CHAPTER 28

Project 9: File Sharing II—Now with GUI!

This is a relatively short project because much of the functionality you need has already been written—in Chapter 27. In this chapter, you see how easy it can be to add a GUI to an existing Python program.

What's the Problem?

In this project, you expand the file sharing system developed in Chapter 27, with a GUI client. This will make the program much easier to use, which means that more people might choose to use it (and, of course, multiple users sharing files is the whole point of the program). A secondary goal of this project is to show that a program that has a sufficiently modular design can be quite easy to extend (one of the arguments for using object-oriented programming).

The GUI client should satisfy the following requirements:

- It should allow you to enter a file name and submit it to the server's `fetch` method.
- It should list the files currently available in the server's file directory.

That's it. Because you already have much of the system working, the GUI part is a relatively simple extension.

Useful Tools

In addition to the tools used in Chapter 27, you will need the wxPython toolkit. For more information about (and installation instructions for) wxPython, see Chapter 12. The code in this chapter was developed using wxPython version 2.6, but will work with the latest version.

If you want to use another GUI toolkit, feel free to do so. The example in this chapter will give you the general idea of how you can build your own implementation, with your favorite tools. (Chapter 12 describes several GUI toolkits.)

Preparations

Before you begin this project, you should have Project 8 (from Chapter 27) in place, and a usable GUI toolkit installed, as mentioned in the previous section. Beyond that, no significant preparations are necessary for this project.

First Implementation

If you want to take a peek at the full source code for the first implementation, it can be found in Listing 28-1 later in this section. Much of the functionality is quite similar to that of the project in the preceding chapter. The client presents an interface (the `fetch` method) through which the user may access the functionality of the server. Let's review the GUI-specific parts of the code.

The client in Chapter 27 was a subclass of `cmd.Cmd`; the `Client` described in this chapter subclasses `wx.App`. While you're not required to subclass `wx.App` (you could create a completely separate `Client` class), it can be a natural way of organizing your code. The GUI-related setup is placed in a separate method, called `OnInit`, which is called automatically after the `App` object has been created. It performs the following steps:

1. It creates a window with the title "File Sharing Client."
2. It creates a text field and assigns that text field to the attribute `self.input` (and, for convenience, to the local variable `input`). It also creates a button with the text "Fetch." It sets the size of the button and binds an event handler to it. Both the text field and the button have the panel `bkg` as their parent.
3. It adds the text field and button to the window, laying them out using box sizers. (Feel free to use another layout mechanism.)
4. It shows the window, and returns `True`, to indicate that `OnInit` was successful.

The event handler is quite similar to the handler `do_fetch` from Chapter 27. It retrieves the query from `self.input` (the text field). It then calls `self.server.fetch` inside a `try/except` statement. Note that the event handler receives an event object as its only argument.

The source code for the first implementation is shown in Listing 28-1.

Listing 28-1. *A Simple GUI Client (simple_guiclient.py)*

```
from xmlrpclib import ServerProxy, Fault
from server import Node, UNHANDLED
from client import randomString
from threading import Thread
from time import sleep
from os import listdir
import sys
import wx
```

```
HEAD_START = 0.1 # Seconds
SECRET_LENGTH = 100

class Client(wx.App):
    """
    The main client class, which takes care of setting up the GUI and
    starts a Node for serving files.
    """
    def __init__(self, url, dirname, urlfile):
        """
        Creates a random secret, instantiates a Node with that secret,
        starts a Thread with the Node's _start method (making sure the
        Thread is a daemon so it will quit when the application quits),
        reads all the URLs from the URL file and introduces the Node to
        them.
        """
        super(Client, self).__init__()
        self.secret = randomString(SECRET_LENGTH)
        n = Node(url, dirname, self.secret)
        t = Thread(target=n._start)
        t.setDaemon(1)
        t.start()
        # Give the server a head start:
        sleep(HEAD_START)
        self.server = ServerProxy(url)
        for line in open(urlfile):
            line = line.strip()
            self.server.hello(line)


    def OnInit(self):
        """
        Sets up the GUI. Creates a window, a text field, and a button, and
        lays them out. Binds the submit button to self.fetchHandler.
        """

        win = wx.Frame(None, title="File Sharing Client", size=(400, 45))

        bkg = wx.Panel(win)

        self.input = input = wx.TextCtrl(bkg);

        submit = wx.Button(bkg, label="Fetch", size=(80, 25))
        submit.Bind(wx.EVT_BUTTON, self.fetchHandler)

        hbox = wx.BoxSizer()
```

```
        hbox.Add(input, proportion=1, flag=wx.ALL | wx.EXPAND, border=10)
        hbox.Add(submit, flag=wx.TOP | wx.BOTTOM | wx.RIGHT, border=10)

        vbox = wx.BoxSizer(wx.VERTICAL)
        vbox.Add(hbox, proportion=0, flag=wx.EXPAND)

        bkg.SetSizer(vbox)

        win.Show()

        return True

    def fetchHandler(self, event):
        """
        Called when the user clicks the 'Fetch' button. Reads the
        query from the text field, and calls the fetch method of the
        server Node. If the query is not handled, an error message is
        printed.
        """

        query = self.input.GetValue()
        try:
            self.server.fetch(query, self.secret)
        except Fault, f:
            if f.faultCode != UNHANDLED: raise
            print "Couldn't find the file", query


def main():
    urlfile, directory, url = sys.argv[1:]
    client = Client(url, directory, urlfile)
    client.MainLoop()

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

Except for the relatively simple code explained previously, the GUI client works just like the text-based client in Chapter 27. You can run it in the same manner, too. To run this program, you need a URL file, a directory of files to share, and a URL for your `Node`. Here is a sample run:

```
$ python simple_guiclient.py urlfile.txt files/ http://localhost:8080
```

Note that the file `urlfile.txt` must contain the URLs of some other `Node`s for the program to be of any use. You can either start several programs on the same machine (with different port numbers) for testing purposes, or run them on different machines. Figure 28-1 shows the GUI of the client.

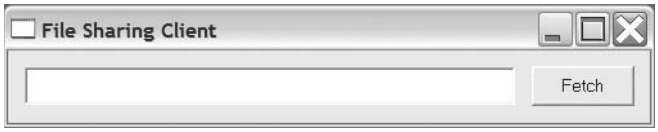

Figure 28-1. *The simple GUI client*

This implementation works, but it performs only part of its job. It should also list the files available in the server's file directory. To do that, the server (`Node`) itself must be extended.

Second Implementation

The first prototype was very simple. It did its job as a file sharing system, but wasn't very user friendly. It would help a lot if users could see which files they had available (either located in the file directory when the program starts or subsequently downloaded from another `Node`). The second implementation will address this file listing issue. The full source code can be found in Listing 28-2.

To get a listing from a `Node`, you must add a method. You could protect it with a password as you have done with `fetch`, but making it publicly available may be useful, and it doesn't represent any real security risk. Extending an object is really easy: you can do it through subclassing. You simply construct a subclass of `Node` called `ListableNode`, with a single additional method, `list`, which uses the method `os.listdir`, which returns a list of all the files in a directory:

```
class ListableNode(Node):

    def list(self):
        return listdir(self.dirname)
```

To access this server method, the method `updateList` is added to the client:

```
    def updateList(self):
        self.files.Set(self.server.list())
```

The attribute `self.files` refers to a list box, which has been added in the `OnInit` method. The `updateList` method is called in `OnInit` at the point where the list box is created, and again each time `fetchHandler` is called (because calling `fetchHandler` may potentially alter the list of files).

Listing 28-2. *The Finished GUI Client (guiclient.py)*

```
from xmlrpclib import ServerProxy, Fault
from server import Node, UNHANDLED
from client import randomString
from threading import Thread
```

```
from time import sleep
from os import listdir
import sys
import wx

HEAD_START = 0.1 # Seconds
SECRET_LENGTH = 100


class ListableNode(Node):
    """
    An extended version of Node, which can list the files
    in its file directory.
    """
    def list(self):
        return listdir(self.dirname)

class Client(wx.App):
    """
    The main client class, which takes care of setting up the GUI and
    starts a Node for serving files.
    """
    def __init__(self, url, dirname, urlfile):
        """
        Creates a random secret, instantiates a ListableNode with that secret,
        starts a Thread with the ListableNode's _start method (making sure the
        Thread is a daemon so it will quit when the application quits),
        reads all the URLs from the URL file and introduces the Node to
        them. Finally, sets up the GUI.
        """
        self.secret = randomString(SECRET_LENGTH)
        n = ListableNode(url, dirname, self.secret)
        t = Thread(target=n._start)
        t.setDaemon(1)
        t.start()
        # Give the server a head start:
        sleep(HEAD_START)
        self.server = ServerProxy(url)
        for line in open(urlfile):
            line = line.strip()
            self.server.hello(line)
        # Get the GUI going:
        super(Client, self).__init__()
```

