Feb. 23, Asmt. #3, xcal, due Mar. 18 @ 11:59 pm and Mar. 21 in class

General submission instructions (subversion)	
Subversion tag	cis2750-a3
What to submit	`makefile, xcal.py, calutil.h/c, caltool.h/c,` pledge.txt and any other source files needed via online submission. It must be possible to type "make" (no target) to build your application, and "python3 xcal.py" to run it. A .py extension is optional on xcal. Drop off your testing checklist on Fri. or in class Mon. Your assignment will not be marked without a checklist! [*checklist coming later*]

Gardner Home | Course Home | Assignments

Warning re Python version:

cis2750.socs is running version 3.4.3 with Tkinter based on Tcl/Tk 8.6.0 and including the Tix GUI library. Python 3 is not backward compatible with Python 2. **Do not develop your assignment using Python 2!** In the past, students have experienced serious compatibility problems when using different versions. Remember that you are fully responsible for ensuring compatibility of your program with the cis2750.socs environment!

Note on evaluation:

Starting from this assignment, all evaluation for marks will be done through your GUI. Unlike the preceding assignments, the lower-level modules will not be individually tested. In general, you can obtain marks for correct front-end GUI functionality, *and* for correct back-end functionality (which can be provided by Python and/or C code), or for only the former in case the latter doesn't work.You will indicate on the testing checklist which features you *claim* to have working; the TA's role is to verify your claim. By using the testing checklist, you can largely predict your mark.

Front-end functionality means that the GUI widgets are present and can be manipulated. For example, if the back end of a control doesn't work yet, you could at least display a message that proves the control is connected to software. Passive widgets that can't be manipulated will not get full front-end credit.

The only elements of style that will be checked are file headers and compiler warnings.

Fixing poor design choices in calutil and caltool

After seeing outputs from caltool, it became clear that a few aspects of the A1 and A2 spec needed tweaks. This is a normal process in any software project. The changes are very minor, but will have a nice effect.

The A2 automarker will be updated to reflect these changes Feb. 29 or later. Watch for an announcement. (The A1 automarker will not be updated, since we won't be running it on your code again.)

Calutil changes

- **parseCalProp():** Don't change parameter values to uppercase. *The original rationale was that we might be looking up parameter values and interpreting them, but caltool didn't end up going in that direction. To be sure, that treatment was consistent with the policy of rendering all case-insensitive text as uppercase. However, this resulted in making the organizer names look ugly.*

Caltool changes

- **calInfo():** Don't look inside any VTIMEZONE components for dates. *It turned out that VTIMEZONE always has a date going back to 1970. That makes the date range printed by calInfo useless on the lower bound, so it's hard to notice an effect from filtering based on a date range.*

- **calFilter():** For filtering purposes, only look at the 4 properties from RFC 3.8.2.1-4, COMPLETED, DTEND, DUE, and DTSTART. *The other 3 properties (CREATED, DTSTAMP, and LAST-MODIFIED) don't affect when the event/todo occurs, so it doesn't make sense to filter based on those dates. On the other hand, they tell when the user has been engaged with the calendar, so it's reasonable for -info to still recognize all 7.*

The xcal application

Now that we have a set of modules for accessing iCalendar files, we can build a visual application on top of them. This application is called **xcal** , which runs close to the ancient UNIX tradition of tacking on an "x" prefix (stands for "X Windows") to denote the GUI version of a command-line program.

xcal will be able to carry out all the operations that are currently done by the four options of the caltool command. We will build the GUI using the Tk toolkit, and use Python to stitch the Tkinter GUI together with our C modules. Python must not be used for data processing (which is done in C), but only as an integration tool that calls various modules and passes data amongst them.

We will pursue the integration of Python with C using two different techniques, since exposure to both will be good experience:

1. To utilize existing caltool command options, your Python program will set up the appropriate caltool command, including its arguments, and then execute it through a system call. (Every kind of scripting language, including shell scripts, can do this, though they have different ways of going about it.) The big advantage of this technique is that it fully leverages code you have already written and debugged. The disadvantage is that Python will not have fine-grained control over the processing; the existing caltool command will be like a "black box" to the Python program.

2. In order to work with an ICS file at the level of components and properties, you will obtain the necessary data by calling calutil C modules directly, almost as if they were Python functions. This technique is known as "extending Python" (**Chapter 19** of *Programming Python*) and is a lot more trouble to set up, but it allows the Python program to exercise full control over the processing.

