```
1> lucene_server:start().
ok
```

```
2> User1 =
        [{name, "Fernando"}
        ,{nick, "elbrujohalcon"}].
[{name, "Fernando"},{nick, "elbrujohalcon"}]

3> User2 =
        [{name, "Ariel Ortega"}
        ,{nick, "burrito"}].
[{name, "Ariel Ortega"},{nick, "burrito"}]

4> lucene:add([User1,User2]).
ok
```

```
5> lucene:match("nick: b*", 10).
{[[{name,"Fernando Benavides"},
   {nick,"elbrujohalcon"},
   {'score',1.0}]],
 [{total_hits,1},
  {first_hit,1},
  {query_time,783},
  {search_time,457}]}
```

```
6> lucene:add([[{code, X}] || X <- lists:seq(1,10)]).
ok

7> {_, M} = lucene:match("code:[0 TO 10]", 5).
{[[{code,1},{'score',1.0}],
  [{code,2},{'score',1.0}],
  [{code,3},{'score',1.0}],
  [{code,4},{'score',1.0}],
  [{code,5},{'score',1.0}]],
 [{total_hits,10},
  {first_hit,1},
  {query_time,14335},
  {search_time,3811},
  {next_page,<<172,237,0,5,115,114,0,36,99,111,109,46,
               116,105,103,101,114,116,101,120,...>>}]}

8> lucene:continue(proplists:get_value(next_page, M), 5).
{[[{code,6},{'score',1.0}],
  [{code,7},{'score',1.0}],
  [{code,8},{'score',1.0}],
  [{code,9},{'score',1.0}],
  [{code,10},{'score',1.0}]],
 [{total_hits,10},
  {first_hit,6},
  {query_time,2368},
  {search_time,1731}]}
```

```
{pre_hooks, [{compile, "mkdir -p bin"},
             {compile, "./copy-jinterface.sh"}]}.

{post_hooks,
  [{clean,
    "rm -rf bin priv/lucene_server.jar priv/OtpErlang.jar"},
   {compile,
    "javac -g -deprecation -sourcepath java_src
           -classpath ./bin:./priv/* -d bin
           java_src/*/*/*/*.java"},
   {compile, "jar cf priv/lucene-server.jar -C bin ."}]}.
```

```
init([]) ->
...
  Port =
    erlang:open_port(
      {spawn_executable, Java},
      [{line,1000}, stderr_to_stdout,
       {args, JavaArgs ++
                    ["-classpath", Classpath,
                     "com.tigertext.lucene.LuceneNode",
                     ThisNode, JavaNode, erlang:get_cookie(),
                     integer_to_list(AllowedThreads)]}]),
  wait_for_ready(
    #state{java_port = Port, java_node = JavaNode})
end.
...
wait_for_ready(State = #state{java_port = Port}) ->
  receive
    {Port, {data, {eol, "READY"}}} ->
      true = link(process()),
      true = erlang:monitor_node(State#state.java_node, true),
      {ok, State};
    Info ->
      ...
  end.
```

```
public static void main(String[] args) {
  String peerName = args.length >= 1 ? args[0]
      : "lucene_server@localhost";
  String nodeName = args.length >= 2 ? args[1]
      : "lucene_server_java@localhost";

  try {
    NODE = args.length >= 3 ? new OtpNode(nodeName, args[2])
        : new OtpNode(nodeName);
    PEER = peerName;

    final OtpGenServer server = new LuceneServer(NODE);

    server.start();

    System.out.println("READY");

  } catch (IOException e1) {
    jlog.severe("Couldn't create node: " + e1);
    e1.printStackTrace();
    System.exit(1);
  }
}
```

```
do_call(Process, Label, Request, Timeout) ->
    try erlang:monitor(process, Process) of
        Mref ->
...
    catch
        error:_ ->
            %% Node (C/Java?) is not supporting the monitor.
            %% The other possible case -- this node is not
            %% distributed -- should have been handled earlier.
            %% Do the best possible with monitor_node/2.
            %% This code may hang indefinitely if the Process
            %% does not exist. It is only used for featureweak
            %% remote nodes.
            Node = get_node(Process),
            monitor_node(Node, true),
...
    end.
```

```
public abstract class OtpGenServer extends OtpSysProcess {
  protected OtpGenServer(OtpNode host) {
    super(host);
  }

  protected OtpGenServer(OtpNode host, String name) {
    super(host, name);
  }
...
  protected abstract OtpErlangObject handleCall(
      OtpErlangObject cmd, OtpErlangTuple from)
      throws OtpStopException, OtpContinueException,
      OtpErlangException;

  protected abstract void handleCast(OtpErlangObject cmd)
      throws OtpStopException, OtpErlangException;

  protected abstract void handleInfo(OtpErlangObject cmd)
      throws OtpStopException;

  protected abstract void terminate(OtpErlangException oee);

}
```

```
protected OtpErlangObject handleCall(OtpErlangObject cmd,
    OtpErlangTuple from)
    throws OtpStopException, OtpContinueException,
    OtpErlangException {

  OtpErlangTuple cmdTuple = (OtpErlangTuple) cmd;
  OtpErlangAtom cmdName = (OtpErlangAtom) cmdTuple.elementAt(0);

  if (cmdName.atomValue().equals("pid")) {
    return super.getSelf();
  } else if (cmdName.atomValue().equals("match")) {
    String queryString =
      ((OtpErlangString) cmdTuple.elementAt(1)).stringValue();
    int pageSize =
      ((OtpErlangLong) cmdTuple.elementAt(2)).intValue();

    runMatch(queryString, pageSize, from);

    throw new OtpContinueException();

  } else
...
