole document, but was split off from user-level documentation to form a separate document as of frigcal 2.0. The material here is primarily oriented towards programmers—it's developer notes for other developers, and serves as a sort of journal for the project. If you're looking for usage details, see instead the [User Guide](http://docs.google.com/UserGuide.html).

***Please note***: all of the older screenshots referenced in this document have been removed to minimize frigcal package size, and some of its former off-page links grew invalid or cumbersome after the docs reorganization. Links that could be repaired were spared, but many former links were culled here. See the newer [screenshots collection](http://docs.google.com/docimgs/index.html) for up-to-date GUI examples. This document still provides useful development context and will be maintained, but its presentation is less visual than it once was.

### Contents

* [**iCalendar Implementation**](#_gjdgxs)
* [**Release History**](#_30j0zll)
  + [Version 2.0.1: August 14, 2017](#_1fob9te)
  + [Version 2.0: June 15, 2017](#_3znysh7)
  + [Version 1.7: August 27, 2016](#_2et92p0)
  + [Version 1.6: October 6, 2015—January 27, 2016](#_44sinio)
  + [Version 1.5: March 18—April 29, 2015](#_z337ya)
  + [Version 1.4: February 10, 2015](#_3j2qqm3)
  + [Version 1.3: December 18, 2014](#_3whwml4)
  + [Version 1.2: October 14—November 10, 2014](#_qsh70q)
  + [Version 1.1: October 3, 2014](#_49x2ik5)
  + [Version 1.0: September 26, 2014](#_2p2csry)
* [**Project Goals and Review**](#_147n2zr)
  + [Why frigcal?](#_3o7alnk)
  + [Design and Results](#_23ckvvd)
  + [Future Directions?](#_ihv636)
  + [A Word on Extensions](#_41mghml)

# iCalendar Implementation

This section is probably of more interest to developers than users. The frigcal program records calendar events in portable, vendor-neutral iCalendar (".ics") text files. These files are loaded in full in startup, and regenerated in full on [quit requests](http://docs.google.com/UserGuide.html#quit) (and only after verification and a successful automatic backup).

The iCalendar files created by this program are standard-conforming, and should be useable in other calendar programs directly. Some other programs may require an initial import to use iCalendar files created here, and an export to create files usable here.

For more-technical readers already familiar with the iCalendar standard, here is a summary of its usage in frigcal:

1. This program can use iCalendar files created by other programs. This program also creates iCalendar files having one calendar object per file, and one or more nested event components per calendar.
2. This program recognizes and creates the following calendar component type to represent its calendar event entries:  
   * VEVENT

1. This program ignores and does not create these component types:  
   * VTODO
   * VJOURNAL
   * VALARM
   * VFREEBUSY
   * VTIMEZONE
   * and others (it's a comprehensive standard)

However, this program retains these component types in the iCalendar file if already present, for use by other programs.

1. For VEVENT components in files, this program uses and generates the following properties:

| Property | Role | Type |
| --- | --- | --- |
| UID | Unique id, for searches | Required |
| DTSTAMP | Last modified, type=date-time | Required |
| DTSTART | Event's date, type=date | Required, if no calendar 'method' |
| SUMMARY | Label text (one line) | Optional |
| DESCRIPTION | Extra text (multiline) | Optional |
| CATEGORIES | Color map text: uses first only | Optional |
| CREATED | To sort events, type=date-time | Optional |

1. Notably, this program does not use or generate DTEND, VALARM, or RRULE specifications for events, which precludes events spanning dates, reminders, and recurring events. You can simulate the spans and recurrence by simply copying and pasting an event on multiple consecutive or recurring days via right-clicks; for simplicity, alarms are outside this program's scope. See also the suggestions for recurrence implementation [ahead](#32hioqz). Journaling is also not supported, though event descriptions support arbitrarily long date-based notations.  
     
   Subtle point: strangely, the UID field's value is not always unique—some calendar programs may reuse the same UID for each instance of a recurring event. Although frigcal doesn't support recurring events directly, this won't generally cause problems: because frigcal indexes and locates events by UID within dates, events with duplicate UIDs display individually and work properly as long as events don't recur on the same day.
2. Though both rare and configurable, when some other programs use the calendar files this program creates, you may see quoted-printable escapes for literal backslashes embedded in an event's text fields. See [this note](#2xcytpi) for details.

For details on the iCalendar standard, search the web or see:

* <http://tools.ietf.org/html/rfc5545>
* [obsoletes <http://tools.ietf.org/html/rfc2445>]

Less formal sources:

* <http://en.wikipedia.org/wiki/ICalendar>
* <http://www.kanzaki.com/docs/ical/>

The following is an example of the iCalendar files this program creates, as of this copy (see also the screenshots of such a file with non-English Unicode content [here](http://docs.google.com/docimgs/macosx/main-window-may.png) and [here](http://docs.google.com/docimgs/macosx/event-view-edit-window.png)):

BEGIN:VCALENDAR  
VERSION:2.0  
PRODID:frigcal 1.5  
BEGIN:VEVENT  
SUMMARY:frigcal ics file created  
DTSTART;VALUE=DATE:20150406  
DTSTAMP:20150406T091225  
UID:frigcal-date150406-time091225-pid6604  
CATEGORIES:system internal  
CREATED:20150406T091225  
DESCRIPTION:Your default iCalendar file was generated.  
X-FRIGCAL-VERSION:1.5  
END:VEVENT  
BEGIN:VEVENT  
SUMMARY:fetch latest Windows 10 preview  
DTSTART;VALUE=DATE:20150408  
DTSTAMP;VALUE=DATE-TIME:20150406T091626Z  
UID:frigcal-date150406-time091626-pid6604  
CATEGORIES:gadgets  
CREATED;VALUE=DATE-TIME:20150406T091626Z  
DESCRIPTION:The current build makes a single-core netbook so slow that it'  
 s unusable.\nThis is probably due to an early tech preview build\, but nee  
 d to verify.\n  
END:VEVENT  
END:VCALENDAR

For more calendar file examples to study, see also the shipped calendar folder's [2.0-examples](http://docs.google.com/Calendars/2.0-examples) subfolder. You can view the content of ".ics" iCalendar files in that folder and its subfolders in any text-file editor or viewer.

# Release History

This section describes program changes and releases, most recent first. In this section:

|  | | Release | Date | Highlights | | --- | --- | --- | | [Version 2.0.1](#_1fob9te) | August 14, 2017 | Rerelease of Mac OS X app to fix broken-pipe error | | [Version 2.0](#_3znysh7) | June 15, 2017 | Mac OS X port, user configs file, search script, app/exes | | [Version 1.7](#_2et92p0) | August 27, 2016 | Event foreground colors, Unicode clarifications, utilities | | [Version 1.6](#_44sinio) | October 6, 2015 | PNGs without Pillow, Tk 8.6 colors, Linux upgrades | | [Version 1.5](#_z337ya) | March 18, 2015 | More error handling, image folder, new release structure | | [Version 1.4](#_3j2qqm3) | February 10, 2015 | Edit cancel verify, new launcher, file format, image tweaks | | [Version 1.3](#_3whwml4) | December 18, 2014 | Event selection lists, cut/copy context, tear-offs dropped | | [Version 1.2](#_qsh70q) | October 14, 2014 | Icon, help, missing configs, read-only text, day shade | | [Version 1.1](#_49x2ik5) | October 3, 2014 | Smarter saves: changed files only | | [Version 1.0](#_2p2csry) | September 26, 2014 | Initial release | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## Version 2.0.1: August 14, 2017 (Mac app)

This minor patch release applies to Mac app users only. Its sole change was a fix for a rare broken-pipe error that could occur only in the Mac app distribution of this program. For details on the issue and fix, please see [this webpage](http://learning-python.com/broken-pipe-workaround-aug17.html). Only the Mac app and source code packages were rereleased as 2.0.1, not the Windows or Linux executables. The source-code package was updated to include the app fix as an example for developers (see "[2.0.1]"), but the fix is irrelevant to both source code and other platforms. Users of the 2.0 Mac app are encouraged to [upgrade](http://docs.google.com/README.txt) to this new release.

***Update***: 2.0.1's source-code package (only) was republished in **May 2022**, with a very minor patch to fix an import statement in the embedded pytz library, which was broken by a change in Python 3.10. See the top of pytz's [lazy.py](http://docs.google.com/pytz/lazy.py) for the fix. No other changes were made, and app and executable packages are immune to this Python breakage because they embed an older Python. The source-code package, however, depends on a separately installed Python; is required for use on Android in the Pydroid 3 app, which will likely include Python 3.10 [soon](https://learning-python.com/using-tkinter-programs-on-android.html#Frigcal); and is recommended on Linux, to work around platform flux that botches [fonts](https://learning-python.com/post-release-updates.html#linux2017executables).

## Version 2.0: June 15, 2017

### ***Highlights***: Mac OS X port, user configs file, search script, app/exes, etc.

This version's enhancements were deemed significant enough to bump its release number up to 2.0. Search for "[2.0]" in the source code for a full accounting of this release's changes. Among the most notable:

1. frigcal was ported to work with full functionality on Mac OS X (in addition to Windows and Linux)
2. Now available as a "frozen" Mac app and Windows and Linux executable (in addition to source code)
3. A new GUI launcher displays an animation with help button until the main program loads calendar files
4. The configs file was split into a defaults base file plus a user-edits file, to simplify upgrades
5. There are new configurable items in [frigcal\_configs\_base.py](http://docs.google.com/frigcal_configs_base.py), including text-area size of event windows
6. Tk BMP fix: Unicode characters outside the BMP are replaced so they don't crash the GUI when displayed
7. The former Readme.html was split into a [UserGuide.html](http://docs.google.com/UserGuide.html) and [DeveloperGuide.html](http://docs.google.com/DeveloperGuide.html), and both were revised
8. A new script, [searchcals.py](http://docs.google.com/searchcals.py), provides calendar search; its output can be used to GoTo dates in the GUI
9. Right-click event cut/copy popups are now opened wider, to make them easier to locate
10. A new utility in docetc, [fixeoln-all.py](http://docs.google.com/fixeoln-all.py), can change end-lines in all of the package's text files

Of these, the app/executable distributions required the most work, and the Mac OS X port produced the most code changes. The latter category included control-click bindings to emulate right-clicks; Labels replacing colored Buttons; cosmetic changes to accommodate Mac Tk's different look-and-feel; top-of-screen menu protocol additions. slide-down dialogs, and ReopenDocument event bindings to deiconify.

