CHAPTER 25

Project 6: Remote Editing with CGI

This chapter's project uses CGI, which is discussed in more detail in Chapter 15. The specific application is *remote editing*—editing a document on another machine via the Web. This can be useful in collaboration systems (groupware), for example, where several people may be working on the same document. It can also be useful for updating your web pages.

What's the Problem?

You have a document stored on one machine and want to be able to edit it from another machine via the Web. This enables you to have a shared document edited by several collaborating authors. You won't need to use FTP or similar file-transfer technologies, and you won't need to worry about synchronizing multiple copies. To edit the file, all you need is a web browser.

Note This sort of remote editing is one of the core mechanisms of *wikis* (see, for example, `http://en.wikipedia.org/wiki/Wiki`).

Specifically, the system should meet the following requirements:

- It should be able to display the document as a normal web page.
- It should be able to display the document in a text area in a web form.
- You should be able to save the text from the form.
- The program should protect the document with a password.
- The program should be easily extensible to support editing more than one document.

As you'll see, all of this is quite easy to do with the standard Python library module `cgi` and some plain Python coding. However, the techniques used in this application can be used for creating web interfaces to all of your Python programs, so it's pretty useful.

Useful Tools

The main tool when writing CGI programs is, as discussed in Chapter 15, the `cgi` module, along with the `cgitb` module for debugging. See Chapter 15 for more information.

Preparations

The steps needed for making your CGI script accessible through the Web are described in detail in Chapter 15 in the section "Dynamic Web Pages with CGI." Just follow those steps, and you should be fine.

First Implementation

The first implementation is based on the basic structure of the greeting script shown in Listing 15-7 (Chapter 15). All that's needed for the first prototype is some file handling.

For the script to be useful, it must store the edited text between invocations. Also, the form should be made a bit bigger than in the greeting script (`simple3.cgi` from Listing 15-7 in Chapter 15), and the text field should be changed into a text area. You should also use the `POST` CGI method instead of the default `GET` method. (Using `POST` is normally the thing to do if you are submitting large amounts of data.)

The general logic of the program is as follows:

1. Get the CGI parameter `text` with the current value of the data file as the default.
2. Save the text to the data file.
3. Print out the form, with the text in the `textarea`.

In order for the script to be allowed to write to your data file, you must first create such a file (for example, `simple_edit.dat`). It can be empty or perhaps contain the initial document (a plain text file, possibly containing some form of markup such as XML or HTML). Then you must set the permissions so that it is universally writable, as described in Chapter 15. The resulting code is shown in Listing 25-1.

Listing 25-1. *A Simple Web Editor (simple_edit.cgi)*

```
#!/usr/bin/env python

import cgi
form = cgi.FieldStorage()


text = form.getvalue('text', open('simple_edit.dat').read())
f = open('simple_edit.dat', 'w')
f.write(text)
f.close()
```

```
print """Content-type: text/html

<html>
  <head>
    <title>A Simple Editor</title>
  </head>
  <body>
    <form action='simple_edit.cgi' method='POST'>
    <textarea rows='10' cols='20' name='text'>%s</textarea><br />
    <input type='submit' />
    </form>
  </body>
</html>
""" % text
```

When accessed through a web server, the CGI script checks for an input value called `text`. If such a value is submitted, the text is written to the file `simple_edit.dat`. The default value is the file's current contents. Finally, a web page (containing the field for editing and submitting the text) is displayed, as shown in Figure 25-1.

Figure 25-1. *The simple_edit.cgi script in action*

Second Implementation

Now that you have the first prototype on the road, what's missing? The system should be able to edit more than one file, and it should use password protection. (Because the document can be viewed by opening it directly in a browser, you won't be paying much attention to the viewing part of the system.)

The main difference from the first prototype is that you'll split it into two separate CGI scripts (one for each "action" your system should be able to perform). The files of the new prototypes are as follows:

`index.html`: A plain web page with a form where you can enter a file name. It also has an Open button, which triggers `edit.cgi`.

`edit.cgi`: A script that displays a given file in a text area. It has a text field for password entry and a Save button, which triggers `save.cgi`.

`save.cgi`: A script that saves the text it receives to a given file and displays a simple message (for example, "The file has been saved"). This script should also take care of the password checking.

Let's tackle these one by one.

Creating the File Name Form

`index.html` is an HTML file that contains the form used to enter a file name:

```
<html>
  <head>
    <title>File Editor</title>
  </head>
  <body>
    <form action='edit.cgi' method='POST'>
      <b>File name:</b><br />
      <input type='text' name='filename' />
      <input type='submit' value='Open' />
  </body>
</html>
```

Notice how the text field is named `filename`. That ensures its contents will be supplied as the CGI parameter `filename` to the `edit.cgi` script (which is the `action` attribute of the `form` tag). If you open this file in a browser, enter a file name in the text field, and click Open, the `edit.cgi` script will be run.

