

```
#SysVisual Properties
#Wed Apr 04 18:14:47 PDT 2001
SysVisual.startNames=ejava^
SysVisual.browserPath=
SysVisual.urlPrefix=jdbc\:cloudscape\:rmi\://localhost\:1099/
SysVisual.caseInsensitiveDDL=true
SysVisual.host=
SysVisual.driver=COM.cloudscape.core.RmiJdbcDriver
SysVisual.port=
SysVisual.attributes=
SysVisual.autoSave=true
SysVisual.queryHistoryLimit=100
```

```
package ejava.ejb.uid;

import java.io.Serializable;

/**
 * This class encapsulates the UID value. The current implementation is
 * quite simple. More enterprise solutions would be slightly more complex
 * in that they would have to account for multiple id generators.
 */
public final class UID implements Serializable {
    public String value_;

    public UID(String value)                { value_ = value; }

    public String toString()                { return value_; }
    public int hashCode()                   { return value_.hashCode(); }
    public boolean equals(Object rhs)       {
        try {
            return ((UID)rhs).value_.equals(value_);
        }
        catch (Exception ex) {
            return false;
        }
    }
}
```

```
package ejava.ejb.uid;

import javax.ejb.EJBObject;

/**
 * This interface aggregates the business interface of the UIDGeneratorBI
 * with the remote EJB properties of EJBObject.
 */
public interface UIDGenerator extends UIDGeneratorBI, EJBObject {}
```

```
package ejava.ejb.uid;
```

```
import java.rmi.RemoteException;
```

```
/**  
    This interface defines the sole interface method for the UIDGenerator  
    object.  
*/  
public interface UIDGeneratorBI {  
    public UID createUID() throws RemoteException;  
}
```

```
package ejava.ejb.uid;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.EJBException;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.rmi.RemoteException;

/**
 * This bean class manages the generation of a unique id for client
 * accessors. It uses BMP to manage interaction with the tracked
 * sequence number in the database.
 */
public class UIDGeneratorBean implements UIDGeneratorBI, EntityBean {
    private DataSource dataSource_;
    private String tableName_;
    private int sequenceNum_;

    /**
     * There is only one UIDGenerator. It will logically be assigned a
     * primary key value of 0.
     */
    public Integer ejbFind() {
        return new Integer(0);
    }

    public Integer ejbFindByPrimaryKey(Integer dummy) {
        return new Integer(0);
    }

    /**
     * Creates a unique UID.
     */
    public ejava.ejb.uid.UID createUID() throws RemoteException {
        return new UID(String.valueOf(++sequenceNum_));
    }

    /**
     * Loads the previous UID value handed out.
     */
    public void ejbLoad() throws RemoteException {
        Connection connection = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            connection = dataSource_.getConnection();
            statement = connection.createStatement();
            rs = statement.executeQuery(
                "select max(seqNum) from " + tableName_);
            rs.next();
            sequenceNum_ = rs.getInt(1);
        }
        catch (SQLException ex) {
            throw new EJBException("error loading sequence value:"+ex);
        }
    }
}
```

```
    }
    finally {
        try {
            if (rs != null) rs.close();
            if (statement != null) statement.close();
            if (connection != null) connection.close();
        }
        catch (SQLException ex) {
            throw new EJBException("error releasing resources:"+ex);
        }
    }
}

/**
 * Stores the last value handed out.
 */
public void ejbStore() throws RemoteException {
    Connection connection = null;
    Statement statement = null;
    ResultSet rs = null;
    try {
        connection = dataSource_.getConnection();
        statement = connection.createStatement();
        statement.executeUpdate(
            "update " + tableName_ + " set seqNum = " + sequenceNum_);
    }
    catch (SQLException ex) {
        throw new EJBException("error saving sequence value:"+ex);
    }
    finally {
        try {
            if (rs != null) rs.close();
            if (statement != null) statement.close();
            if (connection != null) connection.close();
        }
        catch (SQLException ex) {
            throw new EJBException("error releasing resources:"+ex);
        }
    }
}

public void ejbPassivate() {}
public void ejbActivate() {}

/**
 * Clients don't create or remove UIDGenerators. They just find the
 * one and only instance and use it.