There are **three significant learning curves** in this assignment, which should not be underestimated: (1) learning Python; (2) utilizing the Tkinter GUI toolkit; (3) calling C functions from a Python program. The third one--crossing the programming language divide to produce a multi-language application--is traditionally a troublesome area. Once you've done it, it's easy; but, as so often in mastering new software tools, there seem to be a hundred wrong ways to do it, and only one way to do it right. Leave enough time to crawl up all three learning curves!

The Tkinter GUI

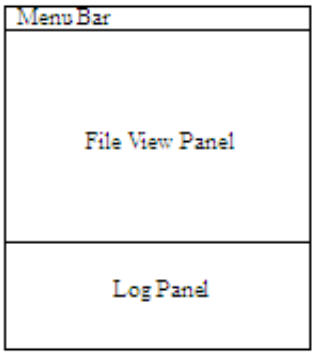
"Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk" (http://wiki.python.org/moin/TkInter; this link is a jumping-off point for Tkinter documentation).

You are free to exercise your artistic and engineering sense in laying out the GUI and determining its exact appearance, but the following items will be required. Items specified as menus or buttons or popups must stay that way; you cannot switch them to all menus or all buttons, etc. See **Section II** of *Programming Python* and the on-line Tkinter manuals.

- You can also use the Tix widget set ("import tkinter.tix"). You might find tkinter.tix.TList "Tabular ListBox" helpful. There are also various "TkTable" implementations on the web.
- **Any other widgets you want to use must be supplied by yourself as suitably credited (as to origin) .py source files in your submission, and they must work on cis2750.socs.** Do not expect any other packages to be installed for you.

The GUI window has a title showing "xcal" and the name of the currently open file, if any. It has the customary menu bar and two panels. The File View Panel (FVP) is scrollable for display of calendar components (minimum of 8), one component per line. Separate from it is the Log Panel for displaying textual output and status messages (minimum of 8 lines). It is also scrollable.

The window should be resizeable by dragging the sides/corners in the usual way, and resizing must not cause disappearance of widgets or display less than the required minimum of items.



Each panel is described in its own section below.

File View Panel

This panel shows components in the currently open file, one line per component, in the order that readCalFile() returns them. This is intended to be a tabular-looking view with rows and columns (though it need not be implemented with a literal table widget). Since there may be >8 components in the file, this view must be scrollable.

The user is capable of selecting one (and only one) component at a time for operations in the usual GUI manner: click to select an item; click again to deselect. If some component is already selected and a different component is clicked, the selection changes to the latter.

A component row is intended to show a few important properties at a glance. These 5 columns are mandatory, should be sized to fit on one line, and must have accurate headings.

- **No.:** A sequence number counting up from 1.
- **Name:** From CalComp.name (e.g., VTIMEZONE).

- **Props:** Properties from CalComp.nprops.
- **Subs:** Subcomponents from CalComp.ncomps.
- **Summary:** For VEVENT and VTODO only, value of SUMMARY property. For other components or if no SUMMARY is available, leave empty.

Here is a sample row:

No.	Name	Props	Subs	Summary
3	VEVENT	6	1	Go to bank

As well as the scrollable display, this panel will feature several action buttons.You may position the buttons wherever you think best. All buttons are only active if a file is currently open in the FVP.

- **Show Selected:** The action is to display in the log all the properties of the currently selected component. This will be done by calling writeCalComp() on the component and capturing the output in the log. This button is only active if a component is selected.
- **Extract Events:** The action is to invoke "caltool -extract e" and capture the output in the log.
- **Extract X- Props:** The action is to invoke "caltool -extract x" and capture the output in the log.

Log Panel

The log is scrollable (e.g., `ScrolledText` widget) and shows any errors from calutils, error messages from caltool, plus any output resulting from actions described above. Messages generated by xcal itself should clearly identify which file and/or line number is implicated. For example, messages that only say "File not opened" (without giving the file name) or "Syntax error" (without giving the line number) are not acceptable.

The log display keeps scrolling up so that the latest messages are in view at the bottom, but the user may operate the scroll bar to reposition the view. In addition, it has a **Clear** button that clears the display.

Menu Bar

There are 3 menus described below.

There is a File menu having 6 commands, some with keyboard shortcuts (Ctrl+):

- **Open...** : (Ctrl+O) If a file is already open in the FVP and there are unsaved changes (Todo menu), require modal popup confirmation to discard the changes before proceeding. Clear the FVP and title, then obtain a filename (see *General rules for all files* immediately below). Run "caltool -info" on the file and display its output in the log. If caltool returns an error, go no further (the FVP and title are unchanged). On good status, call readCalFile() and fill the FVP from the file's components. Display the name of the current file on the title bar of the window.
  - *General rules for all files:* A file browser will pop up for the user to select a filename. If an input file does not exist, a modal popup will display the error. If an output file already exists, a modal popup will appear offering the user the choice of Replace or Cancel.
  - *General rule for all popups/dialogs:* Every popup must be able to be harmlessly dismissed via a Cancel button, which is also activated by pressing the ESC key.

