3.3.5 Exception: DataInvalid

This exception is thrown when data passed to a method is invalid, for example in case of a data field overflow (name is too long).

```
final public class DataInvalid extends org.omg.CORBA.UserException {
  public DataInvalid();
  public DataInvalid(String explanation);
(...)
}
```

Indices for various components of the address (aidx) and telephone numbers (pidx) into the string sequences returned by getAddr() and getPhones().

```
  public void entry(AddrBook.PersEntry entry);
  public AddrBook.PersEntry entry();
```

Get/set accessor methods for the PersEntry associated with this PersEntryDetail. The entry(PersEntry) method, when called, will automatically refresh the postal address, phone numbers and email address in the PersEntryDetail with data from the database record associated with the PersEntry..

```
  public String[] getAddr();
```

Returns a sequence of strings consisting of the components of the address

```
  public void setAddr(String addr1, String addr2, String city,
    String state, String zip) throws AddrBook.DataInvalid;
```

Sets the postal address information for the record as specified.

```
  public String[] getPhones();

  public void setPhones(String home, String work, String fax)
    throws AddrBook.DataInvalid;
```

Sets the telephone information for the record as specified.

```
  public String getEmail();
Sets the address information for the record as specified.
  public void setEmail(String email) throws AddrBook.DataInvalid;
```

Sets the email address for the record as specified.

```
  public void write() throws AddrBook.AccessViolation;
```

Writes the full record associated with the PersEntry back to the database.

```
  public void refresh() throws AddrBook.AccessViolation;
```

Refreshes the data fields retrievable by getAddr(), getPhones(), getEmail(), and entry().getName() methods with information from the database, overwriting any previous values stored via the corresponding set methods that had not been stored with write().

```
}
```

3.3.4 Exception: AccessViolation

This exception is thrown when the specified record does not exist, or a database error occurs.

```
final public class AccessViolation extends org.omg.CORBA.UserException {
  public AccessViolation();
  public AccessViolation(java.lang.String explanation);
(...)
}
```

```
    throws AddrBook.AccessViolation;
```

Returns a sequence of PersEntries which match the specified first and last name sub-strings. The match is case-insensitive.

```
  public void disconnect();
```

Disconnects from the database, freeing up associated system resources.

```
}
```

3.3.2 PersEntry

A class implementing this interface represents a person, or an entry in the address book.

```
public interface PersEntry extends org.omg.CORBA.Object {
  final public static short nidx_first = (short)0;
  final public static short nidx_last = (short)1;
```

These constants are indices (0-based) for the first and last name in the string sequence returned by PersEntry.getName().

```
  public int persid();
```

This is an automatically generated accessor method for the attribute long. The attribute persid is an identifier used to uniquely refer to a PersEntry. It persists across sessions, modifications, deletions etc.

```
  public String[] getName();
```

Returns a sequence (Java array) of two strings consisting of the components of the name.

```
  public void setName(String firstName, String lastName) throws
    AddrBook.DataInvalid;
```

Sets the name information for the record as specified.

```
  public boolean wasModified() throws AddrBook.AccessViolation;
```

Returns true if the PersEntry has been modified since the object's data was last refreshed, i.e. the client's representation of it could be "dirty".

```
}
```

3.3.3 PersEntryDetail

```
public interface PersEntryDetail extends org.omg.CORBA.Object {
  final public static short aidx_addr1 = (short)0;
  final public static short aidx_addr2 = (short)1;
  final public static short aidx_city = (short)2;
  final public static short aidx_state = (short)3;
  final public static short aidx_zip = (short)4;
  final public static short pidx_home = (short)0;
  final public static short pidx_work = (short)1;
  final public static short pidx_fax = (short)2;
```

3.2 Client Logic (David Wong)

The module address book, addrbook, is the first window to show up. Within this GUI, is the list of personal entries, PersEntry, containing the last names and first names of all the entries saved in the server. When a name is clicked upon, PersEntry will contact the server with getAll(), and search for that particular personal entry detail, PersEntryDetail, containing the last and first names.