 */
public void ejbRemove() throws RemoteException {
    throw new EJBException("cannot remove UIDGenerators");
}

private EntityContext entityCtx_;
public void setEntityContext(EntityContext entityCtx) {
    entityCtx_ = entityCtx;

    try {
        Context ctx = new InitialContext();
        dataSource_ = (DataSource)ctx.lookup("java:comp/env/jdbc/uidDB");
        tableName_ = (String)ctx.lookup("java:comp/env/jdbc/tableName");
    }
}
```

```
        catch (NamingException ex) {  
            throw new EJBException("error configuring UIDGeneratorBean:"+ex);  
        }  
    }  
    public void unsetEntityContext() { entityCtx_ = null; }  
}
```

```
package ejava.ejb.uid;

import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;

/**
 * This interface defines the find method for locating the singleton
 * UIDGenerator.
 */
public interface UIDGeneratorHome extends EJBHome {
    UIDGenerator findByPrimaryKey(Integer dummy)
        throws FinderException, RemoteException;

    UIDGenerator find() throws FinderException, RemoteException;
}
```

```
package ejava.ejb.uid.client;

import ejava.ejb.uid.UID;
import ejava.ejb.uid.UIDGenerator;
import ejava.ejb.uid.UIDGeneratorHome;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.rmi.RemoteException;
import java.util.Properties;

public class Client {
    private static String user_;
    private static String password_;

    private static UIDGeneratorHome getHome() throws NamingException {
        Context ctx = getInitialContext();
        Object object = ctx.lookup("UIDGeneratorHome");
        UIDGeneratorHome uHome = (UIDGeneratorHome)
            PortableRemoteObject.narrow(object, UIDGeneratorHome.class);

        return uHome;
    }

    private static Context getInitialContext() throws NamingException {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
            "weblogic.jndi.WLInitialContextFactory");
        props.put(Context.PROVIDER_URL,
            "t3://localhost:7001");
        if (user_ != null) {
            props.put(Context.SECURITY_PRINCIPAL, user_);
        }
        if (password_ != null) {
            props.put(Context.SECURITY_CREDENTIALS, password_);
        }

        return new InitialContext(props);
    }

    public static void main(String args[]) {
        int count=1;

        for (int i=0; i<args.length; i++) {
            if (args[i].equals("-u")) user_ = args[++i];
            else if (args[i].equals("-p")) password_ = args[++i];
            else if (args[i].equals("-c")) count = Integer.parseInt(args[++i]);
        }

        System.out.println("count="+count);

        try {
            UIDGeneratorHome uidHome = getHome();
            UIDGenerator uidGen = uidHome.find();
            for(int i=0; i<count; i++) {
                System.out.println(uidGen.createUID().toString());
            }
        }
        catch (Exception ex) {
```

```
ex.printStackTrace();
```

```
}
```

```
}
```

```
}
```

```
include $(MAKEROOT)/java.rules

PACKAGE_BASE      = ejava.ejb
PACKAGE_SPEC      = uid.client
DEPENDENCIES      =
DEPENDENTS        =
LOCAL_JAVAFLAGS   =
LOCAL_CLASSPATH   = $(EJBCLASS_DIR)
JAVA_CLASSES      = \
    Client

EJB               = UID
EJBJAR            =
EJB_CLASSES       =
METAINF_FILES     =

PROPERTY_FILES    =

SCRIPTS           =

include $(MAKEROOT)/java.cmds
```

```
include $(MAKEROOT)/java.rules

PACKAGE_BASE      = ejava.ejb
PACKAGE_SPEC      = uid
DEPENDENCIES      =
DEPENDENTS        = wls client
LOCAL_JAVAFLAGS   =
LOCAL_CLASSPATH   = $(EJBCLASS_DIR)
JAVA_CLASSES      = \
    UID \
    UIDGeneratorBI \
    UIDGenerator \
    UIDGeneratorHome

EJB               = UID
EJBJAR            =
EJB_CLASSES       = \
    UID \
    UIDGeneratorBI \
    UIDGenerator \
    UIDGeneratorHome \
    UIDGeneratorBean

METAINF_FILES     = ejb-jar

PROPERTY_FILES    =

SCRIPTS           =

include $(MAKEROOT)/java.cmds
```

```
#!/usr/bin/perl
```

```
$WL_HOME="/apps/weblogic";
#$CLOUD_HOME="$WL_HOME/eval/cloudscape";
#$CLOUD_LIB="$CLOUD_HOME/lib";
$HSQL_HOME="/apps/hypersonicsql";
