Matisse® Java Programmer's Guide

May 2013

MATISSE Java Programmer's Guide

PDF generated 30 April 2013

Contents

1   Introduction

Scope of This Document

This document is intended to help Java programmers learn the aspects of Matisse design and programming that are unique to the Matisse Java binding.

Aspects of Matisse programming that the Java binding shares with other interfaces, such as basic concepts and schema design, are covered in *Getting Started with Matisse*.

Future releases of this document will add more advanced topics. If there is anything you would like to see added, or if you have any questions about or corrections to this document, please send e-mail to `support@matisse.com`.

Before Reading This Document

Throughout this document, we presume that you already know the basics of Java programming and either relational or object-oriented database design, and that you have read the relevant sections of *Getting Started with Matisse*.

Before Running the Examples

Before running the following examples, you must do the following:

- Install Matisse 9.0.x or later.

- Install the Java Development Kit version 7 or later for your operating system (a free download from `java.com or oracle.com`).
  Or use the preinstalled Java version on your MacOS machine.

- Set the `MATISSE_HOME` and `JAVA_HOME` environment variables to the top-level directories of the Matisse and JDK installations. On MS Windows, you can set environment variables in 'Control Panel -> System'.

- If you are running Windows, append `;%matisse_home%\bin;%java_home%\bin` to the `PATH` user variable.

- Download and extract the Java sample code from the Matisse Web site:

  `http://www.matisse.com/developers/documentation/`

  The sample code files are grouped in subdirectories by chapter number. For example, the code snippets from the following chapter are in the `chap_2` directory.

- Create and initialize a database. You can simply start the Matisse Enterprise Manager, select the database 'example' and right click on 'Re-Initialize'.

- From a Unix shell prompt or on MS Windows from a 'Command Prompt' window, change to the `chap_x` subdirectory in the directory where you installed the examples.

- If applicable, load the ODL file into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema'. For example you may import `chap_3/objects.odl` for the Chapter 3 demo.

- Generate Java class files.

  ```
  mt_sdl stubgen --lang java -f objects.odl
  ```

- Build the application:

  ```
  Windows:
  javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

  UNIX:
  javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
  ```

- For the examples that include an ODL file, you may optionally create a subdirectory named `docs`, then generate the API reference for your application schema (single command, all on one line). For instance:

  ```
  javadoc -public -classpath .;%MATISSE_HOME%\lib\matisse.jar -d docs Person.java
  Employee.java
  ```

- Run the application. For instance in `chap_3`:

  ```
  java -classpath .;%MATISSE_HOME%\lib\matisse.jar ObjectsExample host database
  ```

2   Connection and Transaction

All interaction between client Java applications and Matisse databases takes place within the context of transactions (either explicit or implicit) established by database connections, which are transient instances of the `MtDatabase` class. Once the connection is established, your Java application may interact with the database using the schema-specific methods generated by `mt_sdl`. The following sample code shows a variety of ways of connecting with a Matisse database.

Note that in this chapter there is no ODL file as you do not need to create an application schema.

Building the Examples

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `chap_2` directory in your installation (under `java_examples`).

3. Build the application:

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

UNIX:
javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
```

Read Write Transaction

The following code connects to a database, starts a transaction, commits the transaction, and closes the connection:

```
try
    {
      MtDatabase db = new MtDatabase(args[0], args[1]);

      db.open();
      db.startTransaction();

      System.out.println("Connection and read-write access to "
                      + db.toString());
      // read/write access

      db.commit();
      db.close();
    }
    catch (MtException mte)
    {
      System.out.println("MtException : " + mte.getMessage());
    }
```

Read-Only Access

The following code connects to a database in read-only mode, suitable for reports:

```
try
    {
      MtDatabase db = new MtDatabase(args[0], args[1]);

      db.open();
      db.startVersionAccess();

      System.out.println("Connection and read-only access to "
                         + db.toString());

      // version connect (=>read-only access)
      db.endVersionAccess();
      db.close();
    }
    catch (MtException mte)
    {
      System.out.println("MtException : " + mte.getMessage());
    }
```

Version Access

The following code illustrates methods of accessing various versions of a database.

```
try
    {
      MtDatabase db = new MtDatabase(args[0], args[1]);

      db.open();

      db.startTransaction();
      System.out.println("Version list before regular commit:");
      listVersions(db);
      // read/write access
      db.commit();

      db.startTransaction();
      System.out.println("Version list after regular commit:");
      listVersions(db);
      // another read/write access
      String verName = db.commit("test");
      System.out.println("Commit to version named: " + verName);

      db.startVersionAccess();
      System.out.println("Version list after named commit:");
      listVersions(db);
      // read-only access on the latest version.
      db.endVersionAccess();

      db.startVersionAccess(verName);
      System.out.println("Sucessful access within version: " + verName);
      // read-only access on a named version. It's not possible to
```

```
      // access a named version in read/write (transaction) mode.
      db.endVersionAccess();

      db.close();
    }
    catch (MtException mte)
    {
      System.out.println("MtException : " + mte.getMessage());
    }

  public static void listVersions(MtDatabase db) {
    MtVersionIterator i = db.versionIterator();
    while (i.hasNext()) {
      String versionName = i.next();
      System.out.println("\t" + versionName);
        }
        i.close();
  }
```

JDBC Connection

The following code accesses a Matisse database using JDBC two ways: first using pure JDBC, then using a mix of JDBC and Matisse methods.

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

    if (args.length < 2)
    {
      System.out.println("Need to specify <HOST> <DATABASE>");
      System.exit(-1);
    }

    String hostname = args[0];
    String dbname = args[1];

        // Build a JDBC connection
    connectFromJDBC(hostname, dbname);
        // Build a JDBC connection from a MtDatabase connection
    connectFromMtDatabase(hostname, dbname);
  }

  /**
   * Create a JDBC connection. (Does not require com.matisse.* import.
   */
  public static void connectFromJDBC(String host, String dbname) {
    System.out.println("=========== connectFromJDBC =========\n");
    try {
      Class.forName("com.matisse.sql.MtDriver");

      String url = "jdbc:mt://" + host + "/" + dbname;

      System.out.println("Query class names from the JDBC connection: " + url);

      Connection jcon = DriverManager.getConnection(url);
```

```
    // Regular JDBC access

    Statement stmt = jcon.createStatement();
    String query = "SELECT MtName FROM MtClass";
    ResultSet rs = stmt.executeQuery(query);
    System.out.println("Result from: " + query);

    while (rs.next()) {
      System.out.println(rs.getString("MtName"));
    }
    System.out.println("\ndone.");
  } catch (ClassNotFoundException e) {
    System.out.println("Matisse JDBC Driver class not found, check your CLASSPATH");
  } catch (SQLException e) {
    System.out.println("SQLException thrown:  " + e.getMessage());
  }
}

/**
 * Create a JDBC connection obtained through a MtDatabase connection
 */
public static void connectFromMtDatabase(String host, String dbname) {
  System.out.println("=========== connectFromMtDatabase ==========\n");

  try {
    MtDatabase db = new MtDatabase(host, dbname);

    db.open();
    System.out.println("Query class names from a JDBC connection obtained through a
MtDatabase connection");
    Connection jcon = db.getJDBCConnection();
    try {
      // Regular JDBC access
      Statement stmt = jcon.createStatement();
              String query = "SELECT * FROM MtClass";
      ResultSet rs = stmt.executeQuery(query);
              System.out.println("Result from: " + query);

      while (rs.next()) {
        System.out.println(rs.getString("MtName"));
      }
      jcon.close();
      System.out.println("\ndone.");
    } catch (SQLException e) {
      System.out.println("SQLException thrown:  " + e.getMessage());
    }
  } catch (MtException mte) {
    System.out.println("MtException : " + mte.getMessage());
      }
}
```

Specific Options

This example shows how to enable the local client-server memory transport and to set or read various connection options and states.

```
class AdvancedConnect {
  static MtDatabase db;

  public static void main(String[] args) {

    if (args.length < 2)
    {
      System.out.println("Need to specify <HOST> <DATABASE>");
      System.exit(-1);
    }
    try
    {
      db = new MtDatabase(args[0],args[1]);

      if (System.getProperty("MT_MEM_TRANS") != null)
        db.setOption(MtDatabase.MEMORY_TRANSPORT, MtDatabase.ON);

      if (System.getProperty("MT_DATA_ACCESS") != null)
      {
        db.setOption(MtDatabase.DATA_ACCESS_MODE,
                  Integer.parseInt(System.getProperty("MT_DATA_ACCESS")));
      }

      db.open(System.getProperty("dbuser"),
            System.getProperty("dbpasswd"));

      start(isReadOnly());
      printState();

      end();

      db.close();
    }
    catch (MtException mte)
    {
      System.out.println("MtException : " + mte.getMessage());
    }
  }

  static void start(boolean readonly) {
    if (readonly)
      db.startVersionAccess();
    else
      db.startTransaction();
  }

  static void end() {
    if (db.isVersionAccessInProgress())
      db.endVersionAccess();
    else if (db.isTransactionInProgress())
      db.commit();
    else
      System.out.println("No transaction/version access in progress");
  }

  static boolean isMemoryTransportOn() {
    return db.getOption(MtDatabase.TRANSPORT_TYPE) == MtDatabase.MEM_TRANSPORT;
  }
```

```
static boolean isReadOnly()
  {
    return (db.getOption(MtDatabase.DATA_ACCESS_MODE) ==
          MtDatabase.DATA_READONLY);
  }

static void printState() {
  if (!db.isConnectionOpen()) {
    dbmsg("not connected");
  } else {
    if (db.isTransactionInProgress())
      dbmsg("read-write transaction underway");
    else if (db.isVersionAccessInProgress())
      dbmsg("read-only version access underway");
    else
      dbmsg("no transaction underway");
  }
  dbmsg("MEMORY_TRANSPORT is " + (isMemoryTransportOn() ? "on" : "off"));

  dbmsg("DATA_ACESS_MODE is " +
        (isReadOnly() ? "readonly" : "readwrite"));

}

static void dbmsg(String msg) {
  System.out.println("database " + db.getName()
                    + " on server " + db.getHost()
                    + ":  " + msg);
}
}
```

More about MtDatabase

As illustrated by the previous sections, the `MtDatabase` class provides all the methods for database connections and transactions. The reference documentation for the `MtDatabase` class is included in the Matisse Java Binding API documentation located from the Matisse installation root directory in `docs/java/api/index.html`.

3   Working with Objects and Values

This chapter explains how to manipulate object with the object interface of the Matisse Java binding. The object interface allows you to directly retrieve objects from the Matisse database without Object-Relational mapping, navigate from one object to another through the relationship defined between them, and update properties of objects without writing SQL statements.

The object interface can be used with JDBC as well. For example, you can retrieve objects with JDBC, then use the object interface to navigate to other objects from these objects, or update properties of these objects using the accessor methods defined on these classes.

Running ObjectsExample

This sample program creates objects from 2 classes (`Person` and `Employee`), lists all `Person` objects (which includes both objects, since `Employee` is a subclass of `Person`), deletes objects, then lists all `Person` objects again to show the deletion. Note that because `FirstName` and `LastName` are not nullable, they *must* be set when creating an object.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `chap_3` directory in your installation (under `java_examples`).

