More Python on Series 60

Larry Rudolph
SMA 5508 MIT 6.883 (Spring 2006)
March 10

Where to get information

- www.forum.nokia.com
  - there are a few more documents (like getting started; building an app; C++ extensions) to be gotten from the 1.2 python release
- Around the web, but bunch of older versions around -- be careful
- our wiki, look at IAP 2006

Today's Topics

- Examples
- Screen
- Network
- Active Objects
- Bluetooth
- Callgate

Processes, Threads, Active Objects

- Process: address space + threads
  - A main thread interacts with user interface. Can directly call UI. If blocks, application blocks
  - Heap (shared by all threads)
  - No statics (in DLL), but yes in new: S60-FP3
- Thread: Program Counter + AO + stack (small)
- AO (Active Object): Pieces of thread code that interacts with User Interface

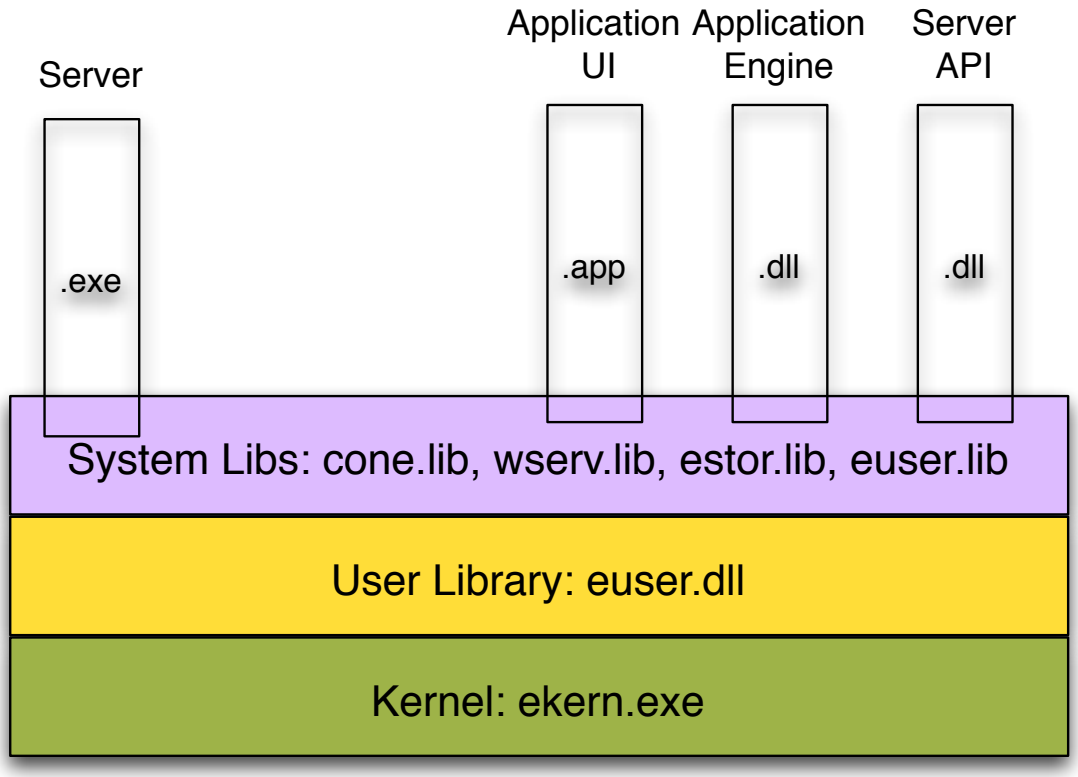
Processes

- Each application and each service (I think) execute as separate processes
  - Each process has its own address space
  - We will not deal with interprocess communication (but could use sockets)
- An application is a process that may have
  - UI and Engine parts
  - Access System and Server APIs