$HSQL_LIB="$HSQL_HOME/lib";

$classpath="$WL_HOME/classes";
#$classpath="$classpath:$CLOUD_LIB/client.jar";
#$classpath="$classpath:$CLOUD_LIB/tools.jar";
#$classpath="$classpath:$CLOUD_LIB/cloudscape.jar";
$classpath="$classpath:$HSQL_LIB/hsqldb.jar";

$javaProps="-classpath $classpath";
#$javaProps="$javaProps -Dcloudscape.system.home=$CLOUD_HOME/data";

$url="\jdbc:cloudscape:wlsdb;create=true;upgrade=true\";
$driver="COM.cloudscape.core.JDBCdriver";
$sql="personnel_cloudscape.ddl";
$url="jdbc:HypersonicSQL:/usr/local/data/hsqldata/wlsdb";
$driver="org.hsqldb.jdbcDriver";
$user="sa";
$password="";
$sql="uid_hsqldb.ddl";

$options="-verbose";

$javaClass="utils.Schema";
$javaArgs="$url $driver -u $user -p $password $options $sql";

$javaCommand="java $javaProps $javaClass $javaArgs";

print("$javaCommand\n");
system($javaCommand);
```

```
include $(MAKEROOT)/java.rules

PACKAGE_BASE      = ejava.ejb
PACKAGE_SPEC      = uid.wls
DEPENDENCIES      =
DEPENDENTS        =
LOCAL_JAVAFLAGS   =
LOCAL_CLASSPATH   = $(EJBCLASS_DIR)
JAVA_CLASSES      =

EJB               = UID
EJBJAR            = UID
EJB_CLASSES       =
METAINF_FILES     = weblogic-ejb-jar

PROPERTY_FILES    =

SCRIPTS           =

include $(MAKEROOT)/java.cmds
```

```
drop table UID
;
create table UID (
    seqNum int not null
);
insert into UID ( seqNum ) values ( 0 );
```

```
package ejava.ejb.personnel;
```

```
import javax.ejb.EJBObject;
```

```
/**
```

```
    This interface brings combines the remote EJB interface with the business  
    method declarations.
```

```
*/
```

```
public interface Person extends PersonInfo, EJBObject {  
}
```

```
package ejava.ejb.personnel;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public class PersonBean implements EntityBean, PersonInfo {
    public String id_;
    public String firstName_;
    public String lastName_;
    public String address_;
    public String phoneNumber_;

    /**
     * Set when the state of the bean has changed and should
     * be synchronized with the database. This is slightly
     * container-specific.
     */
    private boolean dirty_ = false;
    public boolean isDirty() { return dirty_; }

    public void setFirstName(String name) { firstName_ = name; dirty_ = true; }
    public String getFirstName() { return firstName_; }

    public void setLastName(String name) { lastName_ = name; dirty_ = true; }
    public String getLastName() { return lastName_; }

    public void setAddress(String address) { address_ = address; dirty_ = true; }
    public String getAddress() { return address_; }

    public void setPhoneNumber(String number)
    { phoneNumber_ = number; dirty_ = true; }
    public String getPhoneNumber() { return phoneNumber_; }

    public String toXML() {
        StringBuffer text = new StringBuffer();
        text.append("<ejava:person id=\"" + id_ + "\">\n");
        text.append("    <ejava:firstName>" + firstName_ + "</ejava:firstName>\n");
        text.append("    <ejava:lastName>" + lastName_ + "</ejava:lastName>\n");
        text.append("    <ejava:address>" + address_ + "</ejava:address>\n");
        text.append("    <ejava:phoneNumber>" +
            phoneNumber_ + "</ejava:phoneNumber>\n");
        text.append("</ejava:person>\n");

        return text.toString();
    }

    /**
     */
    public PersonPK ejbCreate(String id) {
        id_ = id;
        return null; //as per spec
    }
    public void ejbPostCreate(String id) {}

    /**
     */
    public PersonPK ejbCreate(String id, String firstName, String lastName,
        String address, String phoneNumber) {

        id_ = id;
        firstName_ = firstName;
    }
}
```

```
        lastName_ = lastName;
        address_ = address;
        phoneNumber_ = phoneNumber;

        return null; //as per spec
    }
    public void ejbPostCreate(String id, String firstName, String lastName,
        String address, String phoneNumber) {}

    /**
     * This method is invoked when an instance of this bean class is taken
     * from the bean pool and associated with an EJBObject.
