The final step in building the MP3 streaming application is to provide a Web interface that allows a user to find the songs they want to listen to and add them to a playlist that the Shoutcast server will draw upon when the user’s MP3 client requests the stream URL. For this component of the application, you’ll pull together several bits of code from the previous few chapters: the MP3 database, the define-url-function macro from Chapter 26, and, of course, the Shoutcast server itself.

**Playlists**

The basic idea behind the interface will be that each MP3 client that connects to the Shoutcast server gets its own playlist, which serves as the source of songs for the Shoutcast server. A playlist will also provide facilities beyond those needed by the Shoutcast server: through the Web interface the user will be able to add songs to the playlist, delete songs already in the playlist, and reorder the playlist by sorting and shuffling.

You can define a class to represent playlists like this:

```lisp
(defclass playlist ()
  ((id           :accessor id           :initarg :id)
   (songs-table  :accessor songs-table  :initform (make-playlist-table))
   (current-song :accessor current-song :initform *empty-playlist-song*)
   (current-idx  :accessor current-idx  :initform 0)
   (ordering     :accessor ordering     :initform :album)
   (shuffle      :accessor shuffle      :initform :none)
   (repeat       :accessor repeat       :initform :none)
   (user-agent   :accessor user-agent   :initform "Unknown")
   (lock         :reader   lock         :initform (make-process-lock)))))
```

The id of a playlist is the key you extract from the request object passed to find-song-source when looking up a playlist. You don’t actually need to store it in the playlist object, but it makes debugging a bit easier if you can find out from an arbitrary playlist object what its id is.

The heart of the playlist is the songs-table slot, which will hold a table object. The schema for this table will be the same as for the main MP3 database. The function make-playlist-table, which you use to initialize songs-table, is simply this:

```lisp
(defun make-playlist-table ()
  (make-instance 'table :schema *mp3-schema*))
```
THE PACKAGE

You can define the package for the code in this chapter with the following DEFPACKAGE:

```
(defpackage :com.gigamonkeys.mp3-browser
 (:use :common-lisp
 :net.aserve
 :com.gigamonkeys.html
 :com.gigamonkeys.shoutcast
 :com.gigamonkeys.url-function
 :com.gigamonkeys.mp3-database
 :com.gigamonkeys.id3v2)
 (:import-from :acl-socket
 :ipaddr-to-dotted
 :remote-host)
 (:import-from :multiprocessing
 :make-process-lock
 :with-process-lock)
 (:export :start-mp3-browser))
```

Because this is a high-level application, it uses a lot of lower-level packages. It also imports three symbols from the ACL-SOCKET package and two more from MULTIPROCESSING since it just needs those five and not the other 139 symbols those two packages export.

By storing the list of songs as a table, you can use the database functions from Chapter 27 to manipulate the playlist: you can add to the playlist with `insert-row`, delete songs with `delete-rows`, and reorder the playlist with `sort-rows` and `shuffle-table`.

The current-song and current-idx slots keep track of which song is playing: current-song is an actual song object, while current-idx is the index into the songs-table of the row representing the current song. You’ll see in the section “Manipulating the Playlist” how to make sure current-song is updated whenever current-idx changes.

The ordering and shuffle slots hold information about how the songs in songs-table are to be ordered. The ordering slot holds a keyword that tells how the songs-table should be sorted when it’s not shuffled. The legal values are :genre, :artist, :album, and :song. The shuffle slot holds one of the keywords :none, :song, or :album, which specifies how songs-table should be shuffled, if at all.

The repeat slot also holds a keyword, one of :none, :song, or :all, which specifies the repeat mode for the playlist. If repeat is :none, after the last song in the songs-table has been played, the current-song goes back to a default MP3. When :repeat is :song, the playlist keeps returning the same current-song forever. And if it’s :all, after the last song, current-song goes back to the first song.