3. Load `objects.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `chap_3/objects.odl` for this demo.

4. Generate Java class files:

   ```
   mt_sdl stubgen --lang java -sn examples.java_examples.chap_3 -f examples.odl
   ```

5. Build the application:

   ```
   Windows:
   javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

   UNIX:
   javac -classpath .:$MATISSE_HOME/lib/matisse.jar *.java
   ```

6. Launch the application:

   ```
   Windows:
   java -classpath .;%MATISSE_HOME%\lib\matisse.jar ObjectsExample host database

   UNIX:
   java -classpath .:$MATISSE_HOME/lib/matisse.jar ObjectsExample host database
   ```

Creating Objects

This section illustrates the creation of objects. The stubclass provides a default constructor which is the base constructor for creating persistent objects.

```
public Person(com.matisse.MtDatabase db) {
```

```
        super(getClass(db));
    }

    // Create a new Person object (instance of class Person)
    Person p = new Person(db);
    p.setFirstName("John");
    p.setLastName("Smith");
    p.setAge(42);
    PostalAddress a = new PostalAddress(db);
    a.setCity("Portland");
    a.setPostalCode("97201");
    p.setAddress(a);

    // Create a new Employee object
    Employee e = new Employee(db);
    e.setFirstName("Jane");
    e.setLastName("Jones");
    // Age is nullable we can leave it unset
    e.setHireDate(new GregorianCalendar());
    e.setSalary(new BigDecimal(85000.00));
```

If your application need to create a large number of objects all at once, we recommend that you use the `preallocate()` method defined on `MtDatabase` which provide a substantial performance optimization.

```
    db.startTransaction();

    // Optimize the objects loading
    // Preallocate OIDs so objects can be created in the client workspace
    // without requesting any further information from the server
    db.preallocate(DEFAULT_ALLOCATOR_CNT);

    for (int i = 1; i <= 100; i++) {
        // Create a new Employee object
        Employee e = new Employee(db);
        String fname = fNameSample[sampleSeq.nextInt(MAX_SAMPLES)];
        String lname = lNameSample[sampleSeq.nextInt(MAX_SAMPLES)];
        e.setFirstName(fname);
        e.setLastName(lname);
        e.setHireDate(new GregorianCalendar());
        e.setSalary(new BigDecimal(salarySample[sampleSeq.nextInt(MAX_SAMPLES)]));
        PostalAddress a = new PostalAddress(db);
        int addrIdx = sampleSeq.nextInt(MAX_SAMPLES);
        a.setCity(addressSample[addrIdx][0]);
        a.setPostalCode(addressSample[addrIdx][1]);
        e.setAddress(a);
        System.out.println("Employee "+i+ " " + fname + " " + lname + "  created.");
        if (i % OBJECT_PER_TRAN_CNT == 0) {
            db.commit();
            db.startTransaction();
        }
        // check the remaining number of preallocated objects.
        if (db.numPreallocated() < 2) {
            db.preallocate(DEFAULT_ALLOCATOR_CNT);
        }
    }

    if (db.isTransactionInProgress())
        db.commit();
```

Listing Objects

This section illustrates the enumeration of objects from a class. The `instanceIterator()` static method defined on a generated stubclass allows you to enumerate the instances of this class and its subclasses. The `getInstanceNumber()` method returns the number of instances of this class.

```java
// List all Person objects
System.out.println("\n" + Person.getInstanceNumber(db) +
                    " Person(s) in the database.");
Iterator<Person> iter = Person.instanceIterator(db);
while (iter.hasNext()) {
    Person x = iter.next();
    System.out.println("\t" + x.getFirstName() + " " + x.getLastName()  +
            " from " + (x.getAddress() != null ? x.getAddress().getCity() : "???") +
            " is a " + x.getMtClass().getMtName());
}
```

The `ownInstanceIterator()` static method allows you to enumerate the own instances of a class (excluding its subclasses). The `getOwnInstanceNumber()` method returns the number of instances of a class (excluding its subclasses).

```java
// List all Person objects (excluding Employee sub-class)
System.out.println("\n" + Person.getOwnInstanceNumber(db) +
                    " Person(s) (excluding subclasses) in the database.");

MtObjectIterator<Person> iter = Person.<Person>ownInstanceIterator(db);

while (iter.hasNext()) {
    Person x = iter.next();
    System.out.println("\t" + x.getFirstName() + " " + x.getLastName()  +
            " from " + (x.getAddress() != null ? x.getAddress().getCity() : "???") +
            " is a " + x.getMtClass().getMtName());
}
iter.close();
```

Deleting Objects

This section illustrates the removal of objects. The remove() method delete an object.

```java
// Remove created objects
Person p;
...
// NOTE: does not remove the object sub-parts
p.remove();
```

To remove an object and its sub-parts, you need to override the deepRemove() method in the stubclass to meet your application needs. For example the implementation of `deepRemove()` in the `Person` class that contains a reference to a `PostalAddress` object is as follows:

```java
public void deepRemove() {
    PostalAddress pAddr = getAddress();
    if (pAddr != null)
        pAddr.deepRemove();

    super.deepRemove();
```

```
}
```

```
    Person p;
    ...
     p.deepRemove();
```

The `removeAllInstances()` method defined on MtClass delete all the instances of a class.

```
    Person.getClass(db).removeAllInstances();
```

Comparing Objects

This section illustrates how to compare objects. Persistent objects must be compared with the `equal()` method. You can't compare persistent object with the `==` operator.

```
    Person p1;
    Person p2;
    ...
    if(p1.equals(p2))
        System.out.println("Same objects");
```

Running ValuesExample

This example shows how to get and set values for various Matisse data types including Null values, and how to check if a property of an object is a Null value or not.

This example uses the database created for `ObjectsExample`. It creates objects, then manipulates its values in various ways.

To launch the application:

```
    Windows:
    java -classpath .;%MATISSE_HOME%\lib\matisse.jar ObjectsExample host database

    UNIX:
    java -classpath .;$MATISSE_HOME/lib/matisse.jar ValuesExample host database
```

Setting and Getting Values

This section illustrates the set, update and read object property values. The stubclass provides a set and a get method for each property defined in the class.

```
    Employee e = new Employee(db);

    // Setting strings
    e.setFirstName("John");
    e.setLastName("Jones");

    // Setting numbers
    e.setAge(42);
    System.out.println("\t" + e.getMtClass().getMtName()+ ": " +
```

```
                        e.getFirstName() + " " +
                        e.getLastName());
// suppresses output if no value set
if (!e.isAgeNull())
    System.out.println("\t" + e.getAge() + " years old");

// Setting an attribute of type String to NULL
e.setComment(null);

// Setting an attribute of type int to NULL
e.setNull(Employee.getAgeAttribute(db));
```

Removing Values

This section illustrates the removal of object property values. Removing the value of an attribute will return the attribute to its default value.

```
Employee e;

// Removing value returns attribute to default
e.removeAge();

if (!e.isAgeNull())
    System.out.println("\t" + e.getAge() + " years old");
else
    System.out.println("\tAge: null" +
        (e.isDefaultValue(Employee.getAgeAttribute(db)) ? " (default value)" : ""));
```

Streaming Values

This section illustrates the streaming of blob-type values (`MT_BYTES`, `MT_AUDIO`, `MT_IMAGE`, `MT_VIDEO`). The stubclass provides streaming methods (`setPhotoElements()`, `getPhotoElements()`) for each blob-type property defined in the class. It also provides a method (`getPhotoSize()`) to retrieve the blob size without reading it.

```
// Setting blobs

// set to 512 for demo purpose
// a few Mega-bytes would be more appropriate
// for real multimedia objects (audio, video, high-resolution photos)
int bufSize = 512;
int num, total;
byte buffer[] = new byte[bufSize];
try {
    InputStream is = new FileInputStream("matisse.gif");
    // reset the stream
    e.setPhotoElements(buffer, MtType.BEGIN_OFFSET, 0, true);
    do {
        num = is.read(buffer, 0, bufSize);
        if (num > 0) {
            e.setPhotoElements(buffer, MtType.CURRENT_OFFSET, num,
                                    false);
        }
    } while (num == bufSize);
```

```
        is.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    System.out.println("Image of " + e.getPhotoSize() + " bytes stored.");



    // Getting blobs (save value of e.Photo as out.gif in the
    // program directory)
    total = 0;
    try {
        OutputStream os = new FileOutputStream("out.gif");
        // reset the stream
        e.getPhotoElements(buffer, MtType.BEGIN_OFFSET, 0);
        do {
            num = e.getPhotoElements(buffer, MtType.CURRENT_OFFSET,
                                          bufSize);
            if (num > 0) {
                os.write(buffer, 0, num);
            }
            total += num;
        } while (num == bufSize);
        os.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
```

4   Working with Relationships

One of the major advantages of the object interface of the Matisse .NET binding is the ability to navigate from one object to another through a relationship defined between them. Relationship navigation is as easy as accessing an object property.

Running RelationshipsExample

This example creates several objects, then manipulates the relationships among them in various ways.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `chaps_4_5_6` directory (under `java_examples`).

3. Load `examples.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `chaps_4_5_6/examples.odl` for this demo.

4. Generate Java class files:

    ```
    mt_sdl stubgen --lang java -f examples.odl
    ```

5. Build the application:

    ```
    Windows:
    javac -classpath .;%MATISSE_HOME%\lib\matisse.jar
    examples\java_examples\chaps_4_5_6\*.java *.java

    UNIX:
    javac -classpath .:$MATISSE_HOME/lib/matisse.jar
    *examples/java_examples/chaps_4_5_6/*.java .java
    ```

6. Launch the application:

    ```
    Windows:
    java -classpath .;examples\java_examples\chaps_4_5_6;%MATISSE_HOME%\lib\matisse.jar
    RelationshipsExample host database

    UNIX:
    java -classpath .:examples/java_examples/chaps_4_5_6:$MATISSE_HOME/lib/matisse.jar
    RelationshipsExample host database
    ```

Setting and Getting Relationship Elements

This section illustrates the set, update and get object relationship values. The stubclass provides a set and a get method for each relationship defined in the class.

```
Manager m1 = new Manager(db);
...
// Set a relationship
// Need to report to someone since the relationship
// cardinality minimum is set to 1
```

```
m1.setReportsTo(m1);

Employee e = new Employee(db);
...
// Set a relationship
e.setReportsTo(m1);

Person c1 = new Person(db);
...
Person c2 = new Person(db);
...
// Set successors
m2.setChildren(new Person[] {c1, c2});


...
// Get all successors
Person[] c = m2.getChildren();
```

Adding and Removing Relationship Elements

This section illustrates the adding and removing of relationship elements. The stubclass provides a append, a remove and a clear method for each relationship defined in the class.

```
Person c3 = new Person(db);
...
// add successors
m2.appendChildren(new Person[] {c3});
...
// removing successors (this only breaks links, it does not
// remove objects)
m2.removeChildren(new Person[] {c2});

// clearing all successors (this only breaks links, it does
// not remove objects)
m2.clearChildren();
```

Listing Relationship Elements

This section illustrates the listing of relationship elements for one-to-many relationships. The stubclass provides an iterator method for each one-to-many relationship defined in the class.

```
// Iterate when the relationship is large is always more efficient
Iterator<Person> i = m2.childrenIterator();
while (i.hasNext())
    System.out.println(" "+i.next().getFirstName());
```

Counting Relationship Elements

This section illustrates the counting of relationship elements for one-to-many relationships. The stubclass provides an get size method for each one-to-many relationship defined in the class.

```
// Get the relationship size without loading the Java objects
```

```
// which is the fast way to get the size
int childrenCnt = m2.getChildrenSize();

System.out.println("\t" + m2.getFirstName() + " has " + childrenCnt + " children");

// an alternative to get the relationship size
// but the Java objects are loaded before you can get the count
childrenCnt = m2.getChildren().length;
```

5   Working with Indexes

While indexes are used mostly by the SQL query optimizer to speed up queries, the Matisse Java binding also provides the index query APIs to look up objects based on a key value(s). The stubclass defines both lookup methods and iterator methods for each index defined on the class.