## Version 1.7: August 27, 2016

### ***Highlights***: Event foreground colors, Unicode clarifications, utilities

This version adds foreground (text) color configuration for events, and verifies and extends the support of non-ASCII Unicode characters throughout the system. In short:

* *For colors*, two-item tuples may now be used to give both foreground and background color values for event summary [text](#tyjcwt), and a new utility [script](#3dy6vkm) simplifies custom color selections.
* *For Unicode*, this release adds a utility [script](#26in1rg) for file-encoding conversions (use this for existing calendars not in UTF-8 format); provides a precoded non-bold [fonts](#35nkun2) configuration for Asian characters on Linux (use this if your event text is unreadable); and documents source-based encoding [edits](#lnxbz9) (use this if file conversions are not an option). In addition, the GUI's Unicode display range is [described](#4d34og8), the configs file better supports both non-English [settings](#17dp8vu) and Linux end-line [format](#1ksv4uv), and trace messages no longer assume your console supports arbitrary Unicode [characters](#2s8eyo1).

A handful of minor items were also addressed. Most prominently, a typo was fixed in an error popup, and screenshot folders now have thumbnail index [pages](http://docs.google.com/docimgs/macosx/index.html) courtesy of the [*thumbspage*](http://learning-python.com/thumbspage.html) program.

### Details

This section provides more information about this version's changes. Search on "**[1.7]**" in the code for implementation details underlying the following:

1. **Event foreground colors configuration:**  Event summary entry widgets, displayed on days, can now have both background and foreground colors configured on a category-name or calendar-file basis. Prior versions supported only widget background color, and always used black for foreground text. The text itself can now be colored as well, by providing a ('*bgcolor*', '*fgcolor*') tuple for color settings, where both *bgcolor* and *fgcolor* may be a color name or #*RRGGBB* hex-value string. The former '*bgcolor*' string setting still works to set background color only, making this a fully backward-compatible extension (existing configs files work unchanged). For example, an event summary color value setting of:

* + **'wheat'** sets the event widget's background color only, leaving its foreground (the event text) the default black, as formerly
  + **('navy', 'orange')** sets the event widget's background color to 'navy' and its foreground (the event text) color to 'orange'

In all cases, the default event display background and foreground are white and black, respectively. Also note that this change applies to event summary display widgets only; other widgets' color settings differ. See the event color settings near the end of the [**configs file**](http://docs.google.com/UserGuide.html#configfile) for more documentation and examples.

1. **New color selector utility:**  Given the pervasiveness of color configurations in frigcal, this version also adds a simple GUI utility script for selecting custom colors. The new script, [**pickcolor.py**](http://docs.google.com/pickcolor.py), allows its user to choose a custom color, displays the color, and gives the color's #*RRGGBB* hex-value string in the GUI, ready for cut-and-paste into frigcal config-file settings (and GUIs, webpages, and any other system that uses these values).

1. **Unicode support validation:**  For this release, the system was verified to handle non-ASCII Unicode [characters](http://docs.google.com/docimgs/macosx/event-view-edit-window.png), with some minor limitations imposed by its GUI library. Specifically, both non-English language text and many Unicode symbols can be used in event summary and description test in the GUI, and may be saved in and loaded from calendar files. Basic Unicode support required no changes to code, as both the icalendar and Tk libraries used by frigcal support Unicode text:

* + The ***icalendar parser/generator library*** uses the general UTF-8 Unicode encoding everywhere by default. This allows frigcal to load and save calendar files containing arbitrary Unicode text. See [icalendar](https://pypi.python.org/pypi/icalendar) for more details.
  + The ***Tk GUI library*** supports input and display of a wide variety of Unicode characters. Its range allows for most characters used in modern languages and text, subject to the limitations described in the next item.

1. **Unicode limitations in the GUI:**  As mentioned, the Tk GUI library—underlying the Python tkinter module used by frigcal—supports input and display of Unicode characters, and handles almost all modern languages and a large number of symbols. Like other software, though, Tk does not support every defined Unicode character, and frigcal's Unicode display support is naturally limited to that of Tk. This support is partially font-dependent, though Tk automatically searches all its fonts for a character's glyph to display if absent from the font used. See [this](http://www.tcl.tk/doc/howto/i18n.html#tk) for Tk's glyph lookup mechanism, and [this](http://www.tkdocs.com/tutorial/fonts.html) for Tk fonts in general. Tk's Unicode support is also prone to evolve like Unicode itself, though Tk is currently years behind the Unicode curve.  
     
   More fundamentally, Tk currently seems to be limited to Unicode character code points in the range **U+0000...U+FFFF**—a.k.a. the Unicode Basic Multilingual Plane, or [BMP](https://en.wikipedia.org/wiki/Plane_%28Unicode%29#Basic_Multilingual_Plane). This effectively precludes GUI use of supplementary characters and Unicode planes outside the BMP's range. The details and evidence behind this are fairly technical (and can be safely ignored by most frigcal users), but include tkinter error messages, developer forums, and source code.  
     
   For example, *inserting* a Unicode checkmark character into either the tkinter *Text* or *Entry* widgets with the first line of the following code works as expected, displaying a checkmark in the GUI:  
   text.insert(END, 'xyz\U00002713') # works (checkmark symbol)   
   text.insert(END, 'xyz\U0001F383') # fails (jack-o-lantern emoji)  
   However, the second code line above fails to display a Unicode jack-o-lantern character, instead generating an exception and this tkinter error message on the console (tkinter uses Tk, which uses Tcl):  
   \_tkinter.TclError: character U+1f383 is above the range (U+0000-U+FFFF) allowed by Tcl  
   Moreover, *pasting* a checkmark character from another source into either of these two widgets works fine, but pasting a jack-o-lantern leaves a generic square character on Windows instead. This is a well-known issue, which also impacts Python's IDLE, and is discussed in multiple Tcl/Tk and Python developer threads; for a representative sample, see [this](http://wiki.tcl.tk/1364) and [this](http://bugs.python.org/issue13153). This is also called-out in Tcl/Tk's source code; per its *tcl.h* header file in version 8.6:  
   Tcl is currently UCS-2 and planning UTF-16 for the Unicode string rep that   
   Tcl\_UniChar represents. Changing the size of Tcl\_UniChar is /not/ supported.  
   Per all the evidence, Tk appears to currently support only the UCS-2 subset of Unicode, without the variable-length surrogate extensions of UTF-16. If you're too new to Unicode to know what that means, see [this](https://en.wikipedia.org/wiki/UTF-16) for UTF-16 and its more limited UCS-2 predecessor, and [this](http://www.unicode.org/faq/utf_bom.html#utf16-11) for more on UCS-2's limitations. In sum, Tk's UCS-2 range can represent only the code points in the Unicode BMP. The net effect is that jack-o-lanterns, emojis in general, and some other silly bits appear to be right out for the time being in Tk, tkinter, and frigcal.  
     
   The *good news* here is that Tk's Unicode support is very broad: modern text consists almost entirely of character code points in the UCS-2 range, and is fully supported by frigcal's GUI. For examples of the sorts of Unicode characters that Tk (and hence frigcal) *does* support, even for portable fonts such as Courier, Times, and Helvetica, open file [**unicode-cheat-sheet.txt**](http://docs.google.com/unicode-cheat-sheet.txt) in any UTF-8 Unicode-aware text editor or viewer. For more symbols to use in frigcal, try [here](https://en.wikipedia.org/wiki/Miscellaneous_Symbols) and [here](https://en.wikipedia.org/wiki/Unicode_symbols), and choose characters whose code points are in Tk's current U+0000...U+FFFF BMP range.  
     
   Also keep in mind that Tk's limits pertain to already-decoded Unicode characters used in the GUI, and are independent of Unicode file encodings. Python, icalendar, and frigcal can still happily load and save UTF-8 calendar-file text having characters outside the BMP, even if the Tk GUI library has issues inputting or displaying content outside this range.

1. **Unicode trace messages:**  As a minor accommodation to Unicode, frigcal's trace messages were specialized in this release to avoid exceptions on terminals that don't support ASCII text, by printing Unicode '\u....' escapes when required to produce ASCII. Trace messages are not significant to most users, and should not trigger exceptions in any event. PYTHONIOENCODING shell variable settings may address this is some cases, but seem less user-friendly. See [sharednames.py](http://docs.google.com/sharednames.py) for the new tracing code.

1. **Unicode strings in configs code:**  As another minor upgrade for Unicode, the user [**configs file**](http://docs.google.com/UserGuide.html#configfile) grew an encoding declaration line, "# -\*- coding: utf-8 -\*-", along with a note and example documenting the use of non-English characters in the file's string literals. Python defaults to UTF-8 for source-file encoding (which handles ASCII), but the declaration line was added to ease changes to other encodings. This scheme allows non-English text to be used in the file's strings, for settings such as file paths and event category names. Note that the configs file is now shipped in UTF-8 form to support and illustrate this change; see its documentation for usage details.

1. **Unicode conversion options:**  Although frigcal already supports Unicode characters in event text and files, its Unicode model always uses UTF-8 text encoding for both existing and newly generated calendar files. For existing files encoded per other schemes, this constraint may be inconvenient. Consequently, this version provides a converter script for existing files, and documents how to change the default encoding in source files. The new options fall into three use cases, depending on your existing files, and in decreasingly recommended order:

* 1. ***No changes or conversions***: when there are no existing calendar files, or all existing calendar files are encoded per either UTF-8 or ASCII, frigcal's default UTF-8 encoding suffices, and no file conversions or source-code changes are required. This case likely applies to the vast majority of users. Note that frigcal's UTF-8 default works for ASCII files and event text too, because ASCII is a subset of UTF-8.
  2. ***Converting existing files—new utility***: for users with existing calendar files encoded with a Unicode scheme other than UTF-8 or ASCII (e.g., Latin-1), a new file-conversion script can be used to perform a one-time conversion to UTF-8. See the new encoding converter script, [**unicodemod.py**](http://docs.google.com/unicodemod.py), for more details. In short, a simple command line:  
     C:\...\frigcal> **unicodemod.py icsfilename latin-1 utf-8**  
     suffices to convert a file from non-ASCII Latin-1 to the required UTF-8 format. Converting existing calendar files this way allows frigcal to use its default UTF-8 encodings unchanged for both input and output. Crucially, this also allows frigcal to generate newly saved calendar files with Unicode characters that may be outside the scope of original narrower encodings of existing files. Because of this generality, this approach is recommended for users with existing non-UTF-8 files.
  3. ***Changing encoding in source***: for users with existing non-UTF-8 calendar files, source-code changes offer another, though more complex, option. The underlying icalendar package's setting *DEFAULT\_ENCODING* in module *icalendar.parser\_tools* controls most or all of its Unicode behavior. Unfortunately, this setting can be changed *only* in its source code (file [*icalendar\parser\_tools.py*](http://docs.google.com/icalendar/parser_tools.py)), because it is imported throughout the icalendar package with "from" statements before any modules outside the package have a chance to change it—thus precluding a frigcal configs file solution. If are willing to change source code, though, this setting might be used to avoid file conversions by changing icalendar's (and hence frigcal's) Unicode encoding scheme globally. Its preset value is UTF-8, but any encoding name string supported by Python is allowed, and will be used by frigcal's file processing (as of this release).  
       