     */
    public void ejbActivate() {}

    /**
     * This method is invoked when an instance if this bean class is about
     * to be dis-associated from an EJBObject and returned to the bean pool.
     */
    public void ejbPassivate() {}

    /**
     * This method is invoked prior to invoking a business or ejbRemove method
     * as a part of synchronizing the cached values with the database. This
     * method must make sure any non-container loaded attributes are fully
     * initialized by the container loaded values.
     */
    public void ejbLoad()      { dirty_ = false; }

    /**
     * This method is invoked just after a business method has changed the
     * state of the cached values. This method must prepare the container-stored
     * values for the container.
     */
    public void ejbStore() { dirty_ = false; }

    /**
     * This method will be invoked just prior to removing the object from the
     * database.
     */
    public void ejbRemove() {}

    private EntityContext entityCtx_;
    public void setEntityContext(EntityContext ctx) { entityCtx_ = ctx; }
    public void unsetEntityContext()              { entityCtx_ = null; }
}
```

```
package ejava.ejb.personnel;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Collection;

/**
 * This interface defines the lifecycle methods for the Person bean to create
 * and find instances.
 */
public interface PersonHome extends EJBHome {
    /**
     * This method creates an empty person that can be later populated with
     * values.
     */
    Person create(String id)
        throws CreateException, RemoteException;

    /**
     * This method creates a fully populated person.
     */
    Person create(String id,
        String firstName, String lastName, String address, String phoneNumber)
        throws CreateException, RemoteException;

    Person findByPrimaryKey(PersonPK pkey)
        throws FinderException, RemoteException;

    Collection findByLikeName(String firstName, String lastName)
        throws FinderException, RemoteException;
}
```

```
package ejava.ejb.personnel;

import java.rmi.RemoteException;

/**
 * This interface defines the methods that are accessible from Person
 * objects. It is defined separate from its EJBObject interface so that
 * both the EJBObject and the EntityBean can both implement the same
 * interface for compile-time consistency checking.
 */
public interface PersonInfo {
    public void setFirstName(String name) throws RemoteException;
    public String getFirstName() throws RemoteException;

    public void setLastName(String name) throws RemoteException;
    public String getLastName() throws RemoteException;

    public void setAddress(String address) throws RemoteException;
    public String getAddress() throws RemoteException;

    public void setPhoneNumber(String number) throws RemoteException;
    public String getPhoneNumber() throws RemoteException;

    public String toXML() throws RemoteException;
}
```

```
package ejava.ejb.personnel;

import java.io.Serializable;

public class PersonPK implements Serializable {
    public String id_;

    public PersonPK() {} //this one is required
    public PersonPK(String id) { id_ = id; } //this one is optional

    public String toString() { return id_; }
    public int hashCode() { return id_.hashCode(); }
    public boolean equals(Object rhs) {
        try {
            return ((PersonPK)rhs).id_.equals(id_);
        }
        catch (ClassCastException ex) {
            return false;
        }
    }
}
```

```
package ejava.ejb.personnel;
```

```
import java.rmi.RemoteException;
```

```
public class PersonnelException extends RemoteException {  
    public PersonnelException() {}  
    public PersonnelException(String reason) { super(reason); }  
}
```

```
package ejava.ejb.personnel;
```

```
import javax.ejb.EJBObject;
```

```
/**
```

```
    This interface is used to define the remote interface to the  
    Registrar.
```

```
*/
```

```
public interface Registrar extends RegistrarBI, EJBObject {}
```

```
package ejava.ejb.personnel;

import java.rmi.RemoteException;

/**
 * The Registrar is the front door to personnel actions by client
 * applications. All compound methods having to do with managing personnel
 * are handled by this interface.
 */
public interface RegistrarBI {
    /**
     * Adds the requested person to the system. Takes care of generating
     * a unique primary key for the individual and recording them.
     */
    Person add(String firstName, String lastName,
               String address, String phoneNumber)
        throws PersonnelException, RemoteException;

    /**
     * Returns an XML listing of each matching person. The first and
     * lastName parameters will be matched using %name% wildcards.