The server sends the information back to the GUI and creates a new window with the address fields, the phone number fields, and the email address field. Assuming that a PersEntry is found, those fields will be loaded with information from the server. If the entry is not found, then the fields would be empty. You will have the option to create a new entry newEntry() and fill in each of the slots. setName, setAddr, setPhones, setEmail will call the server and save this information in those specific functions when write() is used to save the entry to the server. This is the same for edit. Once inside the edit page, all the function fields will be editable. And this will be updated when write() is called. There is another function called refresh() which will bring a copy of an existing entry from the server to refresh the GUI. This option will allow the user to bring up the existing entry just in case mistakes are made and the user doesn't remember the original settings. If there is an unwanted entry to be deleted, deleteEntry() will be used to delete the entry from the server.

As for searching the database for a specific entry, the search() would do the trick. The last name and first name fields are open for input. search() will use setName to look for the entry with the particular last and first name. When the valid entry is found, it will be displayed back in the address book window.

3.3 Server Objects (Nicholas Riley)

3.3.1 AddrBook

A class implementing this interface represents the address book server; it is instantiated on the server.

```
public interface AddrBook extends org.omg.CORBA.Object {
  public AddrBook.PersEntry newEntry() throws AddrBook.AccessViolation;
```

Returns a newly allocated PersEntry object which can be written to the database.

```
  public void deleteEntry(long persid)
    throws AddrBook.AccessViolation, AddrBook.DataInvalid;
```

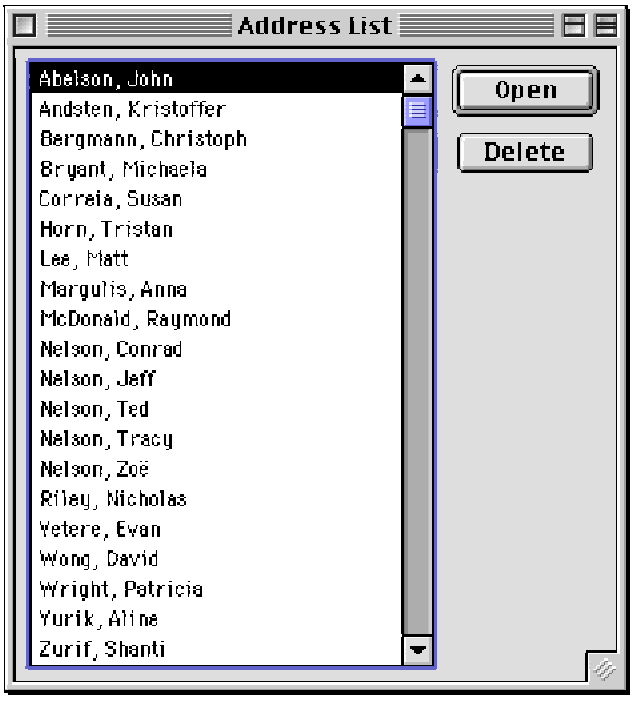
Deletes the PersEntry with the specified persid.

```
  public AddrBook.PersEntry[] getAll() throws AddrBook.AccessViolation;
```

Returns a sequence (Java array) of PersEntries for the entire contents of the address book table.

```
  public AddrBook.PersEntry[] search(String firstName,String lastName)
```