     Due to the icalendar library's implementation, however, this setting is used for *both* input and output file text—it will be used to both load existing calendar files and save generated calendar files. Hence, this setting cannot be used to load a non-UTF-8 existing file *and* save newly generated files in general UTF-8. If changed to a narrower encoding (e.g., Latin-1), this setting makes Unicode characters outside the encoding's scope unusable in event text, because they cannot be saved in files. Users are thus encouraged to convert non-UTF-8 files to UTF-8 per option B above, unless they are both open to changing source code, and sure that their event text will always remain in a different encoding's scope.

1. **Unicode fonts on Linux:**  Per testing thus far, *bold fonts* can sometimes make some Asian characters (and possibly others) render unreadably on Linux. On the other hand, bold seems to work well for all characters and fonts on Windows and most languages on Linux and Mac OS X, and non-bold seems too dull in all other contexts. It's unknown whether the lack of readability for Linux bold fonts reflects testing parameters or a more general issue, but in either case dropping bold font presets for a subset of users on a single platform seems too extreme a response.  
     
   To address this, bold font defaults were retained, but a new setting in the [**configs file**](http://docs.google.com/UserGuide.html#configfile), *LINUX\_ASIAN\_FONTS*, is precoded to use non-bold fonts for all event text displays. Set this to True if your non-English characters are difficult to read on Linux or elsewhere. You may also want to use a larger font *size* or a different font *family* in this same code on Linux, but this is subjective, and may also need to be specialized by platform (see *MONOSPACE\_FONTS* in the configs file for an example of platform-specific fonts).  
     
   It may be possible to automate some font selections (e.g., with locale queries), but any such "clever" automatic choices in this domain would almost certainly be inappropriate for some individuals and use cases. frigcal instead encourages its users to configure the program for their own context and tastes.

1. **File end-line conversion utility:**  For users on Linux and other Unix-like platforms, this version includes a simple script, [**fixeoln.py**](http://docs.google.com/fixeoln.py) which can be used to convert file end-lines from the shipped Windows format to the native Linux format, when no other tools are available. This script is essentially a Python-coded, Unicode-aware, and portable equivalent to the sometimes-absent Linux commands *dos2unix* and *unix2dos*, and works for files of any Unicode encoding type. See this script's docstring, [run log](http://docs.google.com/fixeoln_docetc/fixeoln-test.txt) for usage details. For additional Linux end-line options, see the documentation of the [configs file](http://docs.google.com/UserGuide.html#configfile)—this script's initial motivation. To convert endlines in *all* files in the package, see also the companion script [fixeoln-all.py](http://docs.google.com/fixeoln-all.py) new in version 2.0.

## Version 1.6: October 6, 2015—January 27, 2016

### ***Highlights***: PNGs without Pillow, Tk 8.6 colors, Linux upgrades

This version makes an install of the third-party Pillow (a.k.a. PIL) library optional for some combinations of image file type and Python version; addresses incompatible changes in color names in the Tk 8.6 library underlying tkinter in Python 3.4 and later on Windows; and on Linux improves the behavior of modal dialogs and window icons. As none of this release's changes impacted any existing GUI displays no prior screenshots were retaken, but new shots were taken for the new image load error handling, and usage examples on Windows 10 and Ubuntu Linux. This document was also updated and reformatted for usability.

Repackaged **Jan-27-16** with minor changes reflecting the fact that Python version does not necessarily imply Tk version—it does on a standard python.org Windows install, but may not on Mac OS X and elsewhere, as described in detail [here](http://learning-python.com/python-changes-2014-plus.html#s359). Changed the [configs file](http://docs.google.com/UserGuide.html#configfile) to pick colors based on Tk version (not Python version), and changed the documentation in this file as well as the text of [READMEs](http://docs.google.com/MonthImages/README.txt) and start-up and error messages to better clarify Tk version impact on month images. Also included the latest [Pillow](http://docs.google.com/dependencies/pillow-for-windows/README.txt) installers for Windows, for Pythons or image file types that don't satisfy the Tk 8.6 requirement described in #4 [below](#2jxsxqh).

Repackaged **Nov-10-15** with doc changes (new font, header, and toolbar styling; minor content tweaks; and updated URLs for book site relocation); plus one minor nonfunctional code change (remove a pointless staticmethod declaration for one method).

### Details

The following list provides more information on this release's changes. Search on "**[1.6]**" in the code for more implementation details underlying the following:

1. **Verified on Python 3.5, Windows 10, and Ubuntu Linux:**  frigcal was verified to run on Python 3.5, and on both Windows 10 and Ubuntu Linux. In the standard Windows installer, Python 3.5 (and 3.4) ships with and uses the *Tk 8.6 library* underlying Python's tkinter GUI toolkit, and changes in the Tk library were the motivation for many of this release's code changes. Note that all 1.6 changes are backward compatible with earlier Pythons, so frigcal still runs as before on all other Python 3.X releases. Also note that Python 3.5 appears to *speed up initial calendar file loads* substantially—reducing load time from 5 to 3 seconds in one use case, and from 13 to 8 seconds in another. Given this, and the PNG images support described ahead, Python 3.5 and later are suggested on Windows, and Tk 8.6 is suggested elsewhere.
2. **Color names and notes in configs file for Tk 8.6:**  Tk 8.6 changes the meaning of some color names to be compatible with a Web standard—instead of the X11/VGA standard used for the last 25 years. Namely, "green", "purple", "maroon", and "grey/gray" render more darkly than before, and are now too dark for use as label backgrounds in most contexts. To address this, the user [configs file](http://docs.google.com/UserGuide.html#configfile) was expanded with a note, and pre-coded assignments that map the four old names to their Tk 8.6 equivalents when using Tk 8.6+ only. This is now version-specific, as some of the new names aren't available in prior Tk versions. Tk 8.6 is standard for Python 3.4+ in python.org's Windows installers, though not necessarily so on other platforms (e.g., python.org's installer for Mac OS X currently uses Tk 8.5 with Python 3.5); Tk version can be queried with tkinter.TkVersion.
3. **Customizable current-day shading color in configs file:**  The shading color used for the currently focused day frame is now user-customizable in the [configs file](http://docs.google.com/UserGuide.html#configfile). It still defaults to the former gray, but uses the pre-Tk 8.6 rendition of gray, and allows for customizations both for user preference and future Tk change. This new color setting, "currentdaycolor", is used both when days are clicked, and when moving to the current day via the "Today" button or Escape key presses.

1. **Third-party Pillow library is no longer required for some month images:**  Because Tk 8.6 adds support for PNG image display, the rules regarding [Pillow](http://docs.google.com/UserGuide.html#dependencies) installs have been relaxed. Rather than making Pillow an absolute requirement for month images, frigcal now attempts to load image files with Pillow if it's present, and without Pillow if it's absent, and simply catches the exception that occurs if the image load fails. The net result is that, depending on your image file types and Python version, a Pillow install may no longer be required to use the optional month images display. Specifically, month images now display with just Python itself and do not require a Pillow install for both:  
   * Image type PNG, and a Python using Tk 8.6 and later—including a standard Windows install of Python 3.4 and later
   * Image type GIF or PPM/PPG, and any Python 3.X

For all other combinations of image type and Python release (e.g. JPEGs, and PNGs under Windows Python 3.3 and earlier), Pillow is still currently required for the image display feature, and a new error popup is still issued if images are enabled and a month image cannot be displayed. The new popup is issued on each image load failure, and replaces the former error popup that was issued once at Images toggle-on if Pillow was not installed—the new protocol doesn't require Pillow, and allows some images to succeed if others fail. On image load failures, a default image is also displayed as a placeholder for the month.  
  
Users are generally encouraged to use the popular PNG format and Python 3.4+ on Windows (or a Tk 8.6+ elsewhere), as this both allows for high-quality images and removes the requirement to install Pillow altogether. The latter is especially desirable, given that Pillow releases for new Pythons may be delayed substantially. As shipped, the top-level [MonthImages](http://docs.google.com/UserGuide.html#datafiles) folder was changed to include [PNG files](http://docs.google.com/MonthImages) instead of the former JPEGs, to both mirror suggested usage and support Windows Python 3.4 and later out of the box. Change and [configure](http://docs.google.com/UserGuide.html#configfile) as desired.