     */
    String listPersonnelByName(String firstName, String lastName)
        throws PersonnelException, RemoteException;
}
```

```
package ejava.ejb.personnel;

import ejava.ejb.uid.UIDGeneratorHome;
import ejava.ejb.uid.UIDGenerator;
import ejava.ejb.uid.UID;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import javax.ejb.EJBException;
import javax.rmi.PortableRemoteObject;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.rmi.RemoteException;
import java.util.Collection;
import java.util.Iterator;

public class RegistrarBean implements RegistrarBI, SessionBean {
    PersonHome pHome_;
    UIDGenerator uidGenerator_;

    /**
     * Generates a uid and adds the person to the database.
     */
    public Person add(String firstName, String lastName,
                     String address, String phoneNumber)
        throws PersonnelException, RemoteException {
        Person newPerson = null;
        try {
            UID uid = uidGenerator_.createUID();
            newPerson = pHome_.create(uid.toString(),
                                     firstName, lastName, address, phoneNumber);
        }
        catch (CreateException ex) {
            throw new PersonnelException(ex.toString());
        }
        catch (Exception ex) {
            throw new EJBException("unable to create person:"+ex);
        }

        return newPerson;
    }

    public String listPersonnelByName(String firstName, String lastName)
        throws PersonnelException, RemoteException {

        StringBuffer result = new StringBuffer();
        result.append("<ejava:personList>");

        try {
            Collection people = pHome_.findByLikeName(firstName, lastName);
            for(Iterator itr=people.iterator(); itr.hasNext(); ) {
                Person person = (Person)itr.next();
                result.append(person.toXML());
            }
        }
        catch (FinderException ex) {
            throw new PersonnelException("error locating matches:"+ex);
        }
        catch (Exception ex) {
            throw new EJBException("unable to complete search:"+ex);
        }
    }
}
```

```
    }

    result.append("</ejava:personList>");
    return result.toString();
}

public void ejbCreate() {}
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}

private SessionContext sessionCtx_;
public void setSessionContext(SessionContext sessionCtx)
    throws RemoteException {

    sessionCtx_ = sessionCtx;
    try {
        InitialContext ctx = new InitialContext();
        Object object = ctx.lookup("java:comp/env/ejb/PersonHome");
        pHome_ = (PersonHome)
            PortableRemoteObject.narrow(object, PersonHome.class);

        object = ctx.lookup("java:comp/env/ejb/UIDGeneratorHome");
        UIDGeneratorHome uidHome_ = (UIDGeneratorHome)
            PortableRemoteObject.narrow(object, UIDGeneratorHome.class);
        uidGenerator_ = uidHome_.find();
    }
    catch (FinderException ex) {
        throw new EJBException("uid generator not located;" + ex);
    }
    catch (NamingException ex) {
        throw new EJBException("error locating ejb objects:" + ex);
    }
    finally {
    }
}
}
```

```
package ejava.ejb.personnel;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;

/**
 * Creates a Registrar Session Bean for use by the client.