DLL's and API's

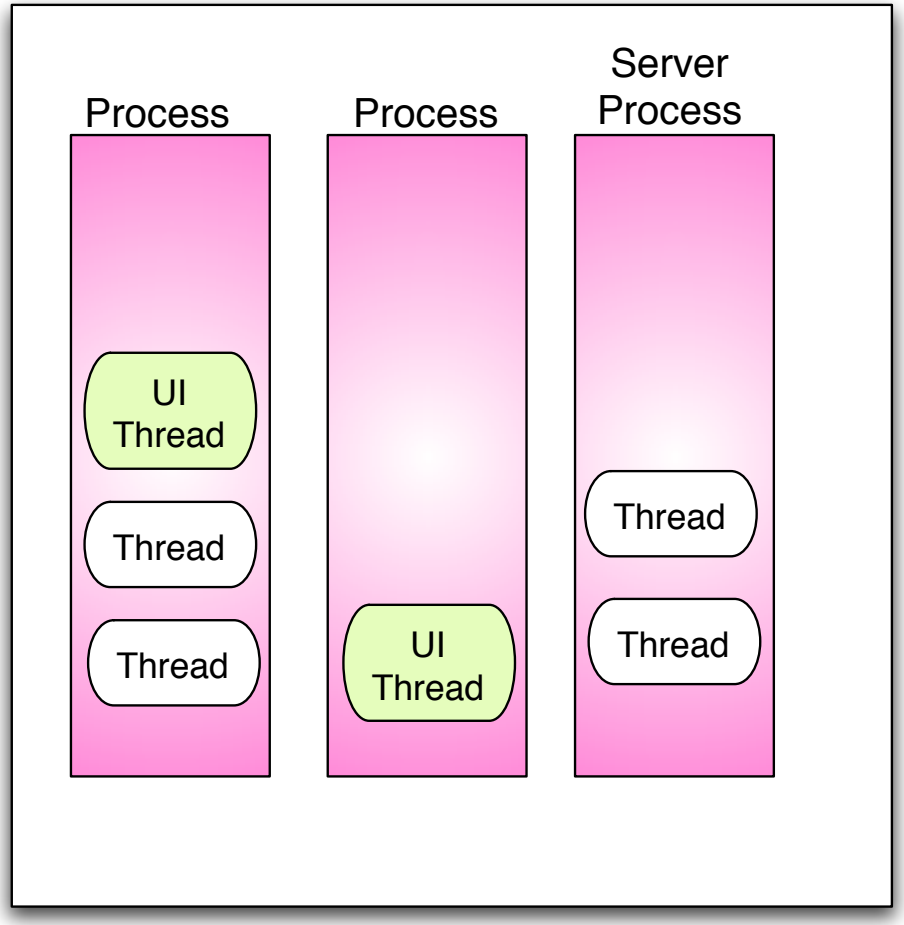
- API: the exported published behaviour a system component exposes to other components
- Symbian OS DLL components can:
  - define an API for use by other components (system libs, app. engines)
  - implement an API defined by a framework
    - GUI applications, device drivers
    - these are plug-in's into a framework

Apps, DLL, API


Server
Application UI
Application Engine
Server API
.exe
.app
.dll
.dll
System Libs: cone.lib, wserv.lib, estor.lib, euser.lib
User Library: euser.dll
Kernel: ekern.exe