1. **On Linux, keep modal dialogs on top:**  Custom modal dialogs on Linux are now kept above their month window, by adding a *transient()* call for the new dialog window. Without this call, frigcal's custom modal dialogs—event selection lists, right-click actions, and event create and edit—pop up on top initially, but can be covered and hence hidden by their disabled month window parent before being processed by the user. This is especially bad for the right-click dialog, given its small size; the unresponsive month window can easily cover the dialog if clicked, leaving the GUI in an odd state. With the added call, modal dialogs can never be covered by their parent on either Fedora or Ubuntu Linux; Fedora also erases the pop up's "X" close button, but the dialogs' Cancel closes as usual. This change applies to Linux only: *transient()* is not required and not used on Windows, as that platform's custom modal dialogs do the right thing without it, and dialog behavior on Macs is to be tested. See the "Dialog" superclass in [frigcal.py](http://docs.google.com/frigcal.py) for this change's code.
2. **On Linux, set some window icons:**  tkinter's *iconphoto()*, available in Tk 8.5 and later, is now used to set window icons on Linux. On both Fedora and Ubuntu, this change simply provides an image for the program's icon in the activities/applications bar, not on the program window itself (Fedora also displays a small version of the icon at the top of screen). There are additional ways to set icons on Linux (e.g., XBM bitmap files, via *iconbitmap('@myicon.xbm')*; see the web for pointers. This change applies to Linux only: the existing *iconbitmap()* scheme sets icons on Windows correctly and is retained for that platform, and *iconphoto()* is documented as not relevant for Macs. Search for "try\_set\_window\_icon" in [frigcal.py](http://docs.google.com/frigcal.py) for this change's code.
3. **Also for Linux users in this release:**  There are new [usage notes](http://docs.google.com/UserGuide.html#programfiles), and the [configs file](http://docs.google.com/UserGuide.html#configfile) automatically translates any Windows "\" to "/" in calendar and image folder paths—a minor change but useful when selecting precoded paths in the file.
4. **[Caveat] On Linux, images are not hidden with their months:**  Month image windows are not hidden (minimized) and restored with their month window parents on Linux as they are on Windows. To address this, this version intended to experiment with using tkinter's *withdraw()* to hide image windows on Linux, instead of the *iconify()* used successfully on Windows. The *withdraw()* call is required on Linux (only) if you wish to restore later with *deiconify()*. See the new supplemental example script [\_\_withdraw\_\_.py](http://docs.google.com/__withdraw__.py) for a simple demo of this Linux constraint.  
     
   Unfortunately, the experiment never got that far—on Linux (again, only), the required "<Unmap>" and "<Map> events are never generated on window minimize and restore. Moreover, the "<Configure> event is also useless in this context: it doesn't fire on minimize or restore on Linux either (and only on restore on Windows), and in any event invokes a function on every size and location change. Hence, this issue has been abandoned. In the end, tkinter's portability may be less robust than it might be, and too incomplete to support this particular intended behavior. Run the [\_\_hide-unhide\_\_.py](http://docs.google.com/__hide-unhide__.py) supplemental example on Linux for a demo of the events issue.  
     
   Note that the *withdraw()* call also works on Windows, so it is potentially more portable than *iconify()*. However, *withdraw()* also prevents the image window from appearing in the taskbar along with the month window (per [\_\_withdraw\_\_.py](http://docs.google.com/__withdraw__.py)). It's not clear which flavor is preferable, but the prior *iconify()* behavior and code was retained for Windows on the grounds of frigcal compatibility. The event triggers issue rendered Linux portability a wholly moot point in this instance. Search for "<Unmap>" in [frigcal.py](http://docs.google.com/frigcal.py) for the related code.  
     
   The good news is that the vast majority of frigcal works and looks the same on Windows and Linux with no code changes; isolated window manager issues might be pardoned as unavoidably platform-specific.

## Version 1.5: March 18—April 29, 2015

### ***Highlights***: more error handling, image folder, new release structure

Version 1.5 adds more explicit error checks to make the system more robust and user-friendly, and avoid silent shutdowns when used in non-console mode on Windows. It also tightens-up image folder handling to allow non-image files, and has a new release structure on the web.

### Details

Search on "**[1.5]**" in the code for changes made in support of the following extensions introduced in this version.

#### March Release

1. **Catch calendar startup errors**: When an error occurs while loading the calendar files(s) at startup, catch the exception and display a new GUI popup, not only Python exception text in the console window. This is especially important on Windows, where the console window may be absent (causing the program to close without any message), or may disappear immediately before the exception's text can be read (the notorious disappearing window feature). Startup failures are most likely due to bad calendar path settings (hence the message's hint), though bad data and parser failures are not impossible. For symmetry, also extended the existing backup/generate error popups to include basic exception text, lest the console be absent on Windows.
2. **Catch config and library startup errors**: Also at start-up time, catch errors in the user [configurations file](http://docs.google.com/UserGuide.html#configfile) and failed imports of 3rd-party [library dependencies](http://docs.google.com/UserGuide.html#dependencies), and report these as GUI popups as well. Libraries are unlikely to fail (they are shipped with the system), but configs file errors may be common. Like calendar errors, these are both fatal, but the main GUI hasn't been built yet; show a special-case popup, instead of printing text to the console (which may not be present on Windows), and wait for an OK press, instead of keeping the console open by pausing for input (which may never come).
3. **Handle pruning delete failures**: File deletions can fail for many reasons, including permission changes, device failures, and in-use locks; this is especially true on Windows, where even the covert operation of the search indexer or an antivirus scanner might interfere with deletions. When deleting an old calendar backup file in the pruning process, don't let a delete failure cancel the entire backup/save process. Instead, issue a warning popup but ignore the delete and cancel the pruning step only, and proceed on to backup/save. It's likely that the next backup/save will be able to prune the file whose delete failed.
4. **Warn about missing image files, bad image path**: Issue a new warning popup in the GUI (once per window, at first image toggle-on) and a message in the console (on each display attempt) if the image folder does not contain exactly 12 image files. Otherwise, some months may appear to not load an image, because indexing a too-short sorted files list triggers an exception, shown only in the console. Having more than 12 images would work (ignoring items past the 12th by sort order), but most likely also reflects a user error. Also added a check and simple error popup for bad image file paths in the [configs](http://docs.google.com/UserGuide.html#configfile) file; this and an existing error popup for Pillow-not-installed are both issued at each image toggle-on and disable images altogether, but don't stop the program. ***Update***: see [version 1.6](#_44sinio)'s relaxation of Pillow install requirements.
5. **Skip non-image files in images folder**: When loading a month's image, skip and ignore any non-image files in the [images folder](http://docs.google.com/UserGuide.html#datafiles), by using Python's *mimetypes* module to determine type from filename extension. This includes any ".txt" text files present (including the [folder](http://docs.google.com/MonthImages)'s README.txt, which works here now); an *.htaccess* web admin file (if ever added to customize indexes for 1.5's web image: see note #6); and any other non-image items added by the user. This test is in addition to the subfolder skipping added in [version 1.4](#_3j2qqm3); with the new check, image file selection proceeds as follows—to display a month image, the system:  
   1. Lists all items in the images folder
   2. Removes non-file items (e.g., subfolders)
   3. Removes non-image files (e.g., text files, web admin files)
   4. Sorts by item filename
   5. Indexes to load the item for the month's relative number

The net effect is that users may place arbitrary subfolders and non-image files in their image folders; only 12 actual image files are expected and used. Unknown or corrupted image files in this folder may still fail to display, and some image displays may fail unless you have Pillow (PIL) installed (see [Dependencies](http://docs.google.com/UserGuide.html#dependencies)).

1. **Default calendar file on adds**: In the GUI's create event dialog, select the "frigcal-default" calendar file as the initial default if it exists. This is somewhat arbitrary, but the first calendar by name sort order seems always inappropriate in the system's author's usage.
2. **New release structure, docs**: Adjusted links in docs to use a new examples structure that's the same in the [zip file](http://learning-python.com/frigcal-products/unzipped) and the new [website](http://learning-python.com/frigcal.html) that hosts this system for online viewing: for the web, use links relative to a simple unpacked copy of the zip file. This required adding an *.htaccess* file in the top-level folder (only) for indexes, reinstating a stub top-level README.txt file for index display removed in [version 1.2](#_qsh70q), and renaming this doc file to "Readme-frigcal.html" (see [.htaccess](http://docs.google.com/.htaccess); this file was later renamed to "UserGuide.html" in version 2.0). Also made minor updates to this docs file, and added new 1.5 screenshots.

#### April Updates

After its March release, this version was repackaged—most recently on **Apr-29-15**—with:

1. Assorted documentation changes in this file (formatting, typos, etc.), and 3 new/updated screenshots (now lost to the ravages of time...).
2. A comment about reading and writing calendar files all at once in [icsfiletools.py](http://docs.google.com/icsfiletools.py). See generate\_ics\_files().
3. A new example-only file related to the prior item, [\_\_chunkio.py\_\_](http://docs.google.com/__chunkio__.py).
4. try/finally wrappers around a read and write in [icsfiletools.py](http://docs.google.com/icsfiletools.py) to ensure closes for non-CPython use (e.g., PyPy). Search for "finally".
5. Minor code changes in [icsfiletools.py](http://docs.google.com/icsfiletools.py) for a very unlikely save of a calendar whose last event has been removed. See generate\_ics\_files().
6. A one-line code addition in [frigcal.py](http://docs.google.com/frigcal.py) to equate the ***Escape*** key ("Esc") with a **Today** button press, for keyboard users. See "<Escape>" binding.
7. A one-line code addition in [frigcal.py](http://docs.google.com/frigcal.py) to force focus to the month window instead of its image (if any) on window restores, for keyboard users. Otherwise, a mouse click is required before using bound keys (like ***Escape***). This subsumes the prior 1.4 month-window lift (raise) operation on restores, which isn't enough to set keyboard focus. See "<Map>" event in the code.

As none of these minor changes impacted normal functionality, a new release number was unwarranted (and as of this release, frigcal will no longer do three-level *X.Y.Z* point releases due to their maintenance overheads). Development on this version is expected to taper off after its final repackaging in late April, 2015.

## Version 1.4: February 10, 2015

### ***Highlights***: edit cancel verify, new launcher, file format, image tweaks

This release adds a grab-bag of usability enhancements, and addresses calendar file formatting. The following lists document this version's minor enhancements. As none of this release's changes impacted any other GUI displays, only a handful of new *screenshots* were taken—the new edit-cancel verify dialog, also showing one of the new month images; the startup message of the new Windows no-console launcher; and the augmented backup/save error dialogs.

### Details

Search on "**[1.4]**" in the code for more implementation details underlying the following changes applied in this release.

#### General Changes

1. **Verify Cancel if any data changed in event edit dialogs:**  Added a close confirmation dialog to the Cancel button in event edit/create dialogs, issued if any changeable field has been modified. Previously, Cancel simply closed the dialog, whether data was edited or not, potentially losing user changes. Closes now happen only if confirmed in a popup dialog, when any input field has been changed (but are still immediate if no inputs have been updated). This verification is also run for the edit dialog window's standard close ("X" or red circle) button.  
     