}
```

```
9> lucene:add(
     [[{index, I},{location, lucene_utils:geo(I/2,I+1.5)}]
      || I <- lists:seq(-10,10)]).
ok

10> lucene:match("location.near:0.0,1.0,100", 3).
{[[{index,0},
   {location,{geo,-8.381903171539307e-8,1.5000000782310963}},
   {'score',-17.292743682861328}],
  [{index,-1},
   {location,{geo,-0.5000000540167093,0.500000137835741}},
   {'score',-24.455478668212891}]],
 [{total_hits,2},
  {first_hit,1},
  {query_time,1984},
  {search_time,1021}]}
```

```
11> lucene:match(
      "index.erlang:\"lists:map:[fun(X) -> X*1.0 end]\"", 3).
{[[{index,10},
   {location,{geo,4.999999953433871,11.500000152736902}},
   {'score',5.0}],
  [{index,9},
   {location,{geo,4.499999983236194,10.499999987706542}},
   {'score',4.499999523162842}],
  [{index,8},
   {location,{geo,4.000000013038516,9.499999936670065}},
   {'score',3.999999761581421}]],
 [{total_hits,21},
  {first_hit,1},
  {query_time,2256},
  {search_time,1240},
  {next_page,<<172,237,0,5,115,114,0,36,99,111,109,46,
               116,105,103,101,114,116,101,120,...>>}]}
```

```
public LuceneServer(OtpNode host, int allowedThreads)
    throws CorruptIndexException, LockObtainFailedException,
           IOException {
  super(host, "lucene_server");
...
  Extensions ext = new Extensions('.');
  ext.add("near", new NearParserExtension());
  ext.add("erlang", new ErlangParserExtension(this.translator));
  this.extensions = ext;
...
}

private QueryParser queryParser() {
  return new LuceneQueryParser(Version.LUCENE_36, this.analyzer,
      this.translator, this.extensions);
}
```

```
public Query parse(ExtensionQuery extQuery)
  throws ParseException {
  String key = extQuery.getField();
  String[] modFun = extQuery.getRawQueryString().split(":");
  if (modFun.length < 2) {
    throw new ParseException(
        "erlang queries expect values in <mod>:<fun> or
         <mod>:<fun>:<args> format");
  } else {
    String mod = modFun[0], fun = modFun[1], args;
    if (modFun.length == 2) {
      args = "[]";
    } else if (modFun.length == 3) {
      args = modFun[2];
    } else {
...
    }
    ErlangFilter filter = new ErlangFilter(mod, fun, args, key,
        this.translator.getFieldType(key));
    ValueSource valSrc = new ErlangValueSource(filter);
    return new CustomScoreQuery(new ConstantScoreQuery(filter),
        new ValueSourceQuery(valSrc));
  }
}
```

```
public DocIdSet getDocIdSet(IndexReader reader)
  throws IOException {
  final int docBase = this.nextDocBase;
  this.nextDocBase += reader.maxDoc();
  final OtpErlangObject[] docValues;
...
  final FixedBitSet bits = new FixedBitSet(reader.maxDoc());
  OtpErlangTuple call =
    new OtpErlangTuple(new OtpErlangObject[] {
      this.mod,this.fun,this.arg, new OtpErlangList(docValues)});
...
  OtpErlangObject response = OtpGenServer.call(LuceneNode.NODE,
      "lucene", LuceneNode.PEER, call);
...
  OtpErlangList results = (OtpErlangList) response;
  for (int docid = 0; docid < docValues.length; docid++) {
    OtpErlangObject result = results.elementAt(docid);
    if (result instanceof OtpErlangDouble) {
      scores.put(docid + docBase,
          ((OtpErlangDouble) result).doubleValue());
      bits.set(docid);
    } else {
      bits.clear(docid);
    }
...
}
```

```
Reply =
  try
    {ok, Scanned, _} = erl_scan:string(Args++"."),
    {ok, Parsed} = erl_parse:parse_exprs(Scanned),
    {value, Arguments, _} = erl_eval:exprs(Parsed, []),
    erlang:apply(
      Mod,Fun,
      Arguments ++ [[parse_value(Value) || Value <- Values]])
  catch
    _:Error ->
      {error, Error}
  end,
gen_server:reply(From, Reply)
```