- **Save:** (Ctrl+S) Write the components currently on the FVP into the ICS file named in the title via writeCalComp(). Report the status of the operation including the number of records written.

- **Save as...** : Obtain a filename. Write the components currently on the FVP into the selected file. Report the status of the operation including the number of records written. On good status, the new filename replaces the old title.

- **Combine...:** Obtain a filename as for Open, then call "caltool -combine *that_filename* " using the components on the FVP as stdin. Capture any error output in the log. On good status, replace the FVP with the output of -combine, and leave the title as before. (This command does not automatically save the combined output on disk.)

- **Filter...:** Popup a modal filter dialog with two radio buttons for Todo Items and Events. Next to Events are text boxes to type From and To dates, with all widgets clearly labeled. The popup has two action buttons labeled Filter (only active if a radio button has been clicked) and Cancel. When Filter is clicked, call "caltool -filter *args* " and capture any error output in the log. On good status, replace the FVP with the output of -filter, and leave the title as before. (This command does not automatically save the filtered output on disk.)

- **Exit** : (Ctrl+X) Shut down the program, deleting all temporary files and freeing all C-side memory. *The user must be asked to confirm*. Attempting to close the primary window (e.g., clicking the [X] button) should be treated the same as Exit.

There is a **Todo** menu having 2 commands:

- **To-do List...** : (Ctrl+T) Open a secondary window labeled "To-do List". It displays every Todo item on its own line along with a check box, with scrolling if needed. To declare that an item has been "done", the user can click one or more check boxes and then click the action button labeled Done (which is only active if one or more boxes are checked). This action closes the window and also updates the FVP by removing the selected VTODO components. The window can also be dismissed without changing the file by clicking its Cancel button. In any case, no disk files are changed; that requires a Save.

- **Undo...:** (Ctrl+Z) Popup a modal dialog warning that all Todo components removed since the last save will be restored, and offering two buttons, Undo and Cancel. Refresh the FVP to show the undeleted components. This command is only active when there are unsaved changes in the FVP. There is only one level of undo: *all* "done" todo items, regardless of how many times the Done button was clicked. There is no redo feature.

There is a **Help** menu having 2 commands:

- **Date Mask...:** Obtain a filename for use as a date mask template, and set this path into the environment (os.environ) so that caltool's getdate() can find it.

- **About xcal...** : Popup a typical "About" panel that tells the app name and your name. Don't expose your student number on this. State that the app is compatible with iCalendar V2.0. If you wish, you may also display this popup as a splash screen when xcal is launched, but it must disappear by itself after 3 seconds.

The Python Program

When your program is executed and the GUI is open, the first thing it does is check whether the environment variable DATEMSK is set (see os.environ object). If it is not set, popup a message inviting the user to set this variable, offering "Not now" (meaning "I don't want to set it now, I don't think I'll need it") and "Yes".

On "Yes", execute the Help > Date Mask command.

Command Execution Technique

Remember that caltool expects its input from stdin, and it outputs to stdout and stderr. When your program wants to execute a "caltool" command, it needs to do these steps:

1. Identify input and output files: Input will come from either the recently opened file, or from a temporary file. Stdout and stderr go to temporary files.

2. Format one or more caltool commands in a string containing the appropriate arguments, just as if it were going to be typed on the command line. Add "< input file > output file 2>> error file" to redirect stdin, stdout, and stderr. Use "|" to pipe between commands.

3. Run the command (os.system on **p. 29**). This method invokves the system() library function ("man system") which interprets the command using the **sh** shell. This means that you can insert *shell metacharacters* such as "<", but the syntax has to be what sh likes.

4. Stderr output (in the temporary error file) should always be added to the log display. Depending on the command, the temporary stdout file's contents will either be reopened as an ICS file and used to update the FVP, or it will be added to the log display. Delete the temp files and don't let them accumulate.

NOTE: There are other, possibly more convenient, ways to execute a command with file redirection without going through a shell. See subprocess.Popen and subprocess.PIPE.