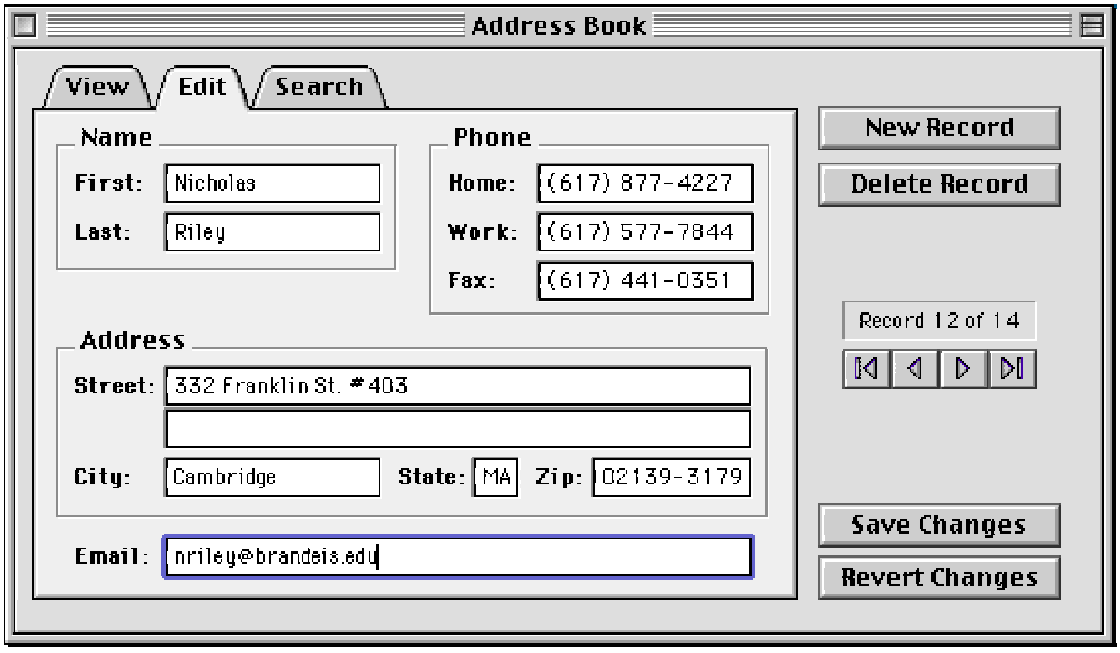
```
      goto PersEntry[n+1]}
    {if (button.PrevRecord is toggled)
      goto PersRecord[n-1]}

  {if (Edit tab toggled)
    goto Edit page
      {if (button.NextRecord is toggled)
        goto PersEntry[n+1]}
      {if (button.PrevRecord is toggled)
        goto PersRecord[n-1]}
      {if (button.NewRecord is toggled)
        call on newEntry();}
      {if (button.DeleteRecord is toggled)
        call on deleteEntry();}
      {if (button.Save is toggled)
        call on write();}
      {if (button.reverseChanges is toggled)
        getAddr PersRecord[n]}
      }
  {if (Search tab is toggled)
    goto Search page
      if (button.search is toggled)
        {get FirstName
         get LastName
         call on AddrBook.search(FirstName, LastName)
         display list PersEntry}
  }

if (button.Delete toggled && PersEntry[n] selected)
        call on delete.Entry();
}
```



```
    const short aidx_addr1 = 0;
    const short aidx_addr2 = 1;
    const short aidx_city = 2;
    const short aidx_state = 3;
    const short aidx_zip = 4;
    StringSeq getPhones();
    void setPhones(in string home, in string work, in string fax)
      raises (DataInvalid);
    const short pidx_home = 0;
    const short pidx_work = 1;
    const short pidx_fax = 2;
    string getEmail();
    void setEmail(in string email) raises (DataInvalid);
    void write() raises (AccessViolation);
    void refresh() raises (AccessViolation);
  };
};
```

3.0 Components

3.1 Client GUI (Anna Margulis)



The GUI will be a Java applet (see sample screen displays, above and below). Here is a basic flow of the way the GUI will operate, in pseudo code.

```
initialize applet Addressbook
list PersEntry
if (button.Open toggled && PersEntry[n] selected){
  { display Entry window
    display PersEntry[n] in View page}
    {if (button.NextRecord is toggled)
```

```
SELECT PERSID, FIRSTNAME, LASTNAME, ... FROM ADDRESSES
WHERE PERSID = 381;

SELECT PERSID, FIRSTNAME, LASTNAME FROM ADDRESSES ORDER BY LASTNAME,
FIRSTNAME;

SELECT PERSID, FIRSTNAME, LASTNAME FROM ADDRESSES WHERE
LOWER(FIRSTNAME) LIKE LOWER('%First');
```

This last query is rather inefficient but necessary to overcome Oracle's case sensitivity. The LOWER() could be omitted for databases that perform case insensitive queries by default, for example Microsoft SQL Server. If necessary, another two columns (e.g. FIRSTNAME_CAP and LASTNAME_CAP) could be added and populated via a trigger with normalized data, to speed queries (at the expense of storage space).