Running IndexExample

This example uses the database created for `RelationshipsExample`. Using the `PersonName` index, it checks whether the database contains an entry for a person matching the specified name. Run it using the following command:

```
Windows:
java -classpath .;examples\java_examples\chaps_4_5_6;%MATISSE_HOME%\lib\matisse.jar
IndexExample host database firstName lastName

UNIX:
java -classpath .:examples/java_examples/chaps_4_5_6:$MATISSE_HOME/lib/matisse.jar
IndexExample host database firstName lastName
```

The application will list the names in the database, indicate whether the specified name was found, and return results within a sample range (defined in the source) using an iterator.

Index Lookup

This section illustrates retrieving objects from an index. The stubclass provides a lookup and a iterator method for each index defined on the class.

```java
// the lookup function returns null to represent no match
Person found = Person.lookupPersonName(db, lastName, firstName);

// open an iterator for a specific range
String fromFirstName = "Fred";
String toFirstName = "John";
String fromLastName = "Jones";
String toLastName = "Murray";
System.out.println("\nLookup from \"" +
                   fromFirstName + " " + fromLastName + "\" to \"" +
                   toFirstName + " " + toLastName + "\"");

MtObjectIterator<Person> ppIter = Person.<Person>personNameIterator(db, fromLastName,
                                                                    fromFirstName,
                                                                    toLastName,
                                                                    toFirstName);
while (ppIter.hasNext()) {
    Person p = ppIter.next();
    System.out.println("  " + p.getFirstName() + " " + p.getLastName());
}
ppIter.close();
```

Index Lookup Count

This section illustrates retrieving the object count for a matching index key. The `getObjectNumber()` method is defined on the `MtIndex` class.

```
Object[] key = new Object[] { lastName, firstName } ;
long count = Person.getPersonNameIndex(db).getObjectNumber(key);
System.out.println(count + " objects retrieved");
```

Index Entries Count

This section illustrates retrieving the number of entries in an index. The `getIndexEntriesNumber()` method is defined on the `MtIndex` class.

```
long count = Person.getPersonNameIndex(db).getIndexEntriesNumber();
System.out.println(count + " entries in the index");
```

6   Working with Entry-Point Dictionaries

An entry-point dictionary is an indexing structure containing keywords derived from a value, which is especially useful for full-text indexing. While the entry-point dictionary can be used with SQL query using `ENTRY_POINT` keyword, the object interface of the Matisse Java binding also provides APIs to directly retrieve objects using the entry-point dictionaries.

Running EPDictExample

This example uses the database created for `RelationshipsExample`. Using the `commentDict` entry-point dictionary, the example retrieves the `Person` objects in the database with `Comments` fields containing a specified character string. To run it, use the following syntax:

```
Windows:
java -classpath .;examples\java_examples\chaps_4_5_6;%MATISSE_HOME%\lib\matisse.jar
EPDictExample host database search_string

UNIX:
java -classpath .:examples/java_examples/chaps_4_5_6:$MATISSE_HOME/lib/matisse.jar
EPDictExample host database search_string
```

Entry-Point Dictionary Lookup

This section illustrates retrieving objects from an entry-point dictionary. The stubclass provides access to lookup methods and iterator methods for each entry-point dictionary defined on the class.

```
// the lookup function returns null to represent no match
// if more than one match an exception is raised
Person found = (Person)Person.getCommentDictDictionary(db).lookup(searchstring);

long hits = 0;

// open an iterator on the matching person objects
MtObjectIterator<Person> pIter = Person.commentDictIterator(db, searchstring);
Person person = null;
while ((person = pIter.next()) != null) {
        System.out.println("  " + person.getFirstName() + " " +
                                person.getLastName());
        hits++;
}
System.out.println(hits + " Person(s) with 'comment' containing '" + searchstring + "'");
```

Entry-Point Dictionary Lookup Count

This section illustrates retrieving the object count for a matching entry-point key. The `getObjectNumber`() method is defined on the `MtEntryPointDictionary` class.

```
long count = Person.getCommentDictDictionary(db).getObjectNumber(searchstring, null);
System.out.println(count + " matching object(s) retrieved");
```

7   Working with Versions

For an introduction to Matisse version access and historical versions, see *Getting Started with Matisse*.

This example is a very simple demonstration of version access using a simple one-class schema with three attributes:

```
module examples {

  module java_examples {

    module chap_7 {

      interface Person : persistent
      {
        attribute Integer Id;
        attribute String FirstName = "(unset)";
        attribute String LastName = "(unset)";

        mt_index personId
          criteria {person::Id MT_ASCEND};
      };

    };

  };

};
```

`VersionExample` allows you to create and modify objects. Whenever an object is created or modified, the application automatically creates a new version (database snapshot).

To create an object, use the set command specifying a new `id` value:

        set *id attribute_name value*

To modify an object, use the set command specifying the `id` value of the object.

        set *id attribute_name value*

To list all versions, objects, and values, use the dump command:

        dump

Building VersionExample

1.   Follow the instructions in *Before Running the Examples* on page 6.

2.   Create and initialize a database as described in *Getting Started with Matisse*. If you wish, you may re-initialize the database you created in the last chapter.

3.   Change to the `chap_7` directory (under `java_examples`).

4. Load `versions.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `chap_7/versions.odl` for this demo.

5. Generate Java class files:

```
mt_sdl stubgen ---lang java -f versions.odl
```

6. Build the application:

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisse.jar
examples\java_examples\chap_7\*.java *.java

UNIX:
javac -classpath .:$MATISSE_HOME/lib/matisse.jar
examples/java_examples/chap_7/*.java *.java
```

Running VersionExample

1. Enter the following command to create an object:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host set 1 FirstName John

UNIX:
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host set 1 FirstName John
```

This creates an object with `id`=1, `FirstName`=John, and `LastName` unset, and command-line output similar to the following (your version name may vary):

```
New Version VerEx00000005 created
```

You can verify this by entering the following command:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host dump

UNIX:
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host dump
```

Which will produce output similar to the following (your version name may vary):

```
Version VEREX00000005
      1: John (unset)
```

2. Enter the following command to modify the object:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host set 1 LastName Smith

UNIX:
```

```
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host set 1 LastName Smith
```

This creates a second version of the object, with `LastName`= Smith. You can see this by repeating the dump command shown in step 1:

```
Version VerEx00000005
      1: John (unset)
Version VEREX100000007
      1: John Smith
```

3. Modify the object again:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host set 1 FirstName Jack

UNIX:
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host set 1 FirstName Jack
```

This creates a third version of the object, with `FirstName`= Jack. You can see this by repeating the dump command:

```
Version VEREX00000005
      1: John (unset)
Version VEREX00000007
      1: John Smith
Version VEREX00000009
      1: Jack Smith
```

4. Now add a second object:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host set 2 FirstName Jane

UNIX:
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host set 2 FirstName Jane
```

5. And modify the second object:

```
Windows:
java -classpath .;examples\java_examples\chap_7;%MATISSE_HOME%\lib\matisse.jar
VersionExample database@host set 2 LastName Jones

UNIX:
java -classpath .:examples/java_examples/chap_7:$MATISSE_HOME/lib/matisse.jar
VersionExample database@host set 2 LastName Jones
```

6. Run the dump command again and you will see that there are now four versions of the database, one for each modification you made:

```
Version VEREX00000005
      1: John (unset)
Version VEREX00000007
      1: John Smith
```

```
Version VEREX00000009
        1: Jack Smith
Version VEREX0000000B
        2: Jane (unset)
        1: Jack Smith
Version VEREX0000000D
        2: Jane Jones
        1: Jack Smith
```

You can also view a list of these versions in Matisse Enterprise Manager, by selecting the Database Snapshots node under Data.

Creating a Version

This example shows how to create a historical version (database snapshot). Versions are created at commit time. You need to provide a version prefix name to the `commit()` method.

```
db.startTransaction();
Person p = new Person(db);
p.setId(1);
p.setFirstName("John");
p.setLastName("Doe");
String vname = db.commit("VerEx");
System.out.println("Version " + vname);
```

Accessing a Version

This example shows how to access a historical version (database snapshot). You need to provide the version name (or version prefix if unique) to the `startVersionAccess()` method.

```
db.startVersionAccess("VerEx");
Person p = Person.lookupPersonId(db, id);
System.out.println(p.getFirstName());
System.out.println(p.getLastName());
db.endVersionAccess();
```

Listing Versions

This example shows how to list historical versions (database snapshots). The `MtVersionIterator` iterator allows you to enumerate the versions.

```
db.startVersionAccess();
MtVersionIterator it = db.versionIterator();
while (it.hasNext()) {
    System.out.println("Version " + it.next());
}
it.close();
db.endVersionAccess();
```

8 Working with JDBC

Running JDBCExample

This sample program demonstrates how to manipulate objects via JDBC. It creates objects (`Person` `Employee and Manager`) and it executes SELECT statements to retrieve objects. It also shows how to create SQL methods and execute them.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `JDBC` directory in your installation (under `java_examples`).

3. Load `examples.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `JDBC/examples.odl` for this demo.

4. Generate Java class files:

   ```
   mt_sdl stubgen --lang java -f examples.odl
   ```

5. Build the application:

   ```
   Windows:
   javac -classpath .;%MATISSE_HOME%\lib\matisse.jar
   examples\java_examples\jdbc\*.java *.java

   UNIX:
   javac -classpath .;$MATISSE_HOME/lib/matisse.jar examples/java_examples/jdbc/*.java
   *.java
   ```

6. Launch the application:

   ```
   Windows:
   java -classpath .;examples\java_examples\jdbc;%MATISSE_HOME%\lib\matisse.jar
   JDBCExample host database

   UNIX:
   java -classpath .:examples/java_examples/jdbc:$MATISSE_HOME/lib/matisse.jar
   JDBCExample host database
   ```

Connecting to a Database

The following code shows how to open a JDBC connection from a Matisse connection.

```
MtDatabase dbcon = new MtDatabase(hostname, dbname);

// Open a connection to the database
dbcon.open();
dbcon.startTransaction();

Connection jdbcon = dbcon.getJDBCConnection();
```

Executing a SQL Statement

After you open a connection to a Matisse database, you can execute statements (i.e., SQL statements or SQL methods) using a `Statement` object. You can create a statement object for a specific `MtDatabase` object using the `createStatement` method.