   *Related note*: as is, the Delete button in event edit dialogs does not verify its operation, because, unlike Cancel, this won't discard user inputs in the form, and Delete does not update calendar files immediately—files are updated only on [quit button](http://docs.google.com/UserGuide.html#quit) clicks which verify their action, allowing Deletes to be abandoned (albeit at the cost of other session changes). Delete could easily verify too, but this may be just enough popups to be dismissed by users blindly. Update does not verify changes for the same reasons (nor does right-click Cut; at some point, users must be trusted, not annoyed!). Cancel the save on quit to back out any inadvertent or unwanted deletes or updates.

1. **Order day's events by creation date year before month:**  Fixed a minor issue regarding the sort order of the events displayed within a month window's day tiles (and in their corresponding event selection lists added in version 1.3). Event labels in each day are now are ordered and thus grouped by calendar filename, and then by creation date (**year#**, month#, day#) within calendar group. Formerly, these events were instead inadvertently grouped by creation (**month#**, day#, year#) within calendar group, such that an event's month was more significant to ordering than its year (e.g., January, 2015 would appear above December, 2014 on a day's tile). This was due to the order in the tuple returned by the event class's **\_\_lt\_\_** operator method, used by sorts; this method now correctly places year first.  
     
   That said, this change is more minor *enhancement* than repair. Most users probably didn't rely on—or even notice?—the date ordering of events, for 3 reasons: a given day typically does not display enough events to make order significant (though the 1.3 [selection list](#2bn6wsx) makes days with many events more manageable); the intended creation date ordering within a calendar group was only tersely documented (calendar filename grouping was emphasized); and it's not clear whether an event's creation or modification date should be used for this ordering (creation date is preferred when present, but this seems arbitrary). Events could also sort on creation *time*, but are not; time is generally ignored by frigcal. See also the related ordering TBD [below](#1ci93xb).

1. **New no-console launcher for Windows:**  Added a new script, **frigcal-noconsole.py**, which launches the main [frigcal.py](http://docs.google.com/UserGuide.html#programfiles) script without opening a console messages window on Windows platforms. Drag this out to a desktop shortcut on Windows if desired. This launcher spawns the main script, and issues a startup message window that closes automatically after a brief timeout. The console is mostly meant for tracing and debugging, and not generally useful for end users. See the launcher script's docstring for more on Windows consoles; it's also possible to suppress the Windows console by renaming the top-level script with a ".pyw" extension, or marking a shortcut to the ".py" as Run Minimized, but the new launcher is more automatic and allows the ".py" to still be used when a console is desired.  
     
   *Three related notes*: First, rather than closing its message window via a simple *timeout*, the launcher could be more precise (e.g., delete and poll for a file to be created, or use an IPC tool like sockets); that seems overkill in this context. Second, *Linux and Mac* users may use the new script too (after making frigcal.py executable), but there's not much reason to do so; scripts in these platforms are generally launched with shell command lines or platform-specific devices. Finally, some error messages or context may go *only* to the console (e.g., [configs file](http://docs.google.com/UserGuide.html#configfile) syntax errors did, until [1.5](#_z337ya)); despite its seeming inconvenience, you may still want to run console mode in some contexts, especially if errors close the program or GUI silently.  
     
   To put the preceding point more strongly: *system errors happen*. They can be triggered by in-use locks, read-only permissions, device failures, and more. [Version 1.5](#_z337ya) later added GUI popups for more types of startup and backup/save errors to make handling more user-friendly, and avoid silent shutdowns in non-console mode. Still, some error context text (e.g. tracebacks) may appear only on the console, and may be crucial in debugging. Although frigcal is generally very robust, and goes to great lengths to save multiple backup copies of your calendar data for safety, running without its console may limit options on some errors.
2. **Drop extra '\n' added by tkinter at end of description field:**  The tkinter Text widget always adds an extra **'\n'** newline at the end of the full text fetched from it. These are now forcibly stripped—dropping all '\n' (only) at the text's end, and putting one '\n' back in case some ICS parsers balk at empty (but legal) field values. Prior to this change, the system added one new '\n' to an event's description in the calendar file per event update or paste. This was harmless, and the extra space trivial (just 2 bytes per '\n'), but these could accumulate needlessly. Stripping doesn't effect change tests, because it is immediate on text widget fetches. *Tradeoff*: this could get Text to END+'-1c' instead of stripping, but strips drop any extras already present from prior releases (albeit dropping any intended but pointless blank lines at the end).
3. **Help on backup/save errors:**  In the very unlikely event that a calendar backup/save operation fails, the resulting error dialogs for backup and save are now augmented to include a pointer to the [new help text](http://docs.google.com/UserGuide.html#saveerrors) for more details on restoring from automatic backups and other possible resolutions.

1. **Retain literal backslashes in user-input text:**  Added a new [configs file](http://docs.google.com/UserGuide.html#configfile) setting, *retainLiteralBackslashes*. When **True** (its default), data translations ensure that any literal **'\'** characters in text entered in the GUI's Summary or Description fields are retained verbatim when data is stored to and fetched from calendar files. When **False**, no data translations are performed, but some '\' characters in user inputs will be interpreted by the underlying ics library as escape sequence prefixes, and discarded when saved to a file and loaded in a future session. Although '\' characters are likely rare in user-input text, the new setting allows them to be retained or interpreted explicitly. Retaining—the default—makes sense in most use cases, as user-input is generally meant as entered.  
     
   In more detail, to retain backslashes, the system escapes and unescapes them using common quoted-printable notation—translating any and all '\' to and from '=5C'. You won't see these replacement characters in the frigcal GUI, as the translation happens automatically on calendar file save and load. For example, a user input [a\nb\nc\;d\,e"f\"g] becomes [a=5Cnb=5Cnc=5C;d=5C,e"f=5C"g] in files only, and is restored to its original form when reloaded in a new session. Without this step, most literal '\' characters would be dropped, and a two-character literal '\n' translated to a one-character newline.  
     
   This translation is *switchable* because: (**1**) you may wish your backslashes to denote escapes (e.g., allowing an entered '\n' to become a newline); and (**2**) the escape mechanism risks loss of *file or data portability*, if you use a calendar file containing such escapes in a GUI that doesn't recognize or undo them. The latter is a non-issue if there are no '\' or frigcal is the sole file user; its likely worst case just means you may see rare and isolated '=5C' instead of '\' in another GUI's event fields. See source file [icsfiletools.py](http://docs.google.com/UserGuide.html#programfiles) for more details; while functional, this may prove to be a temporary workaround, pending a better way of dealing with the underlying library's data mutations.

#### Month Image Changes

1. **Explicitly exclude subfolders in the month image files folder:**  When loading and displaying a month's image, the program now explicitly ignores non-files in the [month images folder](http://docs.google.com/UserGuide.html#images). This makes it less error prone to store alternative image sets in subfolders nested within this folder, to be copied up or selected via [configs file](http://docs.google.com/UserGuide.html#configfile) settings. As it was, the system simply fetched, sorted, and indexed the image directory listing's names on month number without regard to entry type. This allowed some subfolders, but only accidentally—as long as subfolder names were all lexically greater than all month image filenames. This was true if image filenames all began with "NN\_" digits following the pattern of images shipped, but this is unreliable (e.g., a "02\_X" subfolder name that sorted among the true image names would have caused exceptions for its relative month). ***Update***: as of [version 1.5](#_z337ya), non-image *files* are also ignored in image folders.
2. **Alternative month images sets and online zip files:**  Added additional and alternative sets of month images designed and contributed by third parties. These image sets are shipped in the AlternateMonthImages subfolder of the program's [MonthImages](http://docs.google.com/UserGuide.html#datafiles) folder, and enabled by the GUI's [Images](http://docs.google.com/UserGuide.html#images) toggle. To use these instead of the original set, either copy their files up to MonthImages, or set the image file path variable in the [frigcal\_configs.py](http://docs.google.com/UserGuide.html#configfile) file to point to a set's subfolder. See that file's "imgpath" setting for examples, and the MonthImages folder's [README.txt](http://docs.google.com/MonthImages/README.txt) for more details.  
     
   Alternate image sets—possibly including others not shipped with the program—are also available *online* as zip files at this location: <http://learning-python.com/AlternateMonthImages>. Download a zip file from there, and unzip its image files into your [images folder](http://docs.google.com/UserGuide.html#datafiles), either directly, or as a subfolder whose name you list in the [configs file](http://docs.google.com/UserGuide.html#configfile). The program's main zip file includes the original images set and a handful of alternates, but larger sets may not be included due to their added size. Watch for additional image set zip files to appear at the online collection in the future, as time, interest, or contributions allow, and see the MonthImages folder's [README.txt](http://docs.google.com/MonthImages/README.txt) file for pointers on creating sets of your own. Item #7 above allows for arbitrary image set subfolders, whether shipped or custom.  
     
   ***Update***: due to lack of time, as of version 1.6's release, the *online* alternative image sets have not yet been updated or maintained. Barring new activity on this front, use the image sets shipped with the frigcal package's zip file, or create some of your own. See the new support for PNGs without Pillow in [version 1.6](#_44sinio) which makes custom sets easier to deploy.
3. **Automatically hide/unhide image windows with their month windows:**  When a month window is minimized/restored, automatically minimize/restore its associated [month image window](http://docs.google.com/UserGuide.html#images), if its image is enabled. Image windows are strongly associated with their month windows logically. This change makes the pairing literal—treating an image as a *dependent extension* to its month window, automatically hidden/unhidden with the month in tandem. Image windows may still be hidden and restored independently, but month window hide/unhide operations are always propagated to the associated image. Without this linking, images loitered on the display disjointedly when months were minimized.
4. **Month image windows: non-resizable, position configuration:**  Make [image windows](http://docs.google.com/UserGuide.html#images) non-user-resizable; resizing was pointless, as the program sizes the window to its image, but never vice-versa. Also, add a new [configs file](http://docs.google.com/UserGuide.html#configfile) setting, "initimgposition", giving the initial position of all image windows (e.g., to always open a horizontal image at the '+0+0' upper left corner initially). By default, the underlying window system still chooses starting positions. This mirrors the existing position option for month windows. Using a fixed starting position means image windows may overlap each other when opened, but this is irrelevant to single month-window sessions, and all windows may be moved freely. Making a window non-resizable also makes it non-maximizable on Windows, but logically so; minimize/restore still work.