Extension Technique

Look over your structure chart and note the few modules that need to be called directly from Python. They are all in calutil.c. To make this work, you need to do these steps:

1. Write "wrapper functions" (also known as "boilerplate") in C for those functions. A wrapper's job is to translate its Python parameters into C-type variables, call the appropriate calutil function, translate its output into Python-type variables, and pass them back to Python. Generally you need one wrapper function for each C function to be called by Python, but there is no reason why one wrapper can't call several C functions. This is spelled out in the function descriptions below.

2. Define a module methods array, so Python knows how to link to your wrapper functions.

3. Use **gcc** to produce a shared object library, e.g., Cal.so, containing your C and wrapper functions. Python can dynamically load and call functions from a shared object library.

   - NOTE on shared object linking with gcc - **shared** option: The .c files destined for a shared object library must be compiled with the **-fPIC** option, so add that to your makefile's CFLAGS.

NOTE ON POINTERS: The wrapper functions shouldn't use any static storage, which means that the Python side has to keep track of CalComp* pointers for any read-in files and deliver those back to the wrapper functions when it calls them. To store a C pointer (64 bits on our hardware) in Python, we can cast it to `unsigned long` in C and build a `PyObject` using `Py_BuildValue` format "k".

The following 3 C wrapper functions (which can all be "static" in file scope) and corresponding module methods array are *suggested* (meaning you are not bound to do it this way as long as you produce the desired result). They will be accessed in Python via the module named "Cal" with "import Cal" and execution syntax like `Cal.readFile()` . All C wrapper functions take the same two parameters, which must be unpacked

using the `PyArg_ParseTuple` function. Use `Py_BuildValue` to build a Python value to be returned to the Python program.

Python call	result = [] # create empty list to hold results
	status = Cal..readFile("foobar.ics", result) # read iCalendar file "foobar.ics"

C prototype	`PyObject *Cal_readFile( PyObject *self, PyObject *args );`
Arguments	char *filename; PyObject *result; PyArg_ParseTuple(args, "sO", &filename, &result);
Description	This function opens a file with the given pathname, calls readCalFile() to return a CalComp* pointer, then closes the file. For good status, it makes the result list as follows: [pcal, [(name, nprops, ncomps, summary), ...]] where: pcal = CalComp* pointer to the entire calendar in memory (should be freed later) [(...), ...] = list of tuples, one per top-level component, each giving the component's name, no. of properties, no. of subcomponents, and SUMMARY property (or else "") For error status, result is untouched. The function returns a string "OK" as good status, or else an understandable error message based on strerror (<string.h>) for fopen, or on the error status of readCalFile.

Python call	status = Cal.writeFile("woofus.ics", pcal, comps) # write selected components of *pcal onto iCalendar file "woofus.ics"
C prototype	`PyObject *Cal_writeFile( PyObject *self, PyObject *args );`
Arguments	char *filename; PyObject *pcal; PyObject *complist; PyArg_ParseTuple(args, "skO", &filename, (unsigned long*)&pcal, &complist);
Description	This function tries to open a file with the given name for writing. If that succeeds, there are two cases based on the length of complist: • If complist is a single number *i* , call writeCalComp(file, pcal->comp[*i*]) to output just the specified component and nothing more. This case is for use by Show Selected. • Otherwise, call writeCalComp(file, pcal) after reducing the comp[] array (described next). This case is for use by Save and Save All, and assumes we'll never save a file with just one component. For the second case, compare the length of complist to pcal->ncomps. If complist is shorter, then modify pcal->ncomps and its comp[] array to contain only the listed components (making a shallow copy of comp[] first). After calling writeCalComp(), restore the original comp[] array from the copy so no storage will leak. The function returns a string "OK" as good status, or else an understandable error message based on strerror

	(<string.h>) for fopen, or on the error status of writeCalComp.

Python call	Cal.freeFile(pcal) # free the storage of the previously read iCalendar file
C prototype	`PyObject *Cal_freeFile( PyObject *self, PyObject *args );`
Arguments	CalComp *pcal; PyArg_ParseTuple(args, "k", (unsigned long*)&pcal);
Description	Call freeCalComp(pcal).