You can create more specific `Statement` objects for different purposes:

- `Statement` - It is specifically used for the SQL statements where you don't need to pass any value as a parameter

- `PreparedStatement` - It is a subclass of the statement class. The main difference is that, unlike the statement class, prepared statement is complied and optimized once and can be used multiple times by setting different parameter values.

- `CallableStatement`- It provides a way to call a stored procedure on the server from a Java™ program. Callable statements also need to be prepared first, and then their parameters are set using the set methods.

Retrieving Values

You use the `ResultSet` object, which is returned by the `executeQuery` method, to retrieve values or objects from the database. Use the `next` method combined with the appropriate `getString`, `getInt`, etc. methods to access each row in the result.

The following code demonstrates how to retrieve string and integer values from a `ResultSet` object after executing a `SELECT` statement.

```
MtDatabase dbcon = new MtDatabase(hostname, dbname);
// Open a connection to the database
dbcon.open();

try {
    // Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc' so there is
    // no need to use the full qualified names to acces the schema objects
    //dbcon.setSqlCurrentNamespace("examples.java_examples.jdbc");
    Connection jdbcon = dbcon.getJDBCConnection();
    // Create an instance of PreparedStatement
    String commandText = "SELECT FirstName, LastName, Spouse.FirstName AS Spouse, Age
FROM examples.java_examples.jdbc.Person WHERE LastName = ? LIMIT 10;";
    PreparedStatement pstmt = jdbcon.prepareStatement(commandText);
    pstmt.setEscapeProcessing(false);

    // Set parameters
    pstmt.setString(1, "Watson");


    // Execute the SELECT statement and get a ResultSet
    ResultSet rset = pstmt.executeQuery();

    System.out.println("Total selected:  " +
((MtResultSet)rset).getTotalNumObjects());
```

```
   System.out.println("Total qualified: " +
((MtResultSet)rset).getTotalNumQualified());
   System.out.println("");

   // Print column names
   ResultSetMetaData rsMetaData = rset.getMetaData();
   int numberOfColumns = rsMetaData.getColumnCount();

   // get the column names; column indexes start from 1
   for (int i = 0; i < numberOfColumns; i++) {
      System.out.print(String.format("%16s",rsMetaData.getColumnName(i+1)) + " ");
   }
   System.out.println("");
   for (int i = 0; i < numberOfColumns; ++i) {
      System.out.print("--------------- ");
   }
   System.out.println("");

   String fname, lname, sfname;
   Object age;
   // Read rows one by one
   while (rset.next()){
      // Get values for the first and second column
      fname = rset.getString(1);
      lname = rset.getString(2);
      sfname = rset.getString(3);
      age = rset.getInt(4);
      // The third column 'Age' can be null. Check if it is null or not first.
      if (rset.wasNull())
         age = "NULL";

      // Print the current row
      System.out.println(String.format("%16s",fname) + " " +
                     String.format("%16s",fname) + " " +
                     String.format("%16s",sfname) + " " +
                  age);
   }

   // Clean up and close the database connection
   rset.close();
   pstmt.close();
catch (SQLException e) {
   System.out.println("SQLException:  " + e.getMessage());
}
if (dbcon.isVersionAccessInProgress())
   dbcon.endVersionAccess();
else if (dbcon.isTransactionInProgress())
   dbcon.rollback();

dbcon.close();
```

Retrieving Objects from a SELECT statement

You can retrieve Java objects directly from the database without using the Object-Relational mapping technique. This method eliminates the unnecessary complexity in your application, i.e., O/R mapping layer, and improves your application performance and maintenance.

To retrieve objects, use `REF` in the select-list of the query statement and the `getObject` method returns an object. The following code example shows how to retrieve `Person` objects from a `ResultSet` object.

```
// Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc' so there is
// no need to use the full qualified names to acces the schema objects
//dbcon.setSqlCurrentNamespace("examples.java_examples.jdbc");
// Create an instance of Statement
Statement stmt = dbcon.createStatement();

// Set the SELECT statement. Note that we use REF() in the select-list
// to get the Java objects directly (rather than values)
String commandText = "SELECT REF(p) FROM examples.java_examples.jdbc.Person p WHERE
LastName = 'Watson';";

// Execute the SELECT statement and get a DataReader
ResultSet rset = stmt.executeQuery(commandText);

Person p;
// Read rows one by one
while (rset.next()) {
    // Get the Person object
    p = (Person)rset.getObject(1);
    // Print the current object with spouse's name
    // Note that spouse is directly accessed from the Person object 'p'
    System.out.println("Person: " +
                String.format("%16s",p.getFirstName()) +
                String.format("%16s",p.getLastName()) +
                " Spouse: " +
                String.format("%16s", p.getSpouse().getFirstName()));
    }

// Clean up and close the database connection
rset.close();
stmt.close();
```

Retrieving Objects from a Block Statement

You can also retrieve a collection of Java objects directly from the database by executing a SQL block statement.

The `getObject` method defined on a `MtCallableStatement` is used to return one object as well as an object collection. The following code example shows how to retrieve a collection of `Person` objects from a `MtCallableStatement`.

```
// Set a block statement
String commandText =
    "BEGIN\n" +
    "  DECLARE res SELECTION(Employee);\n" +
    "  DECLARE emp_sel SELECTION(Employee);\n" +
    "  DECLARE mgr_sel SELECTION(Manager);\n" +
    "  SELECT REF(p) FROM ONLY Employee p WHERE p.ReportsTo IS NULL INTO emp_sel;\n" +
    "  SELECT REF(p) FROM Manager p WHERE COUNT(p.Team) > 1 INTO mgr_sel;\n" +
    "  SET res = SELECTION(emp_sel UNION mgr_sel);\n" +
    "  RETURN res;\n" +
    "END";
```

```
// Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc' so there is
// no need to use the full qualified names to acces the schema objects
//dbcon.setSqlCurrentNamespace("examples.java_examples.jdbc");
Connection jdbcon = dbcon.getJDBCConnection();
// Create an instance of CallableStatement
CallableStatement stmt = jdbcon.prepareCall(commandText);
stmt.setEscapeProcessing(false);

//  Execute a block statement, and get the returned object selection
boolean isRset = stmt.execute();

Employee[] sel = (Employee[])stmt.getObject(0);

System.out.println("result Cnt: " + sel.length);

for (Employee e : sel) {
    System.out.println(e.getMtClass().getMtName() + ": " +
                       e.getFirstName() + " " +
                       e.getLastName() + " - Hiring Date: " +
                       String.format("%1$tY-%1$tm-%1$td", e.getHireDate()));
}
```

Executing DDL Statements

You can also create schema objects from a Java application via JDBC.

Creating a Class

You can create schema objects using the executeUpdate Method as long as the transaction is started in the DATA DEFINITION mode.

```
MtDatabase db = new MtDatabase(hostname, dbname);
// In order to execute DDL statements, the transaction needs to be
// started in the "Data Definition" mode
db.setOption (MtDatabase.DATA_ACCESS_MODE, MtDatabase.DATA_DEFINITION);
db.startTransaction();
// Execute the DDL statement
stmt = db.createStatement ();
stmt.executeUpdate ("CREATE CLASS Manager UNDER Employee (bonus INTEGER)");
db.commit();
db.close();
```

Creating a SQL Method

Creating a schema object using the execute Method does not require to start a transaction. A transaction will be automatically started in the DATA DEFINITION mode.

```
MtDatabase dbcon = new MtDatabase(hostname, dbname);
// Open a connection to the database. No need to start a transaction.
// execute() will start a transaction in the schema-definition mode automatically.
dbcon.open();
// Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc' so there is
// no need to use the full qualified names to acces the schema objects
dbcon.setSqlCurrentNamespace("examples.java_examples.jdbc");
```

```
// Create an instance of Statement
Statement stmt = dbcon.createStatement();
// The first method returns the number of Person objects which have a specified last
name
String commandText =
    "CREATE STATIC METHOD CountByLName(lname STRING)\n" +
    "RETURNS INTEGER\n" +
    "FOR Person\n" +
    "BEGIN\n" +
    "  DECLARE cnt INTEGER;\n" +
    "  SELECT COUNT(*) INTO cnt FROM Person WHERE LastName = lname;\n" +
    "  RETURN cnt;\n" +
    "END;";
System.out.println("creating...\n" + commandText);
stmt.execute(commandText);
// clean up
stmt.close();
// Commit the transaction and close the connection
dbcon.commit();
dbcon.close();
```

Executing SQL Methods

You can call a SQL method using the CALL syntax, i.e., simply passing the SQL method name followed by its arguments as an SQL statement. You can also use the CallableStatement object, which allows you to explicitly specify the method's parameters.

The following program code shows how to call the SQL method CountByLName of the Person class.

```
MtDatabase dbcon = new MtDatabase(hostname, dbname);

// Open a connection to the database.
dbcon.open();
dbcon.startVersionAccess();
// Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc' so there is
// no need to use the full qualified names to acces the schema objects
dbcon.setSqlCurrentNamespace("examples.java_examples.jdbc");
// Specify the stored method. Since it is a static method that we will call,
// the name is consisted of class name and method name.
String commandText = "CALL Person::CountByLName(?);";

Connection jdbcon = dbcon.getJDBCConnection();
// Create an instance of CallableStatement
CallableStatement stmt = jdbcon.prepareCall(commandText);
stmt.setEscapeProcessing(false);
// Set parameters
stmt.setString(1, "Watson");
//Execute the stored method
stmt.execute();
// Get the returned value
int count = (int)stmt.getInt(0);
// Print it
System.out.println(count + " objects found");
// clean up
stmt.close();
```

```
// close the connection
dbcon.endVersionAccess();
dbcon.close();
```

9 Working with SQL Methods

Running MethodCallExample

This example creates several objects, then manipulates the relationships among them in various ways as described in the source-code comments.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `sqlmethods` directory (under `java_examples`).