Writing the Editor Script

The page displayed by the `edit.cgi` script should include a text area containing the current text of the file you're editing, and a text field for entering a password. The only input needed is the file name, which the script receives from the form in `index.html`. Note, however, that it is possible to open the `edit.cgi` script directly, without submitting the form in `index.html`. In that case, you have no guarantee that the `filename` field of `cgi.FieldStorage` is set. So you need to add a check to ensure that there *is* a file name. If there is, the file will be opened from a directory that contains the files that may be edited. Let's call the directory `data`. (You will, of course, have to create this directory.)

■ **Caution** Note that by supplying a file name that contains path elements such as `..` (two dots), it may be possible to access files outside this directory. To make sure that the files accessed are within the given directory, you should perform some extra checking, such as listing all the files in the directory (using the `glob` module, for example) and checking that the supplied file name is one of the candidate files (making sure you use full, absolute path names all around). See the section "Validating File Names" in Chapter 27 for another approach.

The code, then, becomes something like Listing 25-2.

Listing 25-2. *The Editor Script (edit.cgi)*

```
#!/usr/bin/env python

print 'Content-type: text/html\n'

from os.path import join, abspath
import cgi, sys

BASE_DIR = abspath('data')

form = cgi.FieldStorage()
filename = form.getvalue('filename')
if not filename:
    print 'Please enter a file name'
    sys.exit()
text = open(join(BASE_DIR, filename)).read()

print """
<html>
  <head>
    <title>Editing...</title>
  </head>
  <body>
    <form action='save.cgi' method='POST'>
      <b>File:</b> %s<br />
      <input type='hidden' value='%s' name='filename' />
      <b>Password:</b><br />
      <input name='password' type='password' /><br />
      <b>Text:</b><br />
      <textarea name='text' cols='40' rows='20'>%s</textarea><br />
      <input type='submit' value='Save' />
    </form>
  </body>
```

```
</html>
""" % (filename, filename, text)
```

Note that the `abspath` function has been used to get the absolute path of the `data` directory. Also note that the file name has been stored in a `hidden` form element so that it will be relayed to the next script (`save.cgi`) without giving the user an opportunity to change it. (You have no guarantees of that, of course, because users may write their own forms, put them on another machine, and have those forms call your CGI scripts with custom values.)

For password handling, the sample code uses an input element of type `password` rather than `text`, which means that the characters entered will be displayed as asterisks.

■ **Note** This script is based on the assumption that the file name given refers to an existing file. Feel free to extend it so that it can handle other cases as well.

Writing the Save Script

The script that performs the saving is the last component of this simple system. It receives a file name, a password, and some text. It checks that the password is correct, and if it is, the program stores the text in the file with the given file name. (The file should have its permissions set properly; see the discussion of setting file permissions in Chapter 15.)

Just for fun, you'll use the `sha` module in the password handling. The Secure Hash Algorithm (SHA) is a way of extracting an essentially meaningless string of seemingly random data (a *digest*) from an input string. The idea behind the algorithm is that it is almost impossible to construct a string that has a given digest, so if you know the digest of a password (for example), there is no (easy) way you can reconstruct the password or invent one that will reproduce the digest. This means that you can safely compare the digest of a supplied password with a stored digest (of the correct password) instead of comparing the passwords themselves. By using this approach, you don't need to store the password itself in the source code, and someone reading the code would be none the wiser about what the password actually *was*.

■ **Caution** As I said, this "security" feature is mainly for fun. Unless you are using a secure connection with SSL or some similar technology (which is beyond the scope of this project), it is still possible to pick up the password being submitted over the network.

Here is an example of how you can use sha:

```
>>> from sha import sha
>>> sha('foobar').hexdigest()
'8843d7f92416211de9ebb963ff4ce28125932878'
>>> sha('foobaz').hexdigest()
'21eb6533733a5e4763acacd1d45a60c2e0e404e1'
```

As you can see, a small change in the password gives you a completely different digest. You can see the code for `save.cgi` in Listing 25-3.

Listing 25-3. *The Saving Script (save.cgi)*

```
#!/usr/bin/env python

print 'Content-type: text/html\n'

from os.path import join, abspath
import cgi, sha, sys

BASE_DIR = abspath('data')

form = cgi.FieldStorage()

text = form.getvalue('text')
filename = form.getvalue('filename')
password = form.getvalue('password')

if not (filename and text and password):
    print 'Invalid parameters.'
    sys.exit()

if sha.sha(password).hexdigest() != '8843d7f92416211de9ebb963ff4ce28125932878':
    print 'Invalid password'
    sys.exit()

f = open(join(BASE_DIR,filename), 'w')
f.write(text)
f.close()

print 'The file has been saved.'
```

Running the Editor

Follow these steps to use the editor:

1. Open the page `index.html` in a web browser. Be sure to open it through a web server (by using a URL of the form `http://www.someserver.com/index.html`) and not as a local file. The result is shown in Figure 25-2.
2. Enter a file name of a file that your CGI editor is permitted to modify, and then click Open. Your browser should then contain the output of the `edit.cgi` script, as shown in Figure 25-3.
3. Edit the file to taste, enter the password (one you've set yourself, or the one used in the example, which is `foobar`), and click Save. Your browser should then contain the output of the `save.cgi` script, which is simply the message "The file has been saved."
4. If you want to verify that the file has been modified, repeat the process of opening the file (steps 1 and 2).