 */
public interface RegistrarHome extends EJBHome {
    public Registrar create() throws CreateException, RemoteException;
}
```

```
package ejava.ejb.personnel.client;

import ejava.ejb.personnel.PersonHome;
import ejava.ejb.personnel.Person;
import javax.ejb.FinderException;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.rmi.RemoteException;
import java.util.Properties;
import java.util.Collection;
import java.util.Iterator;

public class Client {
    private static final String USAGE =
        "java ejava.ejb.personnel.Client <username> <password>";
    private String user_;
    private String password_;

    public Client(String user, String password) {
        user_ = user;
        password_ = password;
    }

    public void listPersonnel()
        throws NamingException, RemoteException, FinderException {
        Context ctx = getInitialContext();

        Object object = ctx.lookup("PersonHome");
        /** could not locate ioser12 library
        PersonHome pHome = (PersonHome)
            PortableRemoteObject.narrow(object, PersonHome.class);
        */
        PersonHome pHome = (PersonHome)(object);
        Collection people = pHome.findByLikeName("", "");
        for(Iterator itr=people.iterator(); itr.hasNext(); ) {
            Person person = (Person)itr.next();
            System.out.println(person.toXML());
        }
    }

    private Context getInitialContext() throws NamingException {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
            "weblogic.jndi.WLInitialContextFactory");
        props.put(Context.PROVIDER_URL,
            "t3://localhost:7001");
        if (user_ != null) {
            props.put(Context.SECURITY_PRINCIPAL, user_);
        }
        if (password_ != null) {
            props.put(Context.SECURITY_CREDENTIALS, password_);
        }

        return new InitialContext(props);
    }

    public static void main(String args[]) {
        String user = null;
        String password = null;
    }
}
```

```
switch (args.length) {
    case 1: user = args[0]; break;
    case 2: user = args[0]; password = args[1]; break;
}
try {
    Client client = new Client(user, password);
    client.listPersonnel();
    //client.addPerson();
}
catch (Exception ex) {
    ex.printStackTrace();
}
}
```

```
package ejava.ejb.personnel.client;

import ejava.ejb.personnel.PersonHome;
import ejava.ejb.personnel.Person;
import javax.ejb.FinderException;
import javax.ejb.CreateException;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.rmi.RemoteException;
import java.util.Properties;
import java.util.Collection;
import java.util.Iterator;

public class PersonClient {
    private static final String USAGE =
        "java ejava.ejb.personnel.PersonClient " +
        "[-user <username> " +
        "[-password <password>] " +
        "<-add|-list>";
    private String user_;
    private String password_;

    public PersonClient(String user, String password) {
        user_ = user;
        password_ = password;
    }

    public void listPersonnel()
        throws NamingException, RemoteException,
            FinderException, CreateException {

        PersonHome pHome = getPersonHome();

        Collection people = pHome.findByNameLikeName("%", "%");
        for(Iterator itr=people.iterator(); itr.hasNext(); ) {
            Person person = (Person)itr.next();
            System.out.println(person.toXML());
        }
    }

    public void addPerson(
        String id, String firstName, String lastName,
        String address, String phoneNumber)
        throws NamingException, RemoteException, CreateException {

        PersonHome pHome = getPersonHome();

        Person newPerson = pHome.create(id, firstName, lastName,
            address, phoneNumber);
    }

    protected PersonHome getPersonHome()
        throws NamingException {

        Context ctx = getInitialContext();
        Object object = ctx.lookup("PersonHome");
        PersonHome pHome = (PersonHome)
            PortableRemoteObject.narrow(object, PersonHome.class);
    }
}
```

```
        return pHome;
    }

protected Context getInitialContext() throws NamingException {
    Properties props = new Properties();
    props.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
    props.put(Context.PROVIDER_URL,
        "t3://localhost:7001");
    if (user_ != null) {
        props.put(Context.SECURITY_PRINCIPAL, user_);
    }
    if (password_ != null) {
        props.put(Context.SECURITY_CREDENTIALS, password_);
    }

    return new InitialContext(props);
}

public static void main(String args[]) {
    String user = null;
    String password = null;