2.2 CORBA

The client Java applet and server Java objects communicate using CORBA. The interface (IDL) specification follows:

```
module AddrBook {
  exception DataInvalid {
    string explanation;
  };
  exception AccessViolation {
    string explanation;
  };
  typedef sequence< string > StringSeq;
  interface PersEntry {
    readonly attribute long persid;
    StringSeq getName();
    void setName(in string firstName, in string lastName) raises
      (DataInvalid);
    const short nidx_first = 0;
    const short nidx_last = 1;
    boolean wasModified() raises (AccessViolation);
  };
  typedef sequence< PersEntry > PersEntrySeq;
  interface AddrBook {
    PersEntry newEntry() raises (AccessViolation);
    void deleteEntry(in long persid) raises (AccessViolation,
      DataInvalid);
    PersEntrySeq getAll() raises (AccessViolation);
    PersEntrySeq search(in string firstName, in string lastName)
      raises (AccessViolation);
    void disconnect();
  };
  interface PersEntryDetail {
    attribute PersEntry entry;
    StringSeq getAddr();
    void setAddr(in string addr1, in string addr2, in string city,
                 in string state, in string zip) raises (DataInvalid);
```

```
/*
   Connect as a user with the appropriate privileges for the first set of
   statements. Assuming that TEMP tablespace already exists.
*/

CREATE TABLESPACE T_ADDRBOOK DATAFILE 'ADDRBOOK.ORA' SIZE 5M;

CREATE USER ADDRBOOK IDENTIFIED BY ADDRBOOK DEFAULT TABLESPACE
T_ADDRBOOK TEMPORARY TABLESPACE TEMP PROFILE DEFAULT
QUOTA 5 M ON T_ADDRBOOK;
GRANT CONNECT TO ADDRBOOK;
GRANT CREATE SEQUENCE TO ADDRBOOK;
GRANT CREATE TABLE TO ADDRBOOK;
ALTER USER ADDRBOOK DEFAULT ROLE ALL;

CONNECT ADDRBOOK/ADDRBOOK;

CREATE TABLE ADDRESSES (
PERSID NUMERIC(8) PRIMARY KEY,
FIRSTNAME VARCHAR2(50),
LASTNAME VARCHAR2(50),
CITY VARCHAR2(80),
STATE VARCHAR2(2),
ZIP VARCHAR2(10),
HOMEPHONE VARCHAR2(15),
WORKPHONE VARCHAR2(15),
FAX VARCHAR2(15),
EMAIL VARCHAR2(80)
);

CREATE UNIQUE INDEX I_NAME ON ADDRESSES (LASTNAME, FIRSTNAME);
CREATE SEQUENCE SEQ_PERSID INCREMENT BY 1 NOMAXVALUE MINVALUE 1 ORDER;
```

2.0 Communication

2.1 JDBC

JDBC is used on the server side only. Testing will be performed with the Oracle 8 “thin” (100% Java) JDBC driver, running on Windows NT, Linux and Mac OS.

A few examples of the types of statements the server objects will execute follow.With the exception of the sequence SELECT statement (the DUAL pseudo-table is peculiar to Oracle), all of these statements are standard SQL-92 and should run on any compliant database.

```
INSERT INTO ADDRESSES (PERSID, FIRSTNAME, LASTNAME, ...)
SELECT SEQ_PERSID.NEXTVAL, 'First', 'Last', ... FROM DUAL;

UPDATE ADDRESSES SET FIRSTNAME = 'First', LASTNAME = 'LAST', ...
WHERE PERSID = 381;

DELETE FROM ADDRESSES WHERE PERSID = 146;
```

AddrBook
An Address Book in Java with CORBA and JDBC

Anna Margulis
Nicholas Riley
David Wong

1.0 System requirements

1.1 Design

AddrBook consists of two components: a client side implemented as a Java applet, and a server side implemented as a collection of Java objects which communicate with the client using CORBA and with a host database using JDBC.

1.2 Client requirements

The client side of AddrBook requires a Web browser or other Java applet hosting environment supporting JDK 1.1.x with Sun's Swing classes (JFC) and a Java ORB such as Inprise's VisiBroker for Java.

1.3 Server setup

A CORBA naming service is required to be located on a predetermined machine, whose address is known to the client. The Java server objects may be located on any computer, but the Java sandbox restrictions on applets may make it necessary to use a proxy server or to run the objects on the same machine as that from which the applet is served.

The server objects should be permitted unrestricted network access in order to facilitate JDBC connections. A Java ORB should also be provided. As with the client, the objects should be configured to be aware of the address of the naming service. In addition, the database JDBC URL and driver class files should be provided. For this implementation, Oracle 7.3.3 and 8.0.5 running on Windows NT Server 4.0 were used.

The following SQL statements can be run on the server in order to initialize the database. Some syntax is specific to Oracle; other databases' syntaxes for equivalent operations may be different.