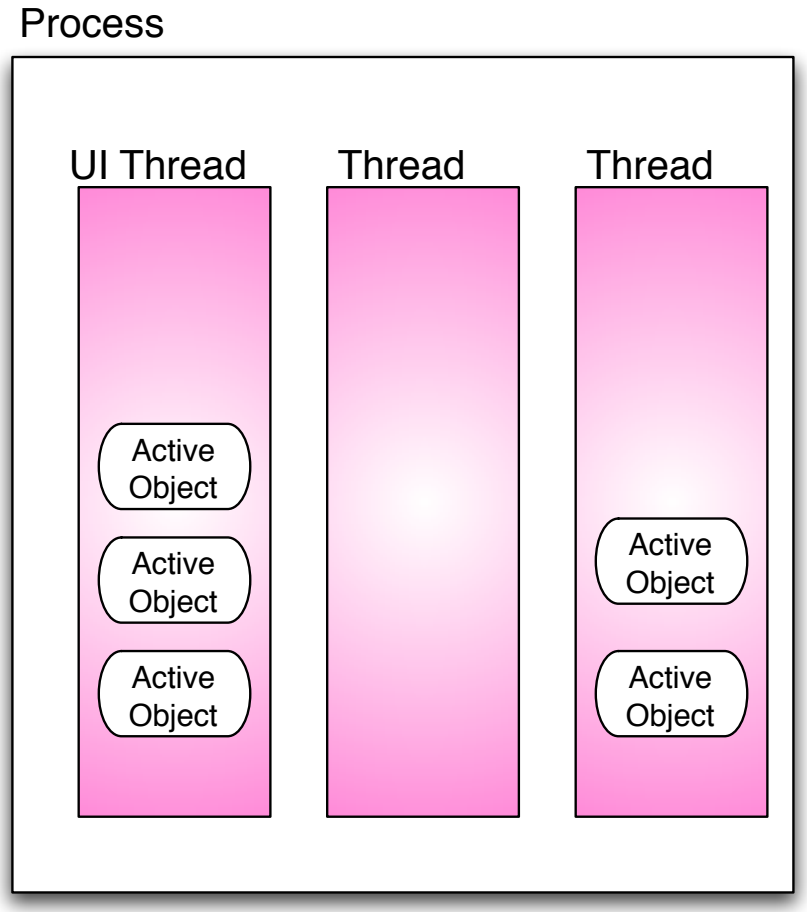
Processes (exe, app)

- an application has only one user interface thread
- sockets & file objects cannot be shared among threads
- why?



Process & Threads

- only one thread in process has access to UI
- sockets & file objects cannot be shared among threads
- why?



What kind of OS?

- Multi-tasking
- Multi-threading
- Real-time

UI Thread

- places objects on screen
- registers callbacks procedures associated with screen & keyboard events
- when event occurs, want to pass control to the callback procedure.
  - what if thread is executing something else?
- Callbacks should execute quickly
- UI thread should spend most of the time idle

Coordination

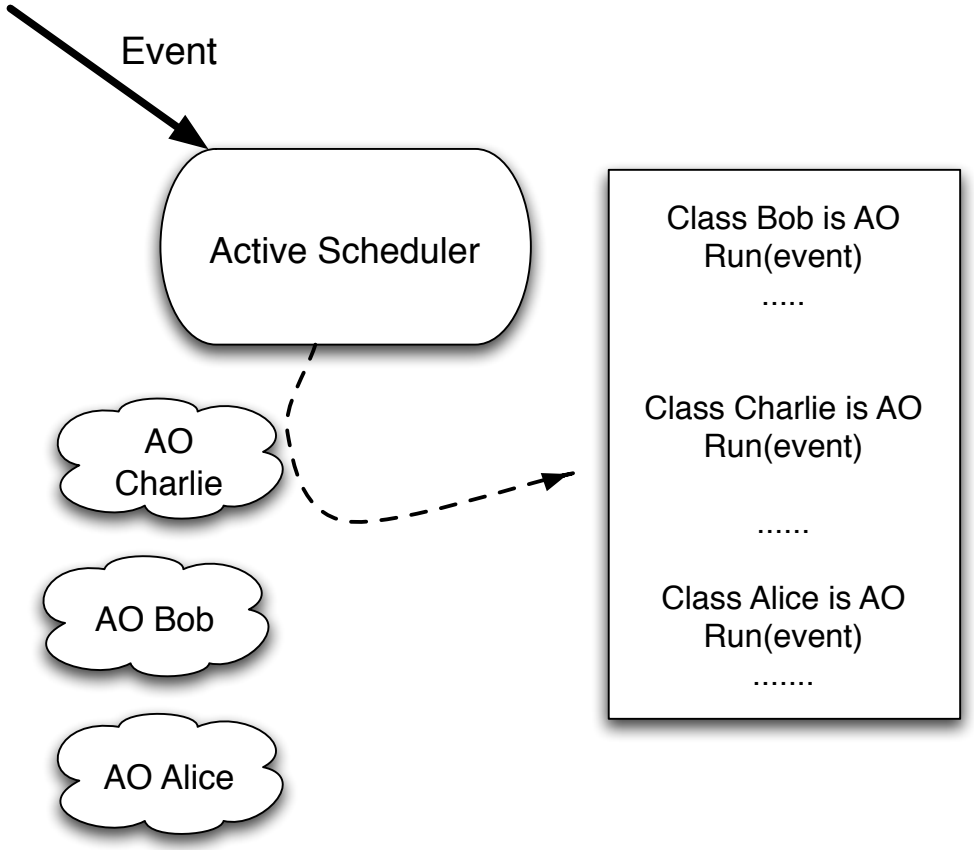
- Don’t use normal thread locks:

```
import thread
lock = thread.allocate_lock()
```