    String option = "list";
    String id = null;
    String firstName = null;
    String lastName = null;
    String address = null;
    String phoneNumber = null;

    for(int i=0; i<args.length; i++) {
        if (args[i].equals("-user")) {
            user = args[i+1];
        }
        else if (args[i].equals("-password")) {
            password = args[i+1];
        }
        else if (args[i].equals("-list")) {
            option = "list";
        }
        else if (args[i].equals("-add")) {
            option = "add";
            id = args[i+1];
            firstName = args[i+2];
            lastName = args[i+3];
            address = args[i+4];
            phoneNumber = args[i+5];
        }
    }

    try {
        PersonClient client = new PersonClient(user, password);
        if (option.equals("list")) {
            System.out.println("listing Persons...");
            client.listPersonnel();
        }
        else if (option.equals("add")) {
            System.out.println("adding Person...");
            client.addPerson(id, firstName, lastName,
                address, phoneNumber);
        }
        else {

```

```
        System.out.println("unknown option");
```

```
    }
```

```
}
```

```
catch (Exception ex) {  
    ex.printStackTrace();
```

```
}
```

```
}
```

```
}
```

```
package ejava.ejb.personnel.client;

import ejava.ejb.personnel.RegistrarHome;
import ejava.ejb.personnel.Registrar;
import ejava.ejb.personnel.Person;
import javax.ejb.FinderException;
import javax.ejb.CreateException;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.rmi.RemoteException;
import java.util.Properties;
import java.util.Collection;
import java.util.Iterator;

public class RegistrarClient extends PersonClient {
    private static final String USAGE =
        "java ejava.ejb.personnel.RegistrarClient " +
        "[-user <username>] " +
        "[-password <password>] " +
        "<-add | -list>";

    public RegistrarClient(String user, String password) {
        super(user, password);
    }

    public void listPersonnel()
        throws NamingException, RemoteException, CreateException {

        RegistrarHome rHome = getRegistrarHome();
        Registrar registrar = rHome.create();

        String people = registrar.listPersonnelByName("", "");
        System.out.println(people);
    }

    public void addPerson(
        String firstName, String lastName,
        String address, String phoneNumber)
        throws NamingException, RemoteException, CreateException {

        RegistrarHome rHome = getRegistrarHome();
        Registrar registrar = rHome.create();

        Person newPerson = registrar.add(firstName, lastName,
                                         address, phoneNumber);
    }

    protected RegistrarHome getRegistrarHome()
        throws NamingException {

        Context ctx = getInitialContext();
        Object object = ctx.lookup("RegistrarHome");
        RegistrarHome rHome = (RegistrarHome)
            PortableRemoteObject.narrow(object, RegistrarHome.class);

        return rHome;
    }

    public static void main(String args[]) {
```

```
String user = null;
String password = null;
String option = "list";
String id = null;
String firstName = null;
String lastName = null;
String address = null;
String phoneNumber = null;

for(int i=0; i<args.length; i++) {
    if (args[i].equals("-user")) {
        user = args[i+1];
    }
    else if (args[i].equals("-password")) {
        password = args[i+1];
    }
    else if (args[i].equals("-list")) {
        option = "list";
    }
    else if (args[i].equals("-add")) {
        option = "add";
        firstName = args[i+1];
        lastName = args[i+2];
        address = args[i+3];
        phoneNumber = args[i+4];
    }
}

try {
    RegistrarClient client = new RegistrarClient(user, password);
    if (option.equals("list")) {
        System.out.println("listing Persons...");
        client.listPersonnel();
    }
    else if (option.equals("add")) {
        System.out.println("adding Person...");
        client.addPerson(firstName, lastName,
            address, phoneNumber);
    }
    else {
        System.out.println("unknown option");
    }
}
catch (Exception ex) {
    ex.printStackTrace();
}
}
```

```
include $(MAKEROOT)/java.rules
```

```
PACKAGE_BASE      = ejava.ejb  
PACKAGE_SPEC      = personnel.client  
DEPENDENCIES      =  
DEPENDENTS        =  
LOCAL_JAVAFLAGS   =  
LOCAL_CLASSPATH   = $(EJBCLASS_DIR)  
JAVA_CLASSES      = \  
                  PersonClient \  
                  RegistrarClient
```

```
EJB               = Personnel  
EJBJAR            =  
EJB_CLASSES       =  