In addition to the wrapper functions above, the following linkage code is needed so that Python can find them when you "import Cal" (note that optional list elements are omitted in CalMethods compared to the sample code shown in the lab session):

```
static PyMethodDef CalMethods[] = {

   {"readFile", Cal_readFile, METH_VARARGS},
   {"writeFile", Cal_writeFile, METH_VARARGS},
   {"freeFile", Cal_freeFile, METH_VARARGS},
   {NULL, NULL} };

static struct PyModuleDef calModuleDef = {

   PyModuleDef_HEAD_INIT,
   "Cal", //enable "import Cal"
   NULL, //omit module documentation
   -1, //don't reinitialize the module
   CalMethods //link module name "Cal" to methods table };

PyMODINIT_FUNC PyInit_Cal(void) { return PyModule_Create( &calModuleDef ); }
```

The Cal.so library needs to contain the wrapper functions, and all the C functions they call in turn. It does not need to have any other functions from calutil.c or caltool.c, but they won't do any harm. It is likely most convenient to simply build it from calutil.o.

Here is some additional documentation for <Python.h> functions that you won't find in *Programming Python:*

```
int PyList_Append(PyObject *list, PyObject *item);
```

Appends the object *item* at the end of *list* . Returns 0 if successful; returns -1 and sets an exception if unsuccessful. Analogous to *list* .append(*item*).

```
PyObject* PyList_GetItem(PyObject *list, Py_ssize_t index);
```

Returns the object at (integer) position *index* in *list* . If *index* is out of bounds, returns `NULL' and sets an `IndexError'.

```
PyObject *Py_BuildValue(const char *format, ...);
```

Full explanation here.

Bonus marks

You can get a bonus mark (0.5-1.0) if the TA considers that your GUI demonstrates superior aesthetic

qualities--i.e., it is attractive to look at. This is unavoidably subjective, and if you can't bear that, then don't spend time pimping your GUI.

Another bonus is available for adding a **CalDAV** menu with 3 commands. The objective is to allow the file in the FVP to be exported to an external CalDAV server that is synchronized with a calendar client (e.g., Thunderbird, Gmail, Blackberry, iCal, etc.) that can put up a nice calendar/todo display. You will be able to add events to that calendar and re-import it from the server into your xcal program. To do this, it is necessary to link xcal with an account that you set up on a CalDAV server such as yahoo.com. Make sure that you give us enough account information to demonstrate this feature. [We're looking into running the CalDAV server ourselves, which will simplify authentication.] You can get 0.5 for implementing either export or import, or 1.0 for implementing both. Export and Import commands should only be active if Account has been filled in.

- **Account...:** [This menu item is subject to change with more investigation.] Popup a dialog that allow the user to enter at least the following 4 items: 1) a Name for the remote calendar; 2) a URL for your account on the CalDAV server; 3) an account name (might be an e-mail address); and 4) a password. All 4 items should be available even if not all of them are needed to connect to your demonstration account. **You should prepopulate the URL text box to make it easier to test.**

- **Export:** Output the FVP as an ICS file and transmit it to the URL, making use of the accound name and password if needed. Display the status of the operation in the log. If the user has a browser open to that account, they should see the calendar/todos updated to reflect xcal's calendar.

- **Import:** Treat this command like Open, in terms of any unsaved changes. Contact the CalDAV account and read its ICS file if possible, reporting status in the log. Try to read in the file. On good status, update the FVP with its contents, and change the title to the account name. The contents can now be saved with **Save as** , but **Save** should not be active (because the FVP does not have an associated disk file).

Incremental development

A suggested road map would go like this:

1. Familiarize yourself with the Tk and Tix widget toolkits, and design your GUI on paper.

2. Convert your GUI design to Python code.

3. Get your Python program working so it successfully displays your GUI. Populate it with enough fake component data to prove that the scroll bars are working.

4. Write the GUI's back end, i.e., the functions that are called when the user clicks on different features of your GUI. Implement your back end as stubs that simply print out messages letting you know that the various functions have been called. Make sure all the menu commands can be picked, and all the buttons clicked. Revamp your layout if it seems ugly or inefficient.

I suggest you finish the steps 1-4 during the first week. The lab session will help you start right.

5. Now shift your attention to the C side: Design your wrapper functions, implementing them as stubs that print out their arguments when called, and pass back some dummy data to Python. Learn how to link them into a shared object library. You'll want to use a makefile, for sure! Change your Python back end so it actually calls these functions, and test this till it works reliably. You've now established two-way communication between Python and C!

6. Modify your wrapper functions so they do their intended job. Now you can read real iCalendar files into your GUI.

7. Complete the functionality for deleting selected VTODO components and undoing.

Finish steps 5-7 the second week. The lab session will help you bridge Python to C.

8. The last step is the "command execution technique": Modify your Python back end to format and execute caltool commands.

Then if you still have time after thorough testing of your finished system, try to improve your GUI's aesthetics and/or implement the CalDAV connection for the bonus.

Gardner Home | Course Home | Assignments