- Whole application gets blocked, since no UI actions would be handled
- Use e32.Ao_lock instead

Active Objects

- If Symbian written today, AO's would be called "listeners"
- Get called by scheduler (have a little bit of state)
- Run to completion then return to scheduler

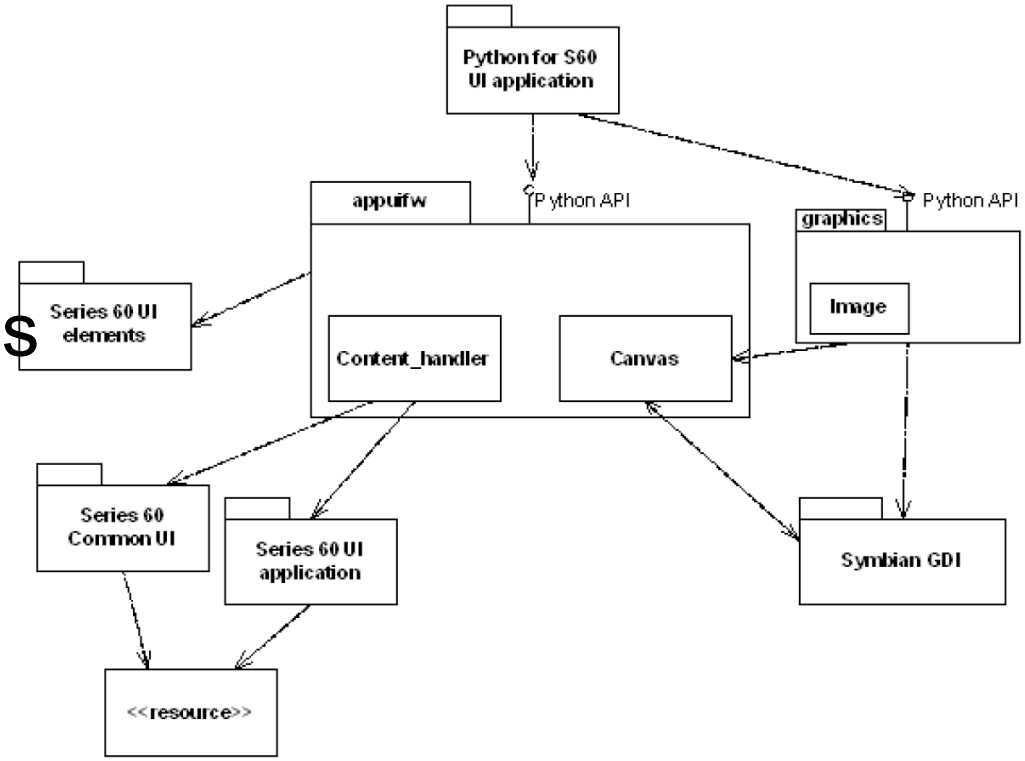


Python's AO

- Previous discussion was for Symbian in general
- Python hides the scheduler
  - but after setting up callbacks, just do a return
- Can control AO by allocating an e32.Ao_lock() and then doing wait() and signal() with this lock