3. Load `SqlMethods.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `sqlmethods/SqlMethods.odl` for this demo.

4. Generate Java class files with the -psm options so Java methods mapping the SQL Methods defined on the class are generated as well:

   ```
   mt_sdl stubgen --lang java --psm -f SqlMethods.odl
   ```

5. Build the application:

   ```
   Windows:
   javac -classpath .;%MATISSE_HOME%\lib\matisse.jar
   examples\java_examples\sql_methods\*.java *.java

   UNIX:
   javac -classpath .;$MATISSE_HOME/lib/matisse.jar
   examples/java_examples/sql_methods/*.java *.java
   ```

6. Launch the application:

   ```
   java -classpath .;examples\java_examples\sql_methods;%MATISSE_HOME%\lib\matisse.jar
   MethodCallExample host database
   Windows:

   UNIX:
   java -classpath .:examples/java_examples/sql_methods:$MATISSE_HOME/lib/matisse.jar
   MethodCallExample host database
   ```

Executing a SQL Method

This example shows how to execute a SQL method mapped into its corresponding method in the Java stubclass. The Java stubclass provides a `GetFullName()` method which execute the `GetFullName()` SQL method defined on the `Member` class in the database schema defined as follows:

```
CREATE INSTANCE METHOD GetFullName()
RETURNS STRING
FOR \"Member\"
--
-- Return the person full name
--
```

```
BEGIN
  DECLARE fullName STRING;

  SET fullName = CONCAT(SELF.FirstName, ' ');
  IF SELF.MiddleName IS NOT NULL THEN
    SET fullName = CONCAT(fullName, SELF.MiddleName);
    SET fullName = CONCAT(fullName, ' ');
  END IF;
  SET fullName = CONCAT(fullName, SELF.LastName);

  RETURN fullName;
END;"
```

The `GetFullName()` SQL method is executed in your Java application as follows:

```
    int mid = 1;
    Member obj = Member.lookupMemberIdIdx(dbcon, mid);

    // "GetFullName" SQL Method call
    String res = obj.getFullName();
```

Executing a Static SQL Method

This example shows how to execute a Static SQL method mapped into its corresponding method in the Java stubclass. The Java stubclass provides a `GetPersonFullName()` static method which execute the `GetPersonFullName()` SQL method defined on the `Member` class in the database schema defined as follows:

```
CREATE STATIC METHOD GetPersonFullName(pid INTEGER)
RETURNS STRING
FOR \"Member\"
--
-- Return a person full name
--
BEGIN
  DECLARE fullName STRING;
  DECLARE vObj Member;

  SELECT REF(c) INTO vObj FROM Member c WHERE c.MemberId = pid;
  SET fullName = vObj.GetFullName();
  RETURN fullName;
END;"
```

The `GetPersonFullName()` SQL method is executed in your Java application as follows:

```
int mid = 2;
// "getPersonFullName" SQL Method call
String res = Member.getPersonFullName(dbcon, mid);
```

10 Working with Class Reflection

This section illustrates Matisse Reflection mechanism. This example shows how to manipulate persistent objects without having to create the corresponding Java stubclass. It also presents how to discover all the object properties.

Running ReflectionExample

This example creates several objects, then manipulates them to illustrate Matisse Reflection mechanism.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `sqlmethods` directory (under `java_examples`).

3. Load `examples.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `reflection/examples.odl` for this demo.

4. Build the application:

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

UNIX:
javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
```

5. Launch the application:

```
Windows:
java -classpath .;%MATISSE_HOME%\lib\matisse.jar ReflectionExample host database

UNIX:
java -classpath .;$MATISSE_HOME/lib/matisse.jar ReflectionExample host database
```

Creating Objects

This example shows how to create persistent objects without the corresponding Java stubclass. The static method `get()` defined on all Matisse Meta-Schema classes (i.e. `MtClass`, `MtAttribute`, etc.) allows you to access to the schema descriptor necessary to create objects. Each object is an instance of the `MtObject` base class. The `MtObject` class holds all the methods to update the object properties (attribute and relationships (i.e. `setString()`, `setSuccessors()`, etc.).

```
        MtDatabase db = new MtDatabase(hostname, dbname, new MtCoreObjectFactory());

        db.open();
        db.startTransaction();

        System.out.println("Creating one Person...");
        // Create a Person object
        MtClass pClass = MtClass.get(db,
"examples.java_examples.reflection.Person");
```

```
        MtAttribute fnAtt = MtAttribute.get(db, "FirstName", pClass);
        MtAttribute lnAtt = MtAttribute.get(db, "LastName", pClass);
        MtAttribute cgAtt = MtAttribute.get(db, "collegeGrad", pClass);
        MtObject p = new MtObject(pClass);
        p.setString(fnAtt, "John");
        p.setString(lnAtt, "Smith");
        p.setBoolean(cgAtt, false);

        System.out.println("Creating one Employee...");
        // Create a Employee object
        MtClass eClass = MtClass.get(db,
"examples.java_examples.reflection.Employee");
        MtAttribute hdAtt = MtAttribute.get(db, "hireDate", eClass);
        MtAttribute slAtt = MtAttribute.get(db, "salary", eClass);
        MtObject e = new MtObject(eClass);
        e.setString(fnAtt, "James");
        e.setString(lnAtt, "Roberts");
        e.setDate(hdAtt, new GregorianCalendar(2009, Calendar.JANUARY, 6));
        e.setNumeric(slAtt, new BigDecimal("5123.25"));
        e.setBoolean(cgAtt, true);

        System.out.println("Creating one Manager...");
        // Create a Manager object
        MtClass mClass = MtClass.get(db,
"examples.java_examples.reflection.Manager");
        MtRelationship tmRshp = MtRelationship.get(db, "team", mClass);
        MtObject m = new MtObject(mClass);
        m.setString(fnAtt, "Andy");
        m.setString(lnAtt, "Brown");
        m.setDate(hdAtt, new GregorianCalendar(2008, Calendar.NOVEMBER, 8));
        m.setNumeric(slAtt, new BigDecimal("7421.25"));
        m.setSuccessors(tmRshp, new MtObject[] {m, e});
        m.setBoolean(cgAtt, true);

        db.commit();
        db.close();
```

Listing Objects

This example shows how to list persistent objects without the corresponding Java stubclass. The
`instanceIterator()` method defined on the `MtClass` object allows you to access all instances defined
on the class.

```
    MtClass pClass = MtClass.get(db, "examples.java_examples.reflection.Person");
    MtAttribute fnAtt = MtAttribute.get(db, "FirstName", pClass);
    MtAttribute lnAtt = MtAttribute.get(db, "LastName", pClass);
    MtAttribute cgAtt = MtAttribute.get(db, "collegeGrad", pClass);

    // List all objects
    System.out.println("\n" + pClass.getInstanceNumber() +
                    " Person(s) in the database.");

    // Retrieve the object from the previous transaction
    MtObjectIterator<MtObject> iter = pClass.<MtObject>instanceIterator();
    while (iter.hasNext()) {
        MtObject p = iter.next();
```

```
        System.out.println("- " + p.getMtClass().getMtName() + " #" + p.getMtOid());

        System.out.println("  " + p.getString(fnAtt) + " " +
                            p.getString(lnAtt) +
                            " collegeGrad=" + p.getBoolean(cgAtt));
    }
    iter.close();
```

Working with Indexes

This example shows how to retrieve persistent objects from an index. The `MtIndex` class holds all the methods retrieves objects from an index key.

```
        MtClass pClass = MtClass.get(db,
"examples.java_examples.reflection.Person");
        MtAttribute fnAtt = MtAttribute.get(db, "FirstName", pClass);
        MtAttribute lnAtt = MtAttribute.get(db, "LastName", pClass);

        // Get the Index Descriptor object
        MtIndex iClass = MtIndex.get(db,
"examples.java_examples.reflection.personName");

        // Get the number of entries in the index
        long count = iClass.getIndexEntriesNumber();
        System.out.println(count + " entries in the index.");

       System.out.println("Looking for: " + firstName + " " + lastName);

        // lookup for the number of objects matching the key
        Object[] key = new Object[] { lastName, firstName } ;
        count = iClass.getObjectNumber(key, null);
        System.out.println(count + " matching objects to be retrieved.");

        if (count > 1) {
            // More than one matching object
            // Retrieve them with an iterator
            MtObjectIterator iter = iClass.iterator(key);
            while (iter.hasNext()) {
                MtObject p = iter.next();
                System.out.println("  found " + p.getString(fnAtt) + " " +
                                   p.getString(lnAtt));
            }
        } else {
            // At most 1 object
            // Retrieve the matching object with the lookup method
            MtObject p = iClass.lookup(key);
            if (p != null) {
                System.out.println("  found " + p.getString(fnAtt) + " " +
                                   p.getString(lnAtt));
            } else {
                System.out.println("  Nobody found");
            }
        }
```

Working with Entry Point Dictionaries

This example shows how to retrieve persistent objects from an Entry Point Dictionary. The `MtEntryPointDictionary` class holds the methods to retrieve objects from a string key.

```
  MtAttribute fnAtt = MtAttribute.get(db, "FirstName", pClass);
  MtAttribute lnAtt = MtAttribute.get(db, "LastName", pClass);
  MtAttribute cgAtt = MtAttribute.get(db, "collegeGrad", pClass);

  // Get the Index Descriptor object
  MtEntryPointDictionary epClass = MtEntryPointDictionary.get(db,
"examples.java_examples.reflection.collegeGradDict");

  System.out.println("Looking for Persons with CollegeGrad=" + collegeGrad);

  // lookup for the number of objects matching the key
  long count = epClass.getObjectNumber(collegeGrad, null);
  System.out.println(count + " matching objects to be retrieved.");

  if (count > 1) {
    // More than one matching object
    // Retrieve them with an iterator
    MtObjectIterator iter = epClass.iterator(collegeGrad);
    while (iter.hasNext()) {
      MtObject p = iter.next();
      System.out.println("  found " + p.getString(fnAtt) + " " +
                     p.getString(lnAtt) +
                     " collegeGrad=" + p.getBoolean(cgAtt));
    }
  } else {
    // At most 1 object
    // Retrieve the matching object with the lookup method
    MtObject p = epClass.lookup(collegeGrad);
    if (p != null) {
      System.out.println("  found " + p.getString(fnAtt) + " " +
                     p.getString(lnAtt) +
                     " collegeGrad=" + p.getBoolean(cgAtt));
    } else {
      System.out.println("  Nobody found");
    }
  }
```

Discovering Object Properties

This example shows how to list the properties directly from an object. The `MtObject` class holds the `attributesIterator()` method, `relationshipsIterator()` method and `inverseRelationshipsIterator()` method which enumerate the object properties.

```
  MtObjectIterator iter = pClass.instanceIterator();
  while (iter.hasNext()) {
    MtObject p = iter.next();

    System.out.println("- " + p.getMtClass().getMtName() + " #" + p.getMtOid());

    System.out.println(" Attributes:");
```

```
MtPropertyIterator<MtAttribute> propIter = p.attributesIterator();
MtAttribute a;
String propName;
int propType, valType;
String fmtVal;
while (propIter.hasNext()) {
  a = propIter.next();
  propName = a.getMtName();
  propType = a.getMtType();
  valType = p.getType(a);
  fmtVal = null;
  switch (valType) {
  case MtType.DATE:
    fmtVal = String.format("%1$tY-%1$tm-%1$td", p.getDate(a));
    break;
  case MtType.NUMERIC:
    fmtVal = p.getNumeric(a).toString();
    break;
  case MtType.NULL:
    fmtVal = null;
    break;
  default:
    fmtVal = p.getValue(a).toString();
  }
  System.out.println("\t" + propName +
                " (" + MtType.toString(propType) +
                "):\t" + fmtVal + " (" + MtType.toString(valType) + ")");
    }
propIter.close();

System.out.println(" Relationships:");
MtPropertyIterator<MtRelationship> rshpIter = p.relationshipsIterator();
MtRelationship r;
while (rshpIter.hasNext()) {
   r = rshpIter.next();
  System.out.println("\t" + r.getMtName() +
                ":\t" + p.getSuccessorSize(r) + " element(s)");
}
rshpIter.close();

System.out.println(" Inverse Relationships:");
rshpIter = p.inverseRelationshipsIterator();
while (rshpIter.hasNext()) {
   r = rshpIter.next();
  System.out.println("\t" + r.getMtName() +
                ":\t" + p.getSuccessorSize(r) + " element(s)");
}
rshpIter.close();
}
iter.close();
```

Adding Classes

This example shows how to add a new class to the database schema. The connection needs to be open in the DDL (`MtDatabase.DATA_DEFINITION`) mode. Then you need to create instances of `MtClass`, `MtAttribute` and `MtRelationship` and connect them together.