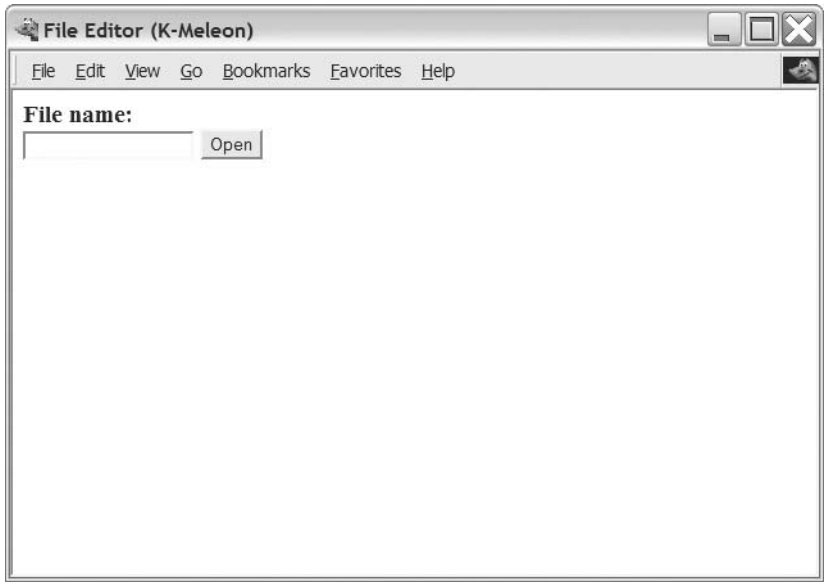

Figure 25-2. *The opening page of the CGI editor*

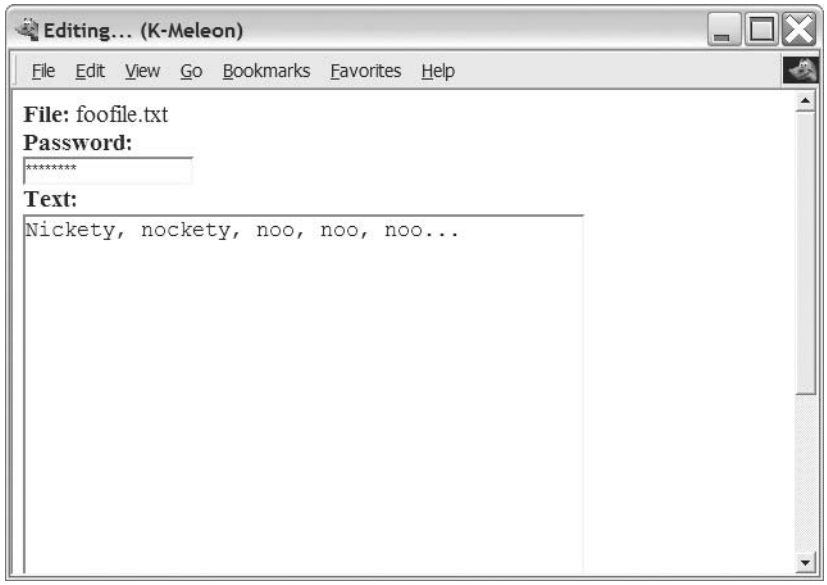

Figure 25-3. *The editing page of the CGI editor*

Further Exploration

With the techniques shown in this project, you can develop all kinds of web systems. Some possible additions to the existing system are as follows:

- Add version control. Save old copies of the edited file so you can "undo" your changes.
- Add support for user names so you know who changed what.
- Add file locking (for example, with the `fcntl` module) so two users can't edit the file at the same time.
- Add a `view.cgi` script that automatically adds markup to the files (like the one in Chapter 20).
- Make the scripts more robust by checking their input more thoroughly and adding more user-friendly error messages.
- Avoid printing a confirmation message like "The file has been saved." You can either add some more useful output or redirect the user to another page/script. Redirection can be done with the `Location` header, which works like `Content-type`. Just add `Location:` followed by a space and a URL to the header section of the output (*before* the first empty line).

In addition to expanding the capabilities of this CGI system, you might want to check out some more complex web environments for Python (as discussed in Chapter 15).

What Now?

Now you've tried your hand at writing CGI scripts. In the next project, you expand on that by using a SQL database for storage. With that powerful combination, you'll implement a fully functional web-based bulletin board.