Python User Interface

- This diagram shows the pieces
- Ignore it

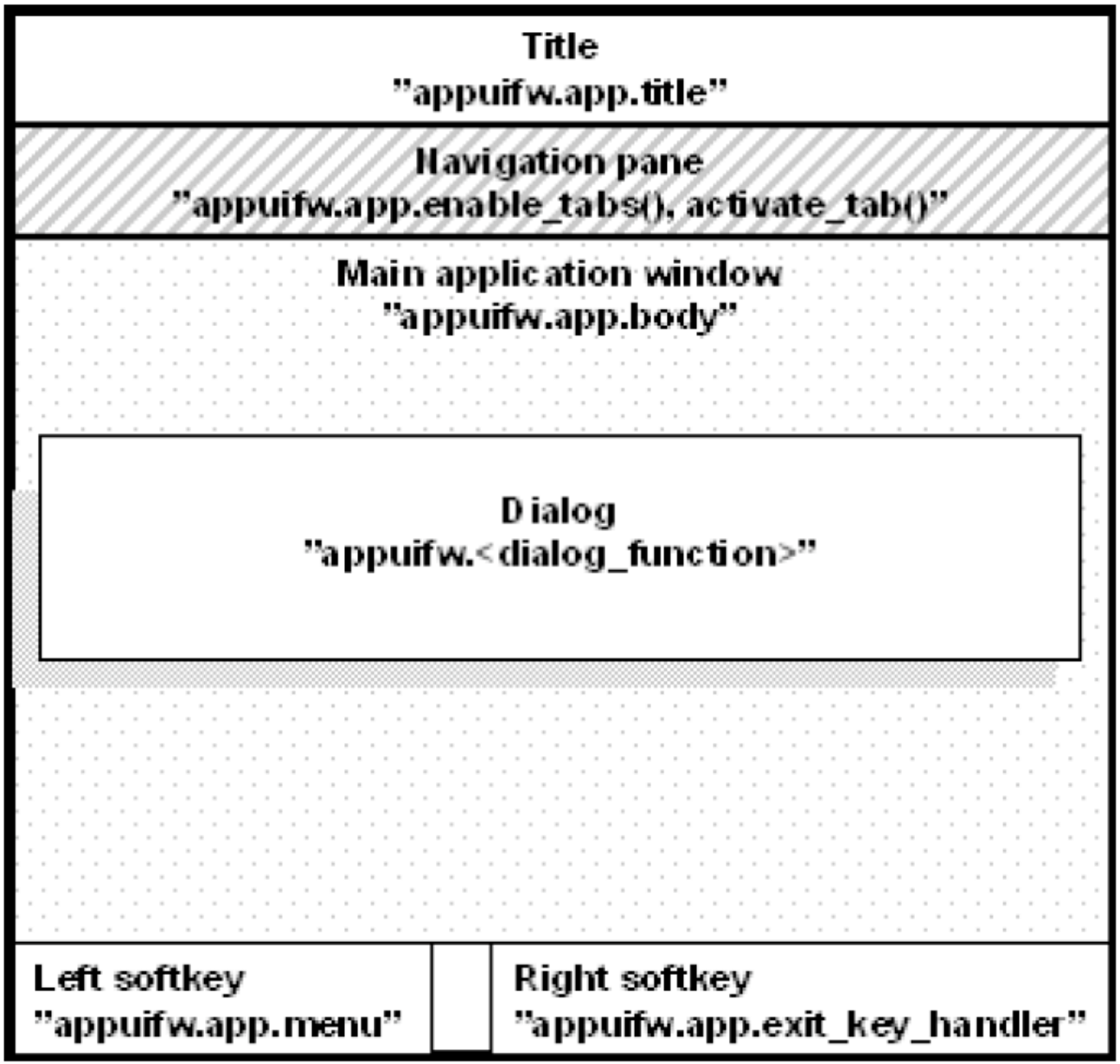


User Interface Approach

- What should we care about?
  - Graphical User Interface (GUI) is big deal
- Small screen ==> make best of poor situation
  - Will screens get bigger? Will Nokia's UI approach scale?
  - What about other input modalities?
- Alternatives: PocketPC, Palm, Blackberry

Nokia's Approach

- Nokia's UI philosophy (are they unique?)
- Uniform across apps; branded look&feel
  - Screen title at top
  - Optional tabs just below that