#### Non-Functional Changes/Notes

1. **Code refactoring—move two dialogs' code to classes:**  For readability, moved the code for the cut/copy and select-list dialogs from their former nested appearance in event handlers, to new top-level classes. This also allows modal wait-state code to be factored out to a common Dialog superclass, and shared by all 5 dialog classes (the 2 new classes, plus the 3 preexisting event edit dialog classes), instead of being repeated in-line; it's not much code, but redundancy is almost always a Bad Thing in software—multiple copies imply multiple future edits.
2. **Code cleanup—rename callbacks "actions":**  To avoid overloading the term "event" (used for user-created events on days), renamed callback handlers to use "actions" universally. For example, the arguably confusing method "register\_event\_events" became "register\_event\_actions".

1. **TBD—event reordering isn't immediate on adds:**  There remains an open but minor event-in-day ordering issue, related to item #2 [above](#1y810tw). In short, a day's events are not reordered immediately when a new event is added (by Create), but only on the next refresh of the containing month's display. It would be simple to refill the entire month window, but that may be slow enough to "flash" the display on slower machines. A more custom coding to refill just one day (in each window displaying the subject month) would be better, but seems complex enough to warrant a pass for now, as the ordering of events on a day tile seems likely to be minor or irrelevant to most users. As is, navigate forward and back (or select Today for the current month) to refill and reorder if desired.

## Version 1.3: December 18, 2014

### ***Highlights***: event selection lists, cut/copy context, tear-offs dropped

This release focuses on the new event selection list dialog, but also adds more usability extensions.

### Details

The following list chronicles this version's changes; search on "**[1.3]**" in the code for more implementation details:

1. **Event selection list dialog**: Specialized *left-clicks* on the ***day number area***—the area with the numerical day number at the top of each day frame of month windows, above any events for the day—to open a new event selection dialog. The dialog displays all the events in that day in a scrollable list, and supports view, edit, and cut/copy actions on an unlimited number of events.  
     
   This new dialog is designed to look and act nearly the same as a day frame in a month window to make it intuitive. For instance, **clicks** on an event's summary text in the list work largely the same as clicks on event summary fields on day frames, with the following triggers and actions:

| Single left-click (tap) on event list item | Open a normal View/Edit dialog for the event clicked. |
| --- | --- |
| Single right-click (tap+hold) on event list item | Open the normal Cut/Copy dialog for the event clicked. |

1. In addition, selection lists and their items also mimic the **appearance** of day frames and their event fields, by applying the same background *color* and *font* configurations used for days and events, and *colorizing* list item events by calendar/category just as on day frames. The net effect is a dialog that is essentially a larger and more flexible version of a month's day frame, opened on left-click request.  
     
   This enhancement addresses the physical size limitation of day frames, which by nature constrains the number of events that can be displayed—a former TBD and [futurism](#_ihv636). If there are too many events in a day to display all in the day's frame widget, you may now view and edit any of them by clicking on the day number area to open the new selection list. This can be useful in general, but may be especially pertinent on small-screen devices such a tablet.  
     
   The new listbox dialog is *modal* (blocking) to avoid the need to update potentially many listboxes. It also includes a **Create** button for adding a new event on this day via the normal Create dialog—a fallback option which works just like just like a left-click on the *day*'s background in general, and allows new events to be added even if no more day frame background is visible below events. The new selection list dialog is skipped if there are no events on a given day, making a day number click the same as a day frame click, just as before.  
     
   *Implementation Notes*—this feature works as planned, but comes with a few tradeoffs:

* + Both single left- and right-clicks in the listbox always activate the item under (really, *nearest to*) the mouse click location. This model is simple, and the new context label on Cut/Copy dialogs helps clarify a subject after right-clicks (see ahead in this section). Nevertheless, this is a subjective choice. This dialog could instead use double-left-clicks for activation, reserving single-left-clicks to *select* an item to be activated on right-clicks. That would require extra (and per the next note, pointless) user steps, though, and seems more likely to confuse.
  + Although the event selection list looks and acts nearly the same as a day frame in a month window, the mimicry is not fully complete. The behavior of clicks in the list itself is not specialized by *click-mode* (single left- and right-clicks are used for both 'mouse' and 'touch'); and a left- or right-click outside (below) the list's items does not act as a *create* or *paste* as it does on day frames (it instead simply uses the *last* item in the list for the click action, the default behavior for listbox widgets). On the other hand, the forgery is close enough to be intuitive, and a distinct single-left-click for 'mouse' mode seems pointless, given that summary text is not editable in a list as it is on day frames.
  + The series of *interactions* for right-clicks in the listbox can be fairly long, and potentially confusing: they may span three actions and dialogs, or more—left-click on day number, right-click in list dialog, click in Cut/Copy/Open dialog, and possibly an action in the View/Edit dialog for Opens. As this sequence is envisioned as an only rarely used failsafe to support cut/copy for days with too many events to display, it probably suffices. If you just wish to edit/view an off-screen event, though, use a simpler left-click in the list dialog.
  + It may not always be obvious that events are *off-screen*, if events don't appear clipped. Adding an explicit indicator for days with too many events to display is possible, but may require geometry calculations, and would clutter the display too much for a use case that is likely atypical, given the program's intent. Instead, to determine days with too many events, either resize the month window as needed, or simply left-click day numbers for days that appear full.

1. **Cut/Copy context label**: Added the event's summary text as a label on the Cut/Copy (and Open) right-click dialog, to give the user some context of the event being operated on. This is usually obvious after right-clicking an event on a day frame, but becomes murkier when right-clicking an event entry in the new selection listbox.
2. **Drop right-click menu tear-off**: Disabled the default tkinter dashed-line in the Cut/Copy/Open menu, as it's pointless in a modal dialog that is immediately erased (it's useful for quick access to actions in more persistent menus).
3. **Better event dialog titles**: Changed the action names in event dialog titles for clarity. The former "View/Update/Delete Event" is now simply "View/Edit Event", and the former "Create/Paste Event" is now specialized by usage mode to be either just "Create Event" or "Paste Event," better differentiating its role. The dual "Create/Paste" seemed especially dubious, as it was not always clear that prefilled but editable data was present for a paste; the dialog seems otherwise difficult to distinguish from an edit or create.
4. **Code cleanup**: Edited code to use more descriptive names where warranted, especially for callback handler names that now pair triggers with actions. Also updated some internal documentation for clarity. Considered splitting off event dialog and view-date classes to separate modules, but punted—this is a tradeoff at best, as it can make searches more complex (you must search N files and windows instead of 1), and adds module coupling and conceptualization complexity (especially given the strong component relationships).
5. **Docs, screenshots**: Updated this file for this version's changes, and rewrote much of it substantially (an open-ended process). Took new screenshots for changed content in this release only—the new selection list, the new event dialog titles, and redesigned right-click dialog and menus. Prior screenshots are from 1.2, but were not retaken, as their content has not otherwise changed, apart from the "1.2" in their titles and the redesigned doc page in some backgrounds.

## Version 1.2: October 14—November 10, 2014

### ***Highlights***: icon, help, missing configs, read-only text, day shade

This release adds an assortment of usability features inspired by regular use.

### Details

The following lists itemizes this version's changes. Search on "**[1.2]**" in the code for more implementation details.

#### October Release

1. **Windows icon**: Added a program icon for Windows, "frigcalset.ico" (now "icons\frigcal.ico"), generated using the new Python-coded [**iconify.py**](http://learning-python.com/iconify.html) example script. Attached to all top-level windows automatically; can also be attached to a frigcal.py desktop shortcut manually, via right-click + Properties + Change Icon.
2. **Help button**: Added a "**?**" help button at upper right, that pops up this HTML file in a web browser portably, via the Python *webbrowser* module.
3. **Start position**: Added an **initwinposition** setting in the [configurations](http://docs.google.com/UserGuide.html#configfile) file for month windows' initial position, in pixels offset from the screen's upper left corner. This may help on tablets with smaller screens, but causes each [Clone](http://docs.google.com/UserGuide.html#clone) to pop up at the same spot.
4. **Allow missing configs**: Wrapped the user [configurations](http://docs.google.com/UserGuide.html#configfile) module in an attribute **proxy class**, such that missing settings issue warnings and use defaults, instead of causing the program to exit (syntax errors and most invalid settings in the file were already handled). Configuration settings may be missing in a configs file due to user error or upgrades to newer versions with new settings (e.g., prior item). Code: [sharednames.py](http://docs.google.com/sharednames.py) (changed), [attributeproxy.py](http://docs.google.com/attributeproxy.py) (new).

1. **Read-only footer text**: Made the [footer's](http://docs.google.com/UserGuide.html#footer) overview text area non-editable, as any changes made there would be ignored. Requires the Text widget's **state** to be set to **normal/disabled** around deletes and inserts in *onEnter\_Event()*. This widget has no readonly state, and disabled state prevents even programmatic changes. On Windows and Linux this works well: it does not alter the text's appearance, and still allows copying and pasting text from the footer (via Ctrl-C/Ctrl-V on Windows and highlight/middle-click on Linux), but nothing else. This is contrary to most advice seen on the web—a forum which also made the curious recommendation of an infinite loop instead of *wait\_visibility()* for Linux dialogs. This change may or may not have unintended consequences on Macs; please report any issues.
2. **Move day shade on mouse single clicks**: When click-mode is "[mouse](http://docs.google.com/UserGuide.html#daygridmouse)" in the [configurations](http://docs.google.com/UserGuide.html#configfile) file, a single left-click on a day now moves the current-day shading to that day; a double left-click still opens the Create dialog as before. The "[touch](http://docs.google.com/UserGuide.html#daygridtouch)" mode has only single-clicks for days (which both moves the shading and opens the dialog), and other day and event clicks already moved the shading. The shading is only a visual cue, but useful nonetheless. See also the [version 1.6](#_44sinio) change that makes the current-day shading's color configurable.
3. **Minor "touch" mode fix**: Fixed a number-arguments code error that disabled event single-left clicks when click-mode was set to the non-default "[touch](http://docs.google.com/UserGuide.html#daygridtouch)" in the [configurations](http://docs.google.com/UserGuide.html#configfile) file—functionality that had not been recently used (user testing matters!).
4. **Other**: Regenerated all screenshots for 1.2; thoroughly overhauled the **docs** in this file; and removed old **README.txt** file, which was just a stub that redirected to this HTML file. (This file was added back in [version 1.5](#_z337ya) for indexes when viewing the unzipped version on the web, and became package install/launch docs in 2.0.)