METAINF_FILES     =
```

```
PROPERTY_FILES    =
```

```
SCRIPTS           =
```

```
include $(MAKEROOT)/java.cmds
```

```
include $(MAKEROOT)/java.rules
```

```
PACKAGE_BASE      = ejava.ejb
PACKAGE_SPEC      = personnel
DEPENDENCIES      =
DEPENDENTS        = wls client
LOCAL_JAVAFLAGS   =
LOCAL_CLASSPATH   = $(EJBCLASS_DIR)$(ENVSEP)$(EJB_DIR)/UID.jar
JAVA_CLASSES      = \
    PersonnelException \
    PersonInfo \
    PersonPK \
    Person \
    PersonHome \
    RegistrarBI \
    Registrar \
    RegistrarHome
```

```
EJB              = Personnel
EJBJAR           =
EJB_CLASSES      = \
    PersonnelException \
    PersonInfo \
    PersonPK \
    Person \
    PersonHome \
    PersonBean \
    RegistrarBI \
    Registrar \
    RegistrarHome \
    RegistrarBean
METAINF_FILES    = ejb-jar
```

```
PROPERTY_FILES  =
```

```
SCRIPTS         =
```

```
include $(MAKEROOT)/java.cmds
```

```
#!/usr/bin/perl
```

```
$WL_HOME="/apps/weblogic";
#$CLOUD_HOME="$WL_HOME/eval/cloudscape";
#$CLOUD_LIB="$CLOUD_HOME/lib";
$HSQL_HOME="/apps/hypersonicsql";
$HSQL_LIB="$HSQL_HOME/lib";

$classpath="$WL_HOME/classes";
#$classpath="$classpath:$CLOUD_LIB/client.jar";
#$classpath="$classpath:$CLOUD_LIB/tools.jar";
#$classpath="$classpath:$CLOUD_LIB/cloudscape.jar";
$classpath="$classpath:$HSQL_LIB/hsqldb.jar";

$javaProps="-classpath $classpath";
#$javaProps="$javaProps -Dcloudscape.system.home=$CLOUD_HOME/data";

$url="\jdbc:cloudscape:wlsdb;create=true;upgrade=true\";
$driver="COM.cloudscape.core.JDBCdriver";
$sql="personnel_cloudscape.ddl";
$url="jdbc:HypersonicSQL:/usr/local/data/hsqldata/wlsdb";
$driver="org.hsqldb.jdbcDriver";
$user="sa";
$password="";
$sql="personnel_hsqldb.ddl";

$options="-verbose";

$javaClass="utils.Schema";
$javaArgs="$url $driver -u $user -p $password $options $sql";

$javaCommand="java $javaProps $javaClass $javaArgs";

print("$javaCommand\n");
system($javaCommand);
```

```
include $(MAKEROOT)/java.rules

PACKAGE_BASE      = ejava.ejb
PACKAGE_SPEC      = personnel.wls
DEPENDENCIES      =
DEPENDENTS        =
LOCAL_JAVAFLAGS   =
#LOCAL_CLASSPATH  = $(EJBCLASS_DIR)$(ENVSEP)$(EJB_DIR)/UID.jar
LOCAL_CLASSPATH   =
JAVA_CLASSES      =

EJB               = Personnel
EJBJAR            = Personnel
EJB_CLASSES       =
METAINF_FILES     = weblogic-ejb-jar weblogic-cmp-rdbms-jar

PROPERTY_FILES    =

SCRIPTS           =

include $(MAKEROOT)/java.cmds
```

```
drop table Person
;
create table Person (
  id varchar(32) not null,
  firstName varchar(80) not null,
  lastName varchar(80) not null,
  address varchar(80) null,
  phoneNumber char(10) null,
  constraint pk_Person primary key(id)
);
```

```
drop table Person
;
create table Person (
    id varchar(32) not null,
    fname varchar(80) not null,
    lname varchar(80) not null,
    address varchar(80) null,
    phone char(10) null,
    constraint pk_Person primary key(id)
);
```

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-rdbms-bean PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB RDBMS Persistence//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd'>

<weblogic-rdbms-bean>
  <pool-name>thinPool</pool-name>
  <table-name>Person</table-name>
  <attribute-map>
    <object-link>
      <bean-field>id_</bean-field>
      <dbms-column>id</dbms-column>
    </object-link>
    <object-link>
      <bean-field>firstName_</bean-field>
      <dbms-column>firstName</dbms-column>
    </object-link>
    <object-link>
      <bean-field>lastName_</bean-field>
      <dbms-column>lastName</dbms-column>
    </object-link>
    <object-link>
      <bean-field>address_</bean-field>
      <dbms-column>address</dbms-column>
    </object-link>
    <object-link>
      <bean-field>phoneNumber_</bean-field>
      <dbms-column>phoneNumber</dbms-column>
    </object-link>
  </attribute-map>

  <finder-list>
    <finder>
      <method-name>findByLikeName</method-name>
      <method-param>java.lang.String</method-param> <!-- firstName -->
      <method-param>java.lang.String</method-param> <!-- lastName -->
      <finder-query>
        <![CDATA[& (like firstName_ $0) (like lastName_ $1)]]>
      </finder-query>
      <finder-expression>
        <expression-number>0</expression-number>
        <expression-text>"%" + @0 + "%"</expression-text>
        <expression-type>java.lang.String</expression-type>
      </finder-expression>
      <finder-expression>
        <expression-number>1</expression-number>
        <expression-text>"%" + @1 + "%"</expression-text>
        <expression-type>java.lang.String</expression-type>
      </finder-expression>
    </finder>
  </finder-list>
</weblogic-rdbms-bean>
```