  - Body (and for pop-ups)
  - Bottom softkeys: Menu (left), Exit (right)


Title
"appuifw.app.title"
Navigation pane
"appuifw.app.enable_tabs(), activate_tab()"
Main application window
"appuifw.app.body"
Dialog
"appuifw.<dialog_function>"
Left softkey
"appuifw.app.menu"
Right softkey
"appuifw.app.exit_key_handler"

SPy60 Approach

- provide option for more usable screen area
- great for prototyping.
  - Use default font & size; minor graphics

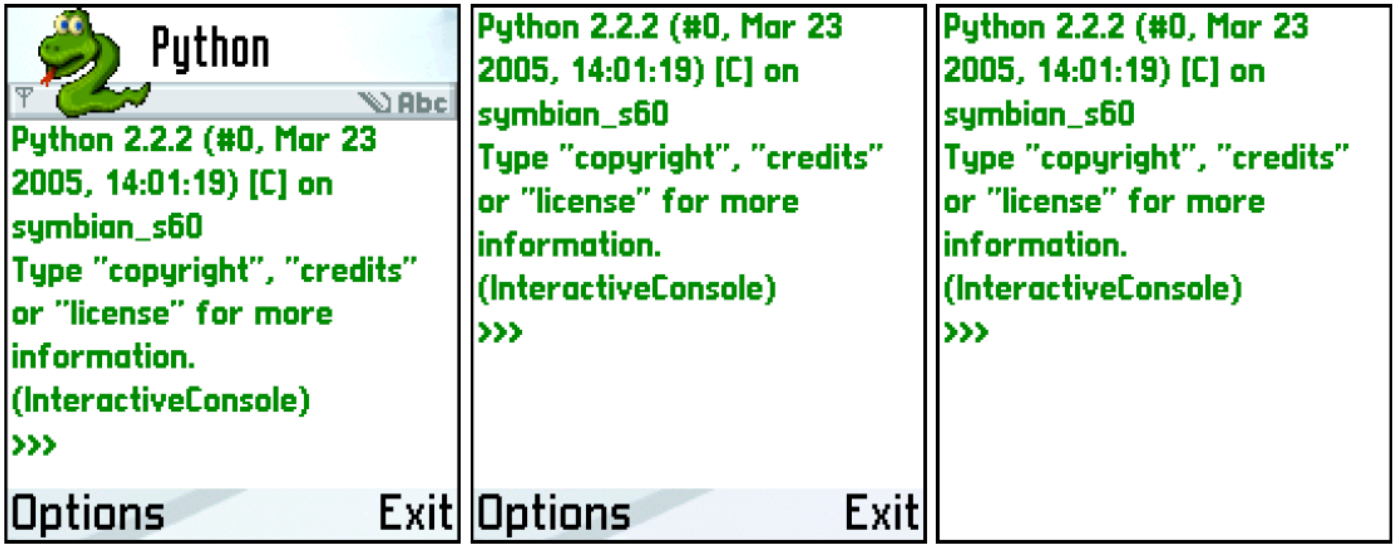


Figure 5.3: UI layouts. left: 'normal', middle: 'large', right: 'full'