#### Version 1.2 Point Releases

This version was repackaged with only minor changes that did not warrant a new major release number here or in the GUI:

* **1.2.1, Oct-31-14**: Minor ***doc updates***—primarily, notes in this file about non-unique event IDs, and event summary display sort order.
* **1.2.2, Nov-07-14**: Minor ***doc updates***—in this file (colored headers), and in code files (a handful of typo fixes in program comments).
* **1.2.3, Nov-10-14**: Three minor changes—  
  + ***doc updates*** in this file (an ongoing task)
  + ***warning messages*** enhanced for rare missing-config-setting case in attributeproxy.py
  + ***footer text*** code additions for read-only mode with no significant behavior impact

For the latter of these—(a) set **state=disabled** initially in *onFooterFlip()*, else text is editable from initial footer enable till first fill on hover; and (b) change **state=normal/disabled** around delete in *clearfooter()* called on month navigations, else the delete is ignored (but don't clear the text on month changes anyhow, unless switched on by a new but optional [configs file](http://docs.google.com/UserGuide.html#configfile) setting, "clearfooter"; this was a TBD and the clear was being ignored as intended, but only accidentally).

## Version 1.1: October 3, 2014

### ***Highlights***: smarter saves—changed files only

New in this version: On window [close button](http://docs.google.com/UserGuide.html#quit) presses (a.k.a. "X", red circle, and quit), don't backup and write the file of a calendar if it has not been changed since program startup or its most recent save. Only calendars newly changed (by updates, deletes, adds, cuts/pastes, or summary edits) are regenerated on close; this both is an optimization, and further minimizes data loss risk. If no calendars have been changed, the backup/save dialog is skipped altogether, and the exit verify appears. Search on "**[1.1]**" in the code for changes.

## Version 1.0: September 26, 2014

### ***Highlights***: initial release

Fully functional. [This version](http://docs.google.com/docimgs/_older-images/x-old--v1.0-win7.png) was current when frigcal was first posted on [python-announce](http://docs.google.com/announce-first.txt).

# Project Goals and Review

As a wrap-up, this section provides a few words of perspective on this project's goals. Because this program was written as a personal-use tool, describing its motivation requires first-person narrative. Much of this section is fully optional reading if you're not interested in personal experience or opinion, but see its final part for future enhancement ideas.

In this section:

* [Why frigcal?](#_3o7alnk)
* [Design and Results](#_23ckvvd)
* [Future Directions?](#_ihv636)
* [A Word on Extensions](#_41mghml)

## Why frigcal?

I wrote this program because:

1. **Sunbird trashed my data.**  
     
   The open source [Sunbird](https://support.mozilla.org/en-US/kb/calendar-lightning-and-sunbird-faq) calendar program erroneously lost 10 years of data in my main iCalendar file—it dropped all data in the file during one of its very frequent updates leaving it completely empty, and I had to resort to restoring a weeks-old manually backed-up archive file copy. *This should never happen*: something as important as calendar data should be saved only on request or exit, and should be backed up before being overwritten. As is, Sunbird updates an iCalendar file on every edit, with no backup of any kind.  
     
   Sunbird also provides functionality I never used, and changes the number of weeks in its month display dynamically, which was visually confusing. Given its fixed interface choices and looming data loss risk, using Sunbird was no longer an option. In its defense, Sunbird is no longer maintained and has been subsumed by a much broader solution, and has other usage modes that may not use iCalendar files directly; neither of these solutions were relevant to my goals.
2. **Other free options seemed too simple, too complex, or too connected.**  
     
   I could find no other open source or free alternatives that suited my tastes and needs. Some seemed toyish—possibly because Outlook's dominance in the domain limits developer interest. Others seemed either radically over-engineered (e.g., [Lightning](https://www.mozilla.org/en-US/projects/calendar/)) or too web-centric (e.g., [Google Calendar](http://www.google.com/calendar)): I need just a calendar, not a grandiose organizational paradigm, and not a system that requires a network connection, web browser, and account login just to look up a date or event.  
     
   Email integration seems overkill for a calendar, and people really aren't always online, despite what you may have heard. Moreover, reworking any of the open source options would be much more work than starting from scratch, especially when coded in other, more tedious programming languages than Python. And we'll leave here aside the implications of having to log into a server account managed by a third party whose bread-and-butter is tracking people's lives for targeted advertising, apart from saying that *privacy matters*—"free" online calendar providers are also free to [read your data](http://www.reuters.com/article/2014/03/19/us-google-gmail-lawsuit-idUSBREA2I13G20140319).
3. **Outlook isn't an option for me.**  
     
   Microsoft [Outlook](http://office.microsoft.com/en-us/outlook/) may be pervasive, but has major downsides—it is no longer included in basic Office bundles and must be purchased separately; has quirks that drove me to Sunbird years ago; is far too feature rich for my use case; seems to support the open/portable iCalendar format only grudgingly; and may move to a subscription model in the future, and/or be used to force users to Microsoft's cloud service.  
     
   The latter point is subjective, but alarming; Outlook's owners have lately taken to parroting a "mobile first, cloud first" mantra that seems a dark omen for developers and users alike. Cloud storage especially comes with some very [negative consequences](http://learning-python.com/mergeall/docs/Usage-Overview.html#clouds). More pragmatically, Outlook's utility just doesn't justify the cost and vendor dependence to me. In the same vein, [Apple Calendar](https://www.apple.com/support/icloud/calendar/) (a.k.a. iCal) shares many of Outlook's platform-specific downsides, as well as Google Calendar's cloud-focused negatives (and there are still a few people on the planet who don't use Apple hardware for such reasons).

Taken together with paper calendars' inherent limitations, a custom coded solution seemed best.

## Design and Results

My use case requires only a simple "refrigerator"-style personal calendar, that uses a portable data file format, and allows notes to be displayed and edited on days freely. I don't use or require email integration, task planning, journaling, reminder alarms, meeting invitations, and the like—specific tools and usage modes which the iCalendar standard oddly seems designed to formalize and even mandate. I also don't have events that recur or span multiple days often enough to justify the complexity of their automation in this release. Some of these may be useful to others (and a few may show up here in future [versions](#32hioqz)), but they also scale up program size and complexity in ways that can lead to error states and data loss, and make user interfaces unwieldy and unnatural.

Those goals and constraints make this program limited by comparison to some, but this is by design: it works, does what I require, and doesn't grow wildly complex and error-prone for rarely used features. As open source, it can also be expanded as desired if you miss some utility here; in fact, its code was written with a primary focus on readability for this purpose. There are ideas for extensions in the [next major section](#_ihv636), if you care to experiment yourself.

### ***Metrics***

In its current form, this program is **4,002** lines of code among [11 source files](http://docs.google.com/UserGuide.html#programfiles) at latest count. This includes comments, but not this documentation file, demo files, or any libraries or packages employed; see and run the included [\_\_sloc\_\_.py](http://docs.google.com/__sloc__.py) to count live.

In terms of effort, this system required just **3 weeks** of initial development time, during which 3 versions were coded—first using functions, then using classes for structure, and finally using refactored classes for edits. Later releases naturally consumed additional occasional time, raising the project's total span to **3.5 months**, when that tally was last updated here. (Later stats: the 1.4 release took an additional week and a half; 1.5, 1.6, and 1.7 required roughly 1 week each; and the project has spanned **2 total years** as of version 1.7.)

Those fairly small sizes and short schedules are due in part to the existence of the complete 3rd-party [*icalendar*](https://pypi.python.org/pypi/icalendar) iCalendar parser/generator package in the Python open source world—a commendable library which accelerated development, but on which this program's correctness is also very much dependent. This is an inherent tradeoff in the "*batteries included*" paradigm promoted by tools such as Python. The good news in this case is that *icalendar* has worked without flaw for two years as of this writing.

***Update, version 2.0***: as of frigcal 2.0, the system has grown to **6,528** lines of code spanning **23** source files (per [this](http://docs.google.com/__sloc__.py)), 4 of which are Python-coded build scripts, and development has spanned 3 years of periodic effort. It's still a relatively minor project.

### ***Usability***

Despite such tradeoffs, the *results* seem compelling: besides serving as a supplemental example for readers of my books, I use the *frigcal* calendar GUI on a regular—and often daily—basis, as part of a personal tools set that also includes the [*mergeall*](http://learning-python.com/mergeall.html) directory synchronization GUI, as well as examples from the book [*Programming Python, 4th Edition*](http://learning-python.com/about-pp4e.html), including:

* The [*PyEdit*](http://learning-python.com/pyedit.html) text editor
* The [*PyMailGUI*](http://learning-python.com/pymailgui.html) POP/SMTP email client
* The [*PyCalc*](http://learning-python.com/gadgets/pycalc.png) calculator
* The [*PyClock*](http://learning-python.com/gadgets/pyclock.png) clock widget

The first two of these have grown radically since their genesis as book examples. For more on the book's examples, see [this page](http://learning-python.com/gadgets).

Although there are precoded alternatives to most of the above, custom-coded programs like these serve real-world purposes; are widely portable; have fully known behavior; and may evolve entirely at their author's discretion. As usual, when equipped with a tool like Python and bit of free time and coding skill, one need not necessarily settle for software that is limited, proprietary, intrusive, or buggy.

## Future Directions?

This program works as intended today, but is naturally open to evolution. Here are some ideas for future enhancements, and a few early implementation musings.

**Day's Events List—Resolved in 1.3**: This issue was addressed in version 1.3, with the new event selection list dialog, triggered by left-clicks on the day number area, and described in the [1.3 release notes](#2bn6wsx) earlier in this document. The prior description that follows is retained, as it contains other and still valid workarounds; in the end, this extension proved useful, and perhaps even required for some use cases.

***Prior description***: As described [elsewhere](http://docs.google.com/UserGuide.html#eventsperday1), there is by design a practical, physical limit on the number of events that can be displayed on a given day. This varies by font, window, and screen sizes, but can be just 3 to 5 in some contexts, and even 3 or 4 events per day is enough to limit the [footer](http://docs.google.com/UserGuide.html#footer) text's display on some smaller screens (though details can still be viewed in the event dialog with a click). A popup triggered by a button for days having more events than a configurable maximum is feasible, but would naturally complicate the GUI and implementation.

