RDM Embedded 9.1

JNI Guide

Trademarks

Contents

General Introduction

The RDM Embedded database management system (DBMS) is designed to provide powerful, flexible, high-performance capabilities for developing C/C ++ or Java language database applications. The Java API on the RDM Embedded DBMS is based on Java Native Interface (JNI) technology.

By an extended C API to the Java programmer via the JNI, RDM Embedded lets you organize and access information efficiently, regardless of the complexity of your data.

This combined technology provides tremendous speed advantages, and minimizes data redundancy.

1.1 RDM Embedded Documentation

1.1.1 About This Manual

This RDM Embedded JNI Guide contains general information and examples on the use of the JNI system and is organized into the following sections.

Chapter 2 - Operational Overview

Gives an overview of each of the components of an RDM Embedded DBMS. It describes the general operational flow for creating an RDM Embedded application program in Java, and provides a simple (but complete) introductory example.

Chapter 3 - RDM Embedded Java Class and Package Definitions

Describes the API for accessing and processing data stored in a RDM Embedded database using the Java programming language.

Chapter 4 - Data Exchange and Representation Using ddlp

Describes the definition of the Java classes generated by the RDM Embedded DDL Processor.

Chapter 5 - RDM Embedded Core API Reference

Describes the mechanics for mapping the RDM Embedded Java API to the RDM Embedded Reference Manual.

Appendix A-1

Lists the Table Class contents

Appendix A-2

Lists the Database Class contents

Appendix B

Lists the functions not supported in the RDM Embedded Java API

Appendix C

Source code for the tims example program

1.1.2 Related RDM Embedded Documents

Additional comprehensive manuals are available as part of the RDM Embedded documentation set, including a User's Guide (operation details and code examples), a Reference Manual (utility and API descriptions), a Multi-User Guide, an SQL Guide, and a Platform-Specific Guide for both Unix and Windows.

Operational Overview

2.1 Introduction

This chapter presents an overview of the basic operation of the RDM Embedded DBMS. Each system component is identified and described, and the operational flow of an RDM Embedded Java program is given. A simple, introductory example is then developed illustrating the basic use of the system. Your understanding of what each component does and how it fits into the overall system is essential to your ability to use RDM Embedded effectively. (See the RDM Embedded User's Guide, Operational Overview, for additional details of RDM Embedded system components and system operation.)

2.2 System Components

The "Operational Flow" diagram shows all of the RDM Embedded system components. Arrows indicate input and output between the components and text files (for example, schema), specific file types in the database (for example, key files), or the database in general.

2.2.1 ddlp

The Database Definition Language Processor, ddlp, is a utility that compiles a DDL specification, called a schema, and produces the database dictionary. The dictionary contains database content and organization data that is used by the RDM Embedded runtime library functions (see below).

Java class files are created by the ddlp. These files contain constants and declarations associated with a specific database for use by the Java programs that access the database.

2.2.2 The Java Native Interface Shared Library