Using the screen

- Appuifw contains an instance of the class application, called app

```
appuifw.app.title = u'title of screen'
appuifw.app.screen = 'normal' # size
*.app.body = Text | Listbox | Canvas
*.app.menu = list of (title, callback)
*.app.set_tabs( list of tab names, callback)
```

SMS messaging

- Can send SMS: sms_send(nmbr, mess)
  - limit of 160 characters
- Can access phone's inbox
  - plop it into a list, access fields of mess
- Register callback for whenever mess arrives
- Need to be connect to phone network and need to be running when msg arrives

```
import e32
import appuifw
from MyDataAccess import MyDataAccess

e32.ao_yield()

def format(item):
    # Format the item as a short unicode string.
    return u"" # omitted
class MyApp:
    def __init__(self):
        self.lock = e32.Ao_lock()

        self.old_title = appuifw.app.title
        appuifw.app.title = u"My Application"

         self.exit_flag = False
        appuifw.app.exit_key_handler = self.abort

        self.data = []
        appuifw.app.body = appuifw.Listbox([u"Loading..."], self.handle_modify)

        self.menu_add = (u"Add", self.handle_add)
        self.menu_del = (u"Delete", self.handle_delete)
        appuifw.app.menu = []
              # First call to refresh() will fill in the menu.
```

```
Def connect(self, host):
    self.db = MyDataAccess(host)
    self.db.listen(self.notify)
          # Set up callback for change notifications.

def loop(self):
    try:
        self.lock.wait()
        while not self.exit_flag:
            self.refresh()
            self.lock.wait()
    finally:
        self.db.close()

def close(self):
    appuifw.app.menu = []
    appuifw.app.body = None
    appuifw.app.exit_key_handler = None
    appuifw.app.title = self.old_title

def abort(self):
    # Exit-key handler.
    self.exit_flag = True
    self.lock.signal()

def notify(self, in_sync):
    # Handler for database change notifications.
    if in_sync:
        self.lock.signal()
```

```
def refresh(self):
    # Note selected item.
    current_item = self.get_current_item()

    # Get updated data.
    self.data = self.db.get_data()

    if not self.data:
        content = [u"(Empty)"]
    else:
        content = [format(item) for item in self.data]

    if current_item in self.data:
        # Update the displayed data,
        # retaining the previous selection.
        index = self.data.index(current_item)
        appuifw.app.body.set_list(content, index)
    else:
        # Previously selected item is no longer present, so allow
        # the selection to be reset.
        appuifw.app.body.set_list(content)

    if not self.data:
        appuifw.app.menu = [self.menu_add]
    else:
        appuifw.app.menu = [self.menu_add, self.menu_del]
```

```
def handle_modify(self):
    item = self.get_current_item()
    if item is not None:
        # Display data in Form for user to edit.
        # Save modified record in database.
        pass                    # omitted

def handle_add(self):
    new_item = self.edit_item(ToDoItem())
    if new_item is not None:
        # User enters new data into Form.
        # Save new record in database.
        pass                    # omitted

def handle_delete(self):
    item = self.get_current_item()
    if item is not None:
        # Remove record from database.
        pass                    # omitted

def get_current_item(self):
    # Return currently selected item, or None if the list is empty.
    if not self.data:
        return None
    else:
        current = appuifw.app.body.current()
        return self.data[current]
```

```
def main():
    app = MyApp()
    try:
        hosts = [u"some.foo.com", u"other.foo.com"]
        i = appuifw.popup_menu(hosts, u"Select server:")
        if i is not None:
            app.connect(hosts[i])
            app.loop()
    finally:
        app.close()

if __name__ == "__main__":
    main()
```