The workaround mentioned earlier—when needed, combining multiple events into a single event, listing them individually in the combined event's description text—seems reasonable for a personal-use tool. This GUI generally strives to show useful information without requiring extra interactions, in keeping with its "refrigerator" calendar metaphor; additional clicks and popups that encourage long event lists might violate this goal.

**Frozen Executable—Resolved in 2.0**: Version finally 2.0 added self-contained Mac apps (built with py2app), and Windows and Linux executables (built with PyInstaller). The DLL requirement proved moot on Windows, though none of these were simple to build. The prior notes on this follow.

***Prior description***: This programs seems a prime candidate for distribution as a frozen binary, which does not require a Python install step to run. See cx\_freeze, Py2Exe, and PyInstaller for packaging options. At first glance, the MS Visual C++ DLL requirement on Windows may complicate this in some cases (if users must install a DLL, why not just install Python instead?), and Python 3.X support may still be sketchy as well.

**Event Search—Resolved in 2.0**: Version 2.0 adds calendar search as a command-line [script](http://docs.google.com/searchcals.py) that prints dates to be pasted into the GUI's GoTo field. This avoids complicating the GUI.

***Prior description***: It may be useful to provide search functionality that locates events with matching keywords or content. A pop-up selection list would direct the GUI to the month and day of the located and selected event, or to the event itself. This might be particularly useful for calendars with many years of events (like those of this note's author), though use of particular keywords seems unlikely to be consistent in the dim calendar past.

**Mac OS X port—Resolved in 2.0**: Version 2.0 was ported to run with full functionality on Mac OS X. Prior to the port, appearance and behavior were degraded on this platform (e.g., control-click bindings were required to simulate right-click, and colored buttons had to be replaced with labels).

***Prior description***: This program has not yet been formally verified on Mac OS X; a user reported initial success, but testing time and budget has not yet allowed more rigorous validation.

**Images Option Ideas—Resolved over time**: As mentioned [elsewhere](http://docs.google.com/UserGuide.html#images), the Images option's current popup window model is tentative, and prone to change. This was initially intended to imitate the photos that fill the top half of a true refrigerator calendar, but may be prove too distracting and gimmicky in a GUI. A future version could show resized photos on the month display itself, or via a pulldown.

***Update***: the related enhancements added in 1.4 and 1.5—alternate image sets, subfolder and non-image file skipping, initial positioning, and pairing of images to their months for window hides/unhides—make this feature more useful and solid (e.g., it's easier to create new month image sets of your own, and images are now more strongly associated with months in the GUI); see [1.4](#_3j2qqm3) and [1.5](#_z337ya) release notes. Version [1.6](#_44sinio) further improves this story by making the third-party Pillow install optional for some image file types and Pythons (e.g., PNG and Python 3.4+). Given the changes, images now seem a likely permanent feature, though still subjective enough to be optional.

**Recurring Events**: A likely next step is implementing basic recurring events: weekly, monthly, and annual events. Currently, these can be simulated manually by cutting and pasting events on recurring days, just as one would do on a paper refrigerator calendar—this program's target usage model.

That said, automatic recurrence might be implemented with iCalendar RRULE file entries for events; new tables for these events, indexed by weekday, day number, and month+day; and additions in the GUI's event View/Edit dialog to specify repeats. In fill\_events(), recurrence days in the displayed month matching the new tables' entries would display a virtual event, but clicks on these events would all open the original event object for which the RRULE recurrence was created (unless individual instances are allowed to differ, a usage mode that seems both subtle and complex). Recurrence dates that do not exist in a viewed month would also pose dilemmas (e.g., what to do for events that recur on the 30th of every month, when viewing a February?).

In any case, note that the iCalendar standard supports a ***very rich*** range of recurrence specifications; this program would probably be best served by implementing a small subset for simplicity, recognizing and generating RRULE file entries with known recurrence patterns only.

**Spanning (Multiday) Events**: Events spanning days may seem a logical later addition, but may also violate the simplicity goals for the GUI and its code. This raises subtle issue with events that cross month boundaries, and others that seem to stretch the calendar metaphor to its breaking point (e.g., should events that span days also be allowed to recur?). Some of these may be too complex for a basic calendar program, and event cut/paste seems a far easier option for spans—in terms of both implementation and usage.

**Event Drag-and-Drop**: As mentioned [elsewhere](http://docs.google.com/UserGuide.html#dnd), event drag-and-drop move operations were omitted intentionally, due to the general inaccuracies of drag-and-drop on tablets and other touch devices, especially for desktop-metaphor interfaces. Indeed, drag-and-drop can be triggered much too easily, and seems to occur accidentally as often as deliberately in a Windows file explorer. In a calendar loaded with events, unintended drag-and-drops could be common, and are more likely to be annoying than useful. Instead, more deterministic right-click [cut-and-paste](http://docs.google.com/UserGuide.html#cutpaste) operations are provided for moving events between days and months.

Still, ergonomics is always partly subjective, tkinter does support drag-and-drop operations, and this is a candidate for exploration. To pursue on your own as an extension, see Python's *tkinter.dnd* module in its *Lib\tkinter* (or *lib/tkinter*) folder for one interface. This may or may not require GUI redesign.

**Tandem Minimize/Restore**: As described [elsewhere](http://docs.google.com/UserGuide.html#clone), the Tandem button causes all month view widows (and their images) to navigate in synch between previous/next months and years. In principle, Tandem could also cause all month windows (and their images) to minimize whenever any is minimized, and restore whenever any is restored. That is, all windows would open and close in synch if Tandem were enabled. This isn't implemented because it's not clear that it's useful or desirable. Specifically:

* *Need*: Because both screen size and GUI complexity pose natural limits on the number of usable windows, most users seem likely to use a single month window most of the time, and at most two in general.
* *Need*: It's easier to open and close multiple month windows manually than it would be to repeat navigations in multiple windows manually ([images](http://docs.google.com/UserGuide.html#images) already are opened and closed with their months).
* *Validity*: There are valid arguments both for and against this behavior—while it may save a few clicks in occasional multi-window usage, some users may very well *prefer* that individual windows stay open independently.

Hence, this seems too rare a use case and too extreme a policy decision to mandate without user feedback. If you're a user and have any, please contact the system's author (see the top or bottom of this page). Or try coding this extension yourself—see the open/close pairing of images to month windows on "<Unmap>" and "<Map>" events in [frigcal.py](http://docs.google.com/frigcal.py) for hints, as well as portability issues on Linux (see [version 1.6](#_44sinio)).

**Refactor as Attachable Component**: As shipped, frigcal is oriented towards running as a standalone program, albeit with multiple copies (clones) of its main window. It may be a minor translation to recode it as a tkinter widget package, that may be attached to other GUIs as a nested component. Essentially, the 'root' passed in would have to be variable, and not hardcoded as a new Tk() or Toplevel(); the latter would be only precoded use cases for leveraging the code as standalone program.

This wasn't pursued, because the frigcal GUI seems too large to be useful as an attached component. This is especially so when there are events displayed on days. Moreover, frigcal is probably overkill if you want just a view-only calendar of eventless days (though its code may serve as inspiration for such a component). See the book [*Programming Python*](http://learning-python.com/about-pp.html) for more on coding GUIs that serve as both program and component; its [*PyEdit*](http://learning-python.com/gadgets/pyedit.png) text editor and [*PyCalc*](http://learning-python.com/gadgets/pycalc.png) calculator are prime examples of such dual-mode code. PyEdit, for example, serves both as standalone program, and attached component in [*PyMailGUI*](http://learning-python.com/gadgets/pymailgui.png). (See also these programs' newer versions [here](http://learning-python.com/programs.html).)

**Android (and Other) Tablets:** This program has been used on both laptops and Windows 8.1 and 10 tablets. It might also work well on an Android tablet, though most phone screens may be too small for its display to be usable. This would require porting Python's tkinter GUI library to Android, but a port of its underlying Tcl/Tk support is already underway, and a tkinter port would enable a host of open source Python tools. For more details, see [this post](https://mail.python.org/pipermail/tkinter-discuss/2014-September/003668.html), and watch the web for developments on emerging tkinter platforms. At present, this idea seems sorely in need of traction.

Apple iPad tablets are another obvious candidate platform (but their proprietary architecture seems less likely to support this program), and Linux tablets likely run this program as is (though their market remains to be seen). Naturally, frigcal *does* run on all Windows tablets today, as a normal desktop program.

**Using PyEdit for event edits**: Consideration was also given to using a PyEdit component for event description edits in frigcal's event view/edit dialog, instead of the current simple scrolled text; this would add search and cut/copy/paste buttons, but was rejected because it would also add GUI clutter for rarely used utility, and frigcal's simplicity seems one of its most valued features—which will suffice as a segue to this document's next and final section.

## A Word on Extensions

One of this project's underlying goals has been to produce a useful and intuitively usable tool with no more functionality than is required for its intended roles. The temptation to add features like those of the prior section reflects a tension common to all software projects, and merits a note of perspective here.

In the end, the ease with which functionality can be added with a tool like Python is both asset and liability. Programs like this must find a *balance* between features and usability that stays focused on the original use case. Over-engineering seems an unfortunately common downfall in software—especially in the open source domain, where changes often come with little or no analysis of their impact. When unchecked, this can quickly lead to a state known to developers as *feature creep*: a cascade of enhancements that conspires to render a system unusable.

Less technically—but just as importantly—over-automation turns a program's users into detached *operators*, instead of active participants, and comes with often unexpected consequences. Personal anecdote: a company recently failed to make a payment owed to me, because they scheduled it to occur on a nonexistent day in February (comical but true: the check didn't get cut because it wasn't a leap year). Had the company's employees been required to manually post the payment date to a calendar, this probably would not have happened.

Software works best when it is a *tool*, not a *replacement*. This is as true for a basic calendar GUI as it is for systems of broader scope. Engaging users by avoiding unwarranted automation can not only make programs more relevant to real goals, it might just sidestep mistakes that people wouldn't otherwise make. Be it this program or others, the same rule applies: don't complicate things that don't need to be complicated.

|  | [**Books**](http://learning-python.com/index.html) | [**Programs**](http://learning-python.com/programs.html) | [**Posts**](http://learning-python.com/posts.html) | [**Email**](mailto:lutz@learning-python.com) | © *M. Lutz* |
| --- | --- | --- | --- | --- | --- |