```
%%
%% A *very* simple "database"
%% In fact a list of tuples, of the form {key, data}
%%
%%
%% Deryk Barker, May 26, 2006
%%

-module (db).
-export ([new/0, insert/2, lookup/2, delete/2, first/1, rest/1]).

new () ->
    [].

insert ({Key, Data}, []) ->
    [{Key, Data}];
insert ({Key, Datanew}, [{Key, _}|T]) ->        %replace old value
    [{Key, Datanew} | T];
insert (NewData, [H|T]) ->
    [H|insert (NewData, T)].

lookup (_, []) ->
    notfound;
lookup (Key, [{Key, Data}|_]) ->
    {ok, Data};
lookup (Key, [_|T]) ->
    lookup (Key, T).

delete (_, []) ->
    notfound;
delete (Key, [{Key, _}|T]) ->
    T;
delete (Key, [H|T]) ->
    [H|delete (Key, T)].

first ([]) ->
    [];
first ([H|_]) ->
    H.

rest ([]) ->
    error;
rest ([_|T]) ->
    T.
```

```
-module (dbclient).
-export ([client/0, client/1]).

client () ->
    runclient (node()).                         %% talk to our own node

client(Node) ->                                 % is remote up?
    monitor_node (Node, true),
    receive
        {nodedown, Node} ->                     %they're not there at all
            io:format ("Remote node ~w is down~n", [Node])
    after 0 ->                                  %they are there
            runclient (Node),                   %do the business
            monitor_node (Node, false),         %stop checking
            ok
    end.

runclient (Node) ->
    Input = strip_nl (io:get_line('dbclient> ')),
    Function = list_to_atom(string:sub_word(Input, 1)), %must be there
    Key = list_to_atom(string:sub_word(Input, 2)),      %might
    Value = list_to_atom(string:sub_word(Input, 3)),    %might
    case {Function, Key, Value} of
        {quit, _, _} ->                                 %we're out of here
            ok;
        {finished, _, _} ->
            {dbase, Node} ! {self (), dbase, finished}, %there's no reply
            ok;
        {list, _, _} ->
            {dbase, Node} ! {self (), dbase, list},
            get_response (1000),
            runclient (Node);
        {reload, _, _} ->
            {dbase, Node} ! {self (), dbase, reload},
            runclient (Node);
        {delete, [], _} ->
            io:format ("Must provide a key to delete~n"),
            runclient (Node);
        {delete, Key, __} ->
            {dbase, Node} ! {self(), dbase, {delete, Key}},
            get_response (0),
            runclient (Node);
        {insert, Key, Value} ->
            {dbase, Node} ! {self(), dbase, {insert, {Key, Value}}},
            get_response (0),
            runclient (Node);
        {lookup, Key, _} ->
            {dbase, Node} ! {self(), dbase, {lookup, Key}},
            get_response (0),
            runclient (Node);
        {Any, _, _} ->
            io:format ("Unknown command ~w~n", [Any]),
            runclient (Node)
        end.

%%
%% Get and process response from server
%%

get_response (Wait) ->
    receive
        {dbase, {lookup, {ok, Value}}} ->
            io:format ("~w~n", [Value]);
        {dbase, {lookup, notfound}} ->
            io:format ("Key not found in database~n", []);
        {dbase, {list, {Key, Value}}} ->        %db listing - more (possibly) to
 come
            io:format ("~w: ~w~n", [Key, Value]),
            get_response (0);
```



```
        Any ->
            Any
    after Wait ->
            ok
    end.

%
% Function which we can use to strip the newline from an input string
%

strip_nl ([]) ->
    [];
strip_nl ([_|T]) when T == [] ->
    [];
strip_nl ([H|T]) ->
    [H|strip_nl(T)].
```

```
-module (dbserver).
-export ([start/0, server/1]).

%%
%% The server - Parameter is the database
%%

server(Database) ->
    receive
        {_, dbase, finished} ->                 % Time to shut down
            io:format("dbase server: finished~n", []),
            ok;

        {Client_PID, dbase, reload} ->          % Reload server - fresh copy
            io:format ("dbase server: reloading~n", []),
            Client_PID ! reloaded,              % Tell client
            dbserver:server (Database);                 % Reload

        {Client_PID, dbase, {insert, Data}} ->          % Insert new key/data
            NewDatabase = db:insert (Data, Database),
            Client_PID ! {dbase, {inserted, Data}},     % Respond
            io:format("dbase server version 6 inserted  ~w from ~w~n", [Data, no
de (Client_PID)]),
            server(NewDatabase);                                % Recurse with u
pdated DB

        {Client_PID, dbase, {delete, Key}} ->           % Insert new key/data
            NewDatabase = db:delete (Key, Database),
            Client_PID ! {dbase, {deleted, Key}},       % Respond
            io:format("dbase server version 6 deleted  ~w from ~w~n", [Key, node
 (Client_PID)]),
            server(NewDatabase);                        % Recurse with updated D
B

        {Client_PID, dbase, {lookup, Key}} ->           % Insert new key/data
            Value = db:lookup (Key, Database),
            Client_PID ! {dbase, {lookup, Value}},      % Respond
            io:format("dbase server version 6 looked up  ~w from ~w~n", [Key, no
de (Client_PID)]),
            server(Database);                           % Recurse with same DB

        {Client_PID, dbase, list} ->            %list db contents
            io:format("dbase server version 6 list dbase for ~w~n", [node (Clien
t_PID)]),
            list (Client_PID, Database),
            server (Database);

        {Client_PID, dbase, Any} ->             %don't recognise this
            Client_PID ! {dbase, {error, Any}}, % Respond
            io:format("dbase server version 6 unknown request  ~w from ~w~n", [A
ny, node (Client_PID)]),
            server(Database)                            % Recurse with same DB
    end.

start() ->                                      % Start our server and register
its name
    register(dbase, spawn(dbserver, server, [db:new ()])).

list (_, []) ->
    ok;
list (Client_PID, Database) ->
    Client_PID ! {dbase, {list, db:first (Database)}},
    list (Client_PID, db:rest (Database)).
```