```
MtDatabase db = new MtDatabase(hostname, dbname, new MtCoreObjectFactory());

// open connection in DDL mode
db.setOption(MtDatabase.DATA_ACCESS_MODE, MtDatabase.DATA_DEFINITION);
db.open();

db.startTransaction();

MtClass pClass = MtClass.get(db, "examples.java_examples.reflection.Person");
System.out.println("Creating 'PostalAddress' class and linking it to 'Person'...");
System.out.println("in 'Person's namespace: " +
pClass.getMtNamespaceClassOf().getMtFullName());

// another way to get the namespace
MtNamespace ns = MtNamespace.get(db, "examples.java_examples.reflection");
if (ns == null) {
   // and to create a namespace path
   ns  =  MtNamespace.create(db, "examples.java_examples.reflection");
}

// Create a new Class

// Create attributes
MtAttribute cAtt = new MtAttribute(db, "City", MtType.STRING);
MtAttribute pcAtt = new MtAttribute(db, "PostalCode", MtType.STRING);


MtClass paClass = new MtClass(db, "PostalAddress", ns, new MtAttribute[] { cAtt, pcAtt },
null);

MtRelationship adRshp = new MtRelationship(db, "Address", paClass, new int[]{0, 1} );
pClass.addMtRelationship(adRshp);
```

Deleting Objects

This example shows how to delete persistent objects.The `MtObject` class holds `remove()` and
`deepRemove()`. Note that on `MtObject deepRemove()` does not execute any cascading delete but only
calls `remove()`.

```
          MtClass pClass = MtClass.get(db,
"examples.java_examples.reflection.Person");

          MtObjectIterator iter = pClass.instanceIterator();
          while (iter.hasNext()) {
               MtObject p = iter.next();

               p.deepRemove();
          }
          iter.close();
```

Removing Classes

This example shows how to remove a class for the database schema. The `deepRemove()` method defined on `MtClass` will delete the class and its properties and indexes. The connection needs to be open in `MtDatabase.DATA_DEFINITION` mode.

```
MtDatabase db = new MtDatabase(hostname, dbname, new MtCoreObjectFactory());

// open connection in DDL mode
db.setOption(MtDatabase.DATA_ACCESS_MODE, MtDatabase.DATA_DEFINITION);
db.open();

db.startTransaction();
MtClass paClass = MtClass.get(db,
"examples.java_examples.reflection.PostalAddress");
System.out.println("Removing " + paClass.getMtClass().getMtName() +
                   " " + paClass.getMtName() +
                   " (#" + paClass.getMtOid() + ")...");

paClass.deepRemove();

db.commit();
db.close();
```

11 Working with Database Events

This section illustrates Matisse Event Notification mechanism. The sample application is divided in two sections. The first section is event selection and notification. The second section is event registration and event handling.

Running EventsExample

This example creates several events, then manipulates them to illustrate the Event Notification mechanism.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `events` directory (under `java_examples`).

3. Build the application:

    ```
    Windows:
    javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

    UNIX:
    javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
    ```

4. Launch the application:
    To run the example, you need to open 2 command line windows and run one command in each windows.

    ```
    Windows:
    java -classpath .;%MATISSE_HOME%\lib\matisse.jar EventsExample host database N
    java -classpath .;%MATISSE_HOME%\lib\matisse.jar EventsExample host database S

    UNIX:
    java -classpath .:$MATISSE_HOME/lib/matisse.jar EventsExample host database N
    java -classpath .:$MATISSE_HOME/lib/matisse.jar EventsExample host database S
    ```

Events Subscription

This section illustrates event registration and event handling. Matisse provides the `MtEvent` class to manage database events. You can subscribe up to 32 events (`MtEvent.EVENT1 to MtEvent.EVENT32`) and then wait for the events to be triggered.

```
static int TEMPERATURE_CHANGES_EVT = MtEvent.EVENT1;
static int RAINFALL_CHANGES_EVT = MtEvent.EVENT2;
static int HIMIDITY_CHANGES_EVT = MtEvent.EVENT3;
static int WINDSPEED_CHANGES_EVT = MtEvent.EVENT4;

MtDatabase dbcon = new MtDatabase(hostname, dbname);
// Open the connection to the database
dbcon.open();
```

```
    // Create an Event subscriber
    MtEvent subscriber = new MtEvent(dbcon);

    // Subscribe to all 4 events
    long eventSet = TEMPERATURE_CHANGES_EVT |
                    RAINFALL_CHANGES_EVT |
                    HIMIDITY_CHANGES_EVT |
                    WINDSPEED_CHANGES_EVT;

    subscriber.subscribe(eventSet);

    long triggeredEvents;
    // Wait 1000 ms for the events to be triggered
    // return 0 if not event is triggered until the timeout is reached
    if ((triggeredEvents = subscriber.wait(1000)) != 0) {
      System.out.println("Events (#" + i + ") triggered:");
      System.out.println((((triggeredEvents & TEMPERATURE_CHANGES_EVT) > 0) ? "" : "No ")
                          + "Change in temperature");
      System.out.println((((triggeredEvents & RAINFALL_CHANGES_EVT) > 0) ? "" : "No ") +
                          "Change in rain fall");
      System.out.println((((triggeredEvents & HIMIDITY_CHANGES_EVT) > 0) ? "" : "No ") +
                          "Change in humidity");
      System.out.println((((triggeredEvents & WINDSPEED_CHANGES_EVT) > 0) ? "" : "No ") +
                          "Change in wind speed\n");
    } else {
        System.out.println("No Event received after 1 sec\n");
    }
    // Unsubscribe to all 4 events
    subscriber.unsubscribe();
```

Events Notification

This section illustrates event selection and notification.

```
    static int TEMPERATURE_CHANGES_EVT = MtEvent.EVENT1;
    static int RAINFALL_CHANGES_EVT = MtEvent.EVENT2;
    static int HIMIDITY_CHANGES_EVT = MtEvent.EVENT3;
    static int WINDSPEED_CHANGES_EVT = MtEvent.EVENT4;

    MtDatabase dbcon = new MtDatabase(hostname, dbname);
    // Open the connection to the database
    dbcon.open();
    // Create an Event notifier
    MtEvent notifier = new MtEvent(dbcon);

    long eventSet = 0;

    eventSet |= TEMPERATURE_CHANGES_EVT;
    eventSet |= WINDSPEED_CHANGES_EVT;
    // Notify of 2 events
    notifier.notify(eventSet);
```

More about MtEvent

As illustrated by the previous sections, the `MtEvent` class provides all the methods for managing database events. The reference documentation for the `MtEvent` class is included in the Matisse Java Binding API documentation located from the Matisse installation root directory in `docs/java/api/index.html`.

12 Handling Packages

The `mt_sdl` utility with the `stubgen` command allows you to generate Java source code for the schema classes defined in the ODL file. The `--sn <namespace>` and `--ln <namespace>` options define the mapping between the schema class namespace and the Java class package. When your persistent classes are defined in a specific package, you need to give this information to the `Connection` object so that it can find these classes when returning objects.

For example, to generate the Java classes in the root package from the schema classes in defined in the `java_examples.chap_3` namespace.

```
mt_sdl stubgen --lang java --sn java_examples.chap_3 -f examples.odl
```

To generate the Java classes in the `com.corp.myapp` package from the schema classes in defined in the root namespace.

```
mt_sdl stubgen --lang java --ln com.corp.myapp -f example.odl
```

To generate the Java classes in the `com.corp.myapp` package from the schema classes in defined in the `Examples.Sample03` namespace.

```
mt_sdl stubgen --lang java --sn Examples.Sample03 --ln com.corp.myapp -f example.odl
```

To generate the Java classes in the `com.corp.myapp` package from the schema classes in defined in the `com.corp.myapp` namespace.

```
mt_sdl stubgen --lang java -f example.odl
```

To generate the Java classes in the root package from the schema classes in defined in the root namespace.

```
mt_sdl stubgen --lang java -f example.odl
```

Connection with Factory

Using MtPackageObjectFactory

For example, the persistent classes are defined in the `com.company.project.module` Java package. In this case, you need to pass an `MtPackageObjectFactory` object as the additional argument for the `MtDatabase` constructor. Assuming that the schema classes are defined in the root namespace:

```
MtDatabase db = new MtDatabase("host", "db", new
MtPackageObjectFactory("com.company.project.module", ""));
```

Now assuming that the schema classes are defined in the `company.project.module.db01` namespace:

```
MtDatabase db = new MtDatabase("host", "db", new
MtPackageObjectFactory("com.company.project.module",
"company.project.module.db01"));
```

Now assuming that the schema classes are defined in the `com.company.project.module` namespace, which matches the Java package name, you can use the default factory:

```
MtDatabase db = new MtDatabase("host", "db");
```

If the persistent classes are in the multiple Java packages, you pass `HashMap` to the `MtPackageObjectFactory` constructor,

```
HashMap<String,String> pkgNsMap = new HashMap<String,String> ();
pkgNsMap.put("com.company.project.module1", "company.project.module.submoduleA");
pkgNsMap.put("com.company.project.module2", "company.project.module.submoduleB");
new MtPackageObjectFactory(pkgNsMap)
```

Using MtExplicitObjectFactory

The class is an alternative the class. The `mt_sdl` utility with the `stubgen` command generates the `<dbName>SchemaMap.txt` file that defines a direct class mapping between the Java classes and the schema classes.

```
MtDatabase db = new MtDatabase("host", "db", new
MtExplicitObjectFactory("examplesSchemaMap.txt"));
```

Using MtCoreObjectFactory

This factory is the basic MtObject-based object factory. This factory is the most appropriate for application which does use generated stubs. This factory is faster than the default Object Factory used by MtDatabase since it doesn't use reflection to build objects.

```
MtDatabase db = new MtDatabase("host", "db", new MtCoreObjectFactory());
```

Creating your Object Factory

Implementing the MtObjectFactory interface

The `MtObjectFactory` interface describes the mechanism used by `MtDatabase` to create the appropriate Java object for each Matisse object. Implementing the `MtObjectFactory` interface requires to define the `getJavaClass()` method which returns Java class corresponding to a Matisse Class Name, the `getDatabaseClass()` method which returns database class name corresponding to the Java class name and the `getObjectInstance()` method which returns a Java object based on an oid.

```
class MyAppFactory : MtObjectFactory
{
  public Class getJavaClass(String mtClsName) {
    return MtObject.class;
  }

  public object getObjectInstance(MtDatabase database, int oid)
  {
    return new MtObject(database, oid);
  }

  final static String NS_REFLECT = "com.matisse.reflect.";

  public String getDatabaseClass(String jClsName) {
    String res = jClsName;
    if (jClsName.startsWith(NS_REFLECT)) {
```

```
      res = jClsName.substring(NS_REFLECT.length());
    }
        return res;
  }

}
```

Implementing a Sub-Class of MtCoreObjectFactory

This `MtCoreObjectFactory` is a basic MtObject-based object factory which can be extended to implement your own Object Factory.