```
def updateList(self):
    """
    Updates the list box with the names of the files available
    from the server Node.
    """
    self.files.Set(self.server.list())


def OnInit(self):
    """
    Sets up the GUI. Creates a window, a text field, a button, and
    a list box, and lays them out. Binds the submit button to
    self.fetchHandler.
    """

    win = wx.Frame(None, title="File Sharing Client", size=(400, 300))

    bkg = wx.Panel(win)

    self.input = input = wx.TextCtrl(bkg);

    submit = wx.Button(bkg, label="Fetch", size=(80, 25))
    submit.Bind(wx.EVT_BUTTON, self.fetchHandler)

    hbox = wx.BoxSizer()

    hbox.Add(input, proportion=1, flag=wx.ALL | wx.EXPAND, border=10)
    hbox.Add(submit, flag=wx.TOP | wx.BOTTOM | wx.RIGHT, border=10)

    self.files = files = wx.ListBox(bkg)
    self.updateList()

    vbox = wx.BoxSizer(wx.VERTICAL)
    vbox.Add(hbox, proportion=0, flag=wx.EXPAND)
    vbox.Add(files, proportion=1,
             flag=wx.EXPAND | wx.LEFT | wx.RIGHT | wx.BOTTOM, border=10)

    bkg.SetSizer(vbox)

    win.Show()

    return True
```

```
    def fetchHandler(self, event):
        """
        Called when the user clicks the 'Fetch' button. Reads the
        query from the text field, and calls the fetch method of the
        server Node. After handling the query, updateList is called.
        If the query is not handled, an error message is printed.
        """
        query = self.input.GetValue()
        try:
            self.server.fetch(query, self.secret)
            self.updateList()

        except Fault, f:
            if f.faultCode != UNHANDLED: raise
            print "Couldn't find the file", query

def main():
    urlfile, directory, url = sys.argv[1:]
    client = Client(url, directory, urlfile)
    client.MainLoop()

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

And that's it. You now have a GUI-enabled peer-to-peer file sharing program, which can be run with this command:

```
$ python guiclient.py urlfile.txt files/ http://localhost:8080
```

Figure 28-2 shows the finished GUI client.

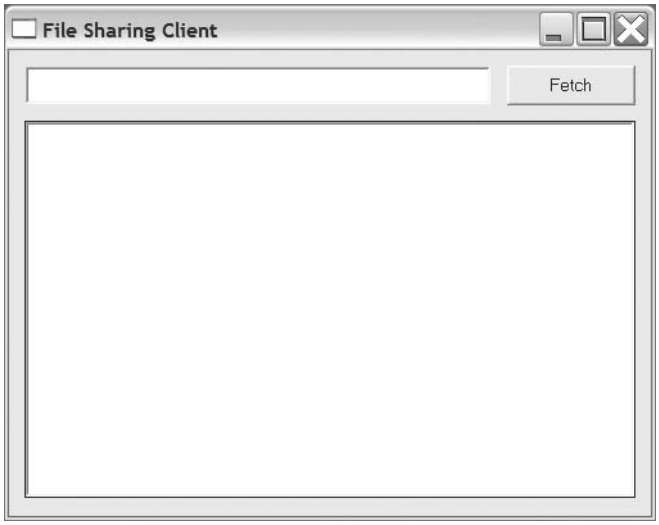

Figure 28-2. *The finished GUI client*

Of course, there are plenty of ways to expand the program. For some ideas, see the next section. Beyond that, just let your imagination go wild.

Further Exploration

Some ideas for extending the file sharing system are given in Chapter 27. Here are some more:

- Add a status bar that displays such messages as "Downloading" or "Couldn't find file `foo.txt`."
- Figure out ways for `Node`s to share their "friends." For example, when one `Node` is introduced to another, each of them could introduce the other to the `Node`s it already knows. Also, before a `Node` shuts down, it might tell all its current neighbors about all the `Node`s it knows.
- Add a list of known `Node`s (URLs) to the GUI. Make it possible to add new URLs and save them in a URL file.

What Now?

You've written a full-fledged GUI-enabled peer-to-peer file sharing system. Although that sounds pretty challenging, it wasn't all that hard, was it? Now it's time to face the last and greatest challenge: writing your own arcade game.