```
            // set lease renewal in place
            leaseManager.renewUntil(reg.getLease(), Lease.FOREVER, this);
            // set the serviceID if necessary
            if (serviceID == null) {
                System.out.println("Getting service ID from lookup service");
                serviceID = reg.getServiceID();
                // try to save the service ID in a file
                DataOutputStream dout = null;
                try {
                    dout = new DataOutputStream(new FileOutputStream(service-
IdFile));
                    serviceID.writeBytes(dout);
                    dout.flush();
                    dout.close();
                    System.out.println("Service id saved in " +  serviceIdFile);
                } catch(Exception e) {
                    // ignore
                }
            }
        }
    }
    public void discarded(DiscoveryEvent evt) {
    }
    public void notify(LeaseRenewalEvent evt) {
        System.out.println("Lease expired " + evt.toString());
    }

} // FileClassifierServerIDConfig
```

This program could be run as follows:

```
java FileClassifierServerIDConfig config/serviceid.config
```

Specifying the Codebase

A Jini service needs to specify the `java.rmi.server.codebase` property so that clients can pick up class definitions. In previous chapters where the command line to start a service has been shown, this has always been done by specifying a property at the command line:

```
java -Djava.rmi.server.codebase=http://... ...
```

The Java runtime handles parsing the command line, extracting the property and its value, and using these to set the property value.

Properties can also be set by using the configuration mechanism. While this approach is more cumbersome than using a command-line parameter, it ensures that all runtime options are stored and handled in the same way. A server can pick up the codebase as follows:

```
package config;
import java.rmi.RMISecurityManager;
```