```
class MyAppFactory : MtCoreObjectFactory
{
  public Class getJavaClass(String mtClsName) {
      // Return a Java Class object as you see fit
    return aJavaClassObject;
  }

  public object getObjectInstance(MtDatabase database, int oid)
  {
    if (isSchemaObject(database, oid))
    {
      return super.getObjectInstance(database, oid);
    }
    else
    {
      // Create your Java object as you see fit
      return anObject;
    }
  }
}
```

13 Working with a Connection Pool

This section illustrates the implementation and usage of a database connection pool. A database connection pool is a straight forward solution to improve the overall performance of a multi-tier database driven application and to control the database resources allocated to the application.

Running MtDatabasePoolManagerExample

This example creates a MtDatabase pool manager, then use it a multi-threaded application to illustrate Matisse Connection pooling mechanism.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `pooling` directory (under `java_examples`).

3. Build the application:

    ```
    Windows:
    javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

    UNIX:
    javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
    ```

4. Launch the application:

    ```
    Windows:
    java -classpath .;%MATISSE_HOME%\lib\matisse.jar MtDatabasePoolManagerExample host
    database

    UNIX:
    java -classpath .;$MATISSE_HOME/lib/matisse.jar MtDatabasePoolManagerExample host
    database
    ```

MtDatabase Connection Pool

Implementing a MtDatabase Connection Pool Manager

The `MtDatabaseConnectionPoolManager` class manages a pool of connections to one Matisse database. It controls the maximum number of open connections included in the pool as well as the maximum time in seconds to wait for a free connection. The class provides basically two public methods, one to get a connection from the pool, the second to return it.

```
public class MtDatabasePoolManager {

  public MtDatabasePoolManager (String hostname, String database, String username,
                                String password, int maxConnections, int timeout) {
  }

  public MtDatabase getConnection() throws MtException {
  }
```

```
public synchronized void recycleConnection (MtDatabase conn) {
}

}
```

Get an Open Connection from the Pool

```
MtDatabasePoolManager poolMgr = new MtDatabasePoolManager(hostname, dbname,
                                            maxConnections, defaultTimeout);
// Get a connection from the Pool
MtDatabase conn = poolMgr.getConnection();
```

Return a Connection to the Pool

```
// Release the connection back to the Pool
poolMgr.recycleConnection(conn);
```

Running JdbcConnectionPoolManagerExample

This example creates a JDBC Connection pool manager, then use it a multi-threaded application to illustrate Matisse Connection pooling mechanism.

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Change to the `pooling` directory (under `java_examples`).

3. Build the application:

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

UNIX:
javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
```

4. Launch the application:

```
Windows:
java -classpath .;%MATISSE_HOME%\lib\matisse.jar JdbcConnectionPoolManagerExample
host database

UNIX:
java -classpath .;$MATISSE_HOME/lib/matisse.jar JdbcConnectionPoolManagerExample
host database
```

JDBC Connection Pool

Implementing a MtDatabase Connection Pool Manager

The `JDBCConnectionPoolManager` class manages a pool of connections to one Matisse database. It controls the maximum number of open connection included in the pool as well as the maximum time in seconds to wait for a free connection. The class provides one public method to get a connection from the pool. When the JDBC is closed a Java `ConnectionEventListener` is triggered and the connection is returned to the pool.

Get an Open Connection from the Pool

```
// Create a Connection Pool Data Source
MtConnectionPoolDataSource dataSource = new MtConnectionPoolDataSource();
dataSource.setDatasourceName ("DataSource_"+dbname);
dataSource.setDatabaseName (dbname);
dataSource.setServerName (hostname);

// Create a Connection Pool Manager for the Data Source
poolMgr = new JdbcConnectionPoolManager(dataSource, maxConnections, defaultTimeout);

// Get a connection from the Pool
Connection conn = poolMgr.getConnection();
```

Return a Connection to the Pool

```
Connection conn;

// Release the connection back to the Pool
conn.close();
```

14 Building your Application

This section describes the process for building an application from scratch with the Matisse Java binding.

Discovering the Matisse Java Classes

All the Java binding classes are provided in a single JAR file. The core classes defined in the **com.matisse** namespace. These classes manages the database connection, the object factories as well as the objects caching mechanisms. It also includes the Matisse meta-schema classes defined in the **com.matisse.reflect** namespace. It also includes the Matisse JDBC implementation defined in the **com.matisse.sql** namespace

Matisse Client Server

The Matisse Java binding is comprised of in 2 files:

1. **matisse.jar** contains all the Java binding classes.

2. **matisseJAVA** library contains the Java Native Interface (JNI) layer that links the binding to the **matisse** library.

Matisse Lite

Matisse Lite is the embedded version of Matisse DBMS. Matisse Lite is a compact library that implements the server-less version of Matisse. The Java binding also includes a Lite version of the binding. The Matisse Lite Java binding is comprised of in 2 files:

1. **matisselite.jar** contains all the Java binding classes.

2. **matisseliteJAVA** library contains the Java Native Interface (JNI) layer that links the binding to the **matisselite** library.

The Matisse Java API documentation included in the delivery provides a detailed description of all the classes and methods.

> **NOTE:** The Java binding API for Matisse Client Server and for Matisse Lite are totally identical making your application working with either one without any code changes.

Generating Stub Classes

The Java binding relies on object-to-object mapping to access objects from the database. Matisse `mt_sdl` utility allows you to generate the stub classes mapping your database schema classes. Generating Java stub classes is a 2 steps process:

1. Design a database schema using ODL (Object Definition Language).

2. Generate the Java classes from the ODL file:

```
mt_sdl stubgen --lang java -f myschema.odl
```

A .java file will be created for each class defined in the database. If you need to define these persistent classes in a specific namespace, use -n option. The following command generates classes under the namespace `com.company.project`:

```
mt_sdl stubgen --lang java --ln com.company.project -f myschema.odl
```

When you update your database schema later, load the updated schema into the database. Then, execute the `mt_sdl` utility in the directory where you first generated the class files, to update the files. Your own program codes added to these stub class files will be preserved.

Extending the generated Stub Classes

You can add your own source code outside of the `BEGIN` and `END` markers produced in the generated stub class.

```
// BEGIN Matisse SDL Generated Code
// DO NOT MODIFY UNTIL THE 'END of Matisse SDL Generated Code' MARK BELOW
...
// END of Matisse SDL Generated Code
```

Compiling the application

Matisse Client Server

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisse.jar *.java

UNIX:
javac -classpath .;$MATISSE_HOME/lib/matisse.jar *.java
```

Matisse Lite

```
Windows:
javac -classpath .;%MATISSE_HOME%\lib\matisselite.jar *.java

UNIX:
javac -classpath .;$MATISSE_HOME/lib/matisselite.jar *.java
```

Running the application

Matisse Client Server

```
Windows:
java -classpath .;%MATISSE_HOME%\lib\matisse.jar MyApp host database
```

```
UNIX:
java -classpath .;$MATISSE_HOME/lib/matisse.jar MyApp host database
```

Matisse Lite

```
Windows:
java -classpath .;%MATISSE_HOME%\lib\matisselite.jar MyApp host database

UNIX:
java -classpath .;$MATISSE_HOME/lib/matisselite.jar MyApp host database
```

Appendix A: Generated Public Methods

The following methods are generated automatically in the `.java` class files generated by `mt_sdl`.

For schema classes

The following methods are created for each schema class. These are class methods (also called static methods): that is, they apply to the class as a whole, not to individual instances of the class. These examples are taken from `Person`.

Count instances	getInstanceNumber(com.matisse.MtDatabase db) getOwnInstanceNumber(com.matisse.MtDatabase db)
Open an iterator	instanceIterator(com.matisse.MtDatabase db) ownInstanceIterator(com.matisse.MtDatabase db)
Sample constructor	`Person`(com.matisse.MtDatabase db)
Sample toString	toString()
Get descriptor	`getClass(com.matisse.MtDatabase db)`

Returns an `MtClass` object. This method supports advanced Matisse programming techniques such as dynamically modifying the schema.

Factory constructor	`Person`(com.matisse.MtDatabase db, int mtKey)

This constructor is called by `MtObjectFactory`. It is public for technical reasons but is not intended to be called directly by user methods.

For all attributes

The following methods are created for each attribute. For example, if the ODL definition for class `Check` contains the attributes `Date` and `Amount`, the `Check.java` file will contain the methods `getDate` and `getAmount`. These examples are taken from `Person.firstName`.

Get value	`getFirstName()`
Set value	`setFirstName`(java.lang.String val)
Remove value	`removeFirstName()`
Check Null value	`isFirstNameNull()`
Check Default value	`isFirstNameDefault()`
Get descriptor	`getFirstNameAttribute`(com.matisse.MtDatabase db)

Returns an `MtAttribute` object. This method supports advanced Matisse programming techniques such as dynamically modifying the schema.

For list-type attributes only

The following methods are created for each list-type attribute. These examples are from `Person.photo`.

Get elements	getPhotoElements(byte[] value, long offset, int len)
Set elements	setPhotoElements(byte[] value, long offset, int len, boolean discardAfter)
Count elements	getPhotoSize()

For all relationships

The following methods are created for each relationship. These examples are from `Person.spouse`.

Clear successors	clear`Spouse()`
Get descriptor	get`Spouse`Relationship(com.matisse.MtDatabase db)
	Returns an `MtRelationship` object. This method supports advanced Matisse programming techniques such as dynamically modifying the schema.

For relationships where the maximum cardinality is 1

The following methods are created for each relationship with a maximum cardinality of 1. These examples are from `Manager.assistant`.

Get successor	Employee getAssistant()
Set successor	`setAssistant(Employee succ)`

For relationships where the maximum cardinality is greater than 1

The following methods are created for each relationship with a maximum cardinality greater than 1. These examples are from `Manager.team`.

Get successors	Person[] getTeam()
Open an iterator	teamIterator()
Count successors	getteamSize()
Set successors	setTeam(Employee[] succs)
Add successors	Insert one successor before any existing successors: `prependTeam(Employee succ)`
	Add one successor after any existing successors: `appendTeamp(Employee succ)`
	Add multiple successors after any existing successors: `appendTeam(Employee[] succs)`
Remove successors	removeTeam(Employee succ)

removeTeam(Employee[] succs)

Remove specified successors.

For indexes

The following methods are created for every index defined for a database. These examples are for the only index defined in the example, `Person.personName`.

Lookup	`lookupPersonName`(com.matisse.MtDatabase db, java.lang.String lastName, java.lang.String firstName)
Open an iterator	personNameIterator(com.matisse.MtDatabase db, java.lang.String fromLastName, java.lang.String fromFirstName, java.lang.String toLastName, java.lang.String toFirstName) personNameIterator(com.matisse.MtDatabase db, java.lang.String fromLastName, java.lang.String fromFirstName, java.lang.String toLastName, java.lang.String toFirstName, com.matisse.reflect.MtClass filterClass, int direction, int numObjPerBuffer)
Get descriptor	getPersonNameIndex(com.matisse.MtDatabase db)

Returns an `MtIndex` object. This method supports advanced Matisse programming techniques such as dynamically modifying the schema.

For entry-point dictionaries

The following methods are created for every entry-point dictionary defined for a database. These examples are for the only dictionary defined in the example, `Person.commentDict`.

Lookup	lookupCommentDict(com.matisse.MtDatabase db, java.lang.String value)
Open an iterator	commentDictIterator(com.matisse.MtDatabase db, java.lang.String value) commentDictIterator(com.matisse.MtDatabase db, java.lang.String value, com.matisse.reflect.MtClass filterClass, int numObjPerBuffer)
Get descriptor	getCommentDictDictionary(com.matisse.MtDatabase db)

Returns an `MtEntryPointDictionary` object. This method supports advanced Matisse programming techniques such as dynamically modifying the schema.

Appendix B: Configuring Java IDEs

Notes Pertaining to All Java IDEs

To use the Matisse Java binding with a Java IDE you must configure the IDE to use the `matisse.jar` installed with Matisse.

The IDE-specific directions below assume that Matisse installation has been successfully completed and that a project has been started in the desired IDE. The instructions have been tested only with the particular versions of the IDEs identified here.

NetBeans IDE

Goal: mount a JAR filesystem for the `matisse.jar` file.

1. Select File / Mount Filesystem

2. Click "Add JAR file."

3. Click "browse."

4. Navigate to and select `matisse.jar`, then click "OK."

5. Click "OK."

Eclipse IDE

Goal: import the `matisse.jar` file.

1. Make sure `%MATISSEHOME%` is in your path.

2. Start VisualAge for Java.

3. Select your project.

4. Select File / Import.

5. Select "Jar File."

6. Click "Next."

7. Click "Browse."

8. Navigate to and select `matisse.jar`, then click "Open."

9. Click "Finish."

Appendix C: Additional Sample Applications

Overloaded Methods Optimize Storage in a Revision-Tracking System
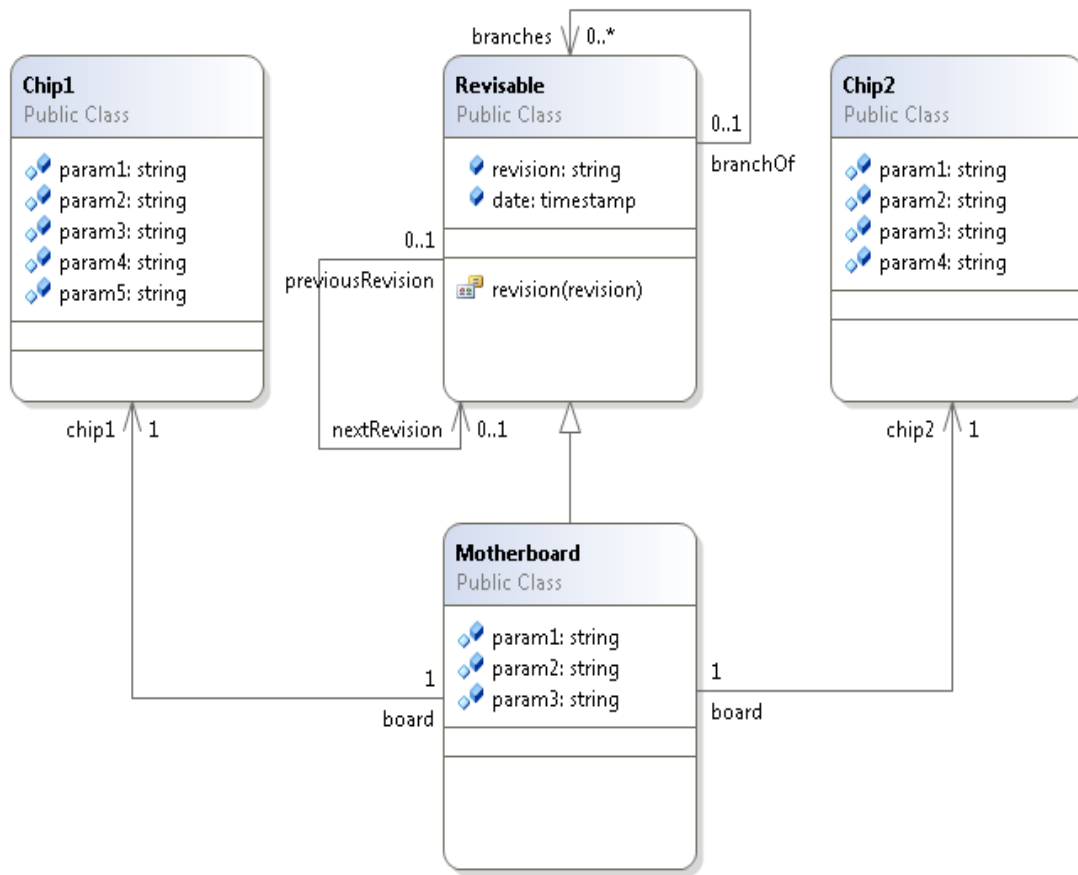
This example is a prototype for a real-world manufacturing application. The schema, as diagrammed above, reflects a motherboard with two chip components, each of which has several parameters. Relationships defined in the `Revisable` superclass track major ("dot-zero") branches and minor (.x, .x.x, etc.) revisions of the motherboard.

When you first run `Revision.java`, it creates a new `Motherboard` object with `revision` set to 1 and all the motherboard and chip parameters set to `null`. If you change a parameter using the application's `set` command, instead of modifying the existing object the application creates a new one, with `revision` incriminated to 2. If you change parameters for the `revision`=2 motherboard, the application will create branch version 2.1. After several such iterations, the database might contain several `Motherboard` objects, each with a different revision, related as shown at right.
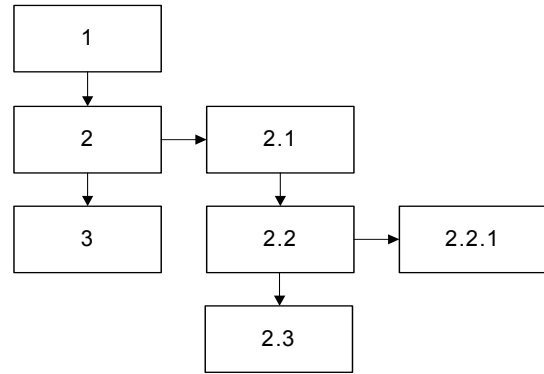
This application's overloaded methods reduce disk requirements by storing only the modified attributes for each object, and searching up the revision/branch hierarchy to find values for the other attributes.

To build and run this sample application:

1. Follow the instructions in *Before Running the Examples* on page 6.

2. Create and initialize a database as described in *Getting Started with Matisse*. If you wish, you may re-initialize a database created for another example.

3. Change to the `revision` directory (under `java_examples`).

4. Load `Revision.odl` into the database. From the Enterprise Manager, select the database 'example' and right click on 'Schema->Import ODL Schema', then select `revision/Revision.odl` for this demo.

5. Generate Java class files:

    ```
    mt_sdl stubgen --lang java -f Revision.odl
    ```

6. Build the application:

    ```
    Windows:
    javac -classpath .;%MATISSE_HOME%\lib\matisse.jar
    examples\java_examples\revision\*.java *.java

    UNIX:
    javac -classpath .;$MATISSE_HOME/lib/matisse.jar
    examples/java_examples/revision/*.java *.java
    ```

7. Run the application:

    ```
    Windows:
    java -classpath .;examples\java_examples\revision;%MATISSE_HOME%\lib\matisse.jar
    Revision database@host [set|dump] parameters

    UNIX:
    java -classpath .:examples/java_examples/revision:$MATISSE_HOME/lib/matisse.jar
    Revision database@host [set|dump] parameters
    ```

The parameters for the two commands are:

```
set revision object parameter value
dump revision
```

For example:

```
... set 1 Motherboard param1 42
... dump 1
... set 2 Motherboard param1 64
... set 3 Chip1 param2 984
... dump 1
... set 2 Chip2 param2 38974
... dump 2.1
... dump 1
```

Notes on `set`:

- The first time you use `set` on a database, specify revision 1.

- If no `nextRevision` successor exists for the specified revision, `Revision` creates one. For example, if you specify revision 2 and there is no 3, the new object will be 3; if you specify 2.1 and there is no 2.2, the new object will be 2.2.

- If a `nextRevision` successor already exists, `Revision` creates a new branch. For example, if you specify 2 and there is already a 3, the new object will be 2.1; if you specify 2.1 and there is already a 2.2, the new object will be 2.2.1.

Notes on `dump`:

The `dump` command lists all objects that are revision or branch successors to the specified revision. For example, if the revisions were as shown in the diagram on the previous page, `dump 2.2` would include 2.2, 2.2.1, and 2.3; `dump 2` would include 2, 2.1, 2.2, 2.2.1, 2.3, and 3.

JSP-Based Database Browser / Front End
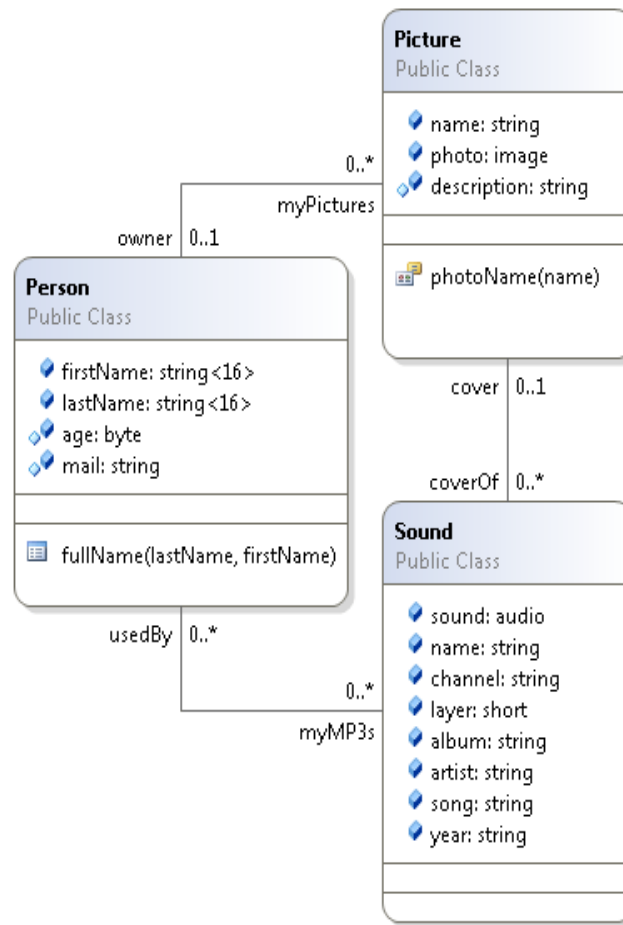
This JSP-based application is a generic Matisse database browser that you can use to explore or query any running Matisse database. When used with the included sample database (schema shown above), it also demonstrates how to build a Web-browser-based front end for a Matisse using JSP.

You can download the code from:

```
http://www.matisse.com/developers/documentation/
```

See the included `readme.txt` file for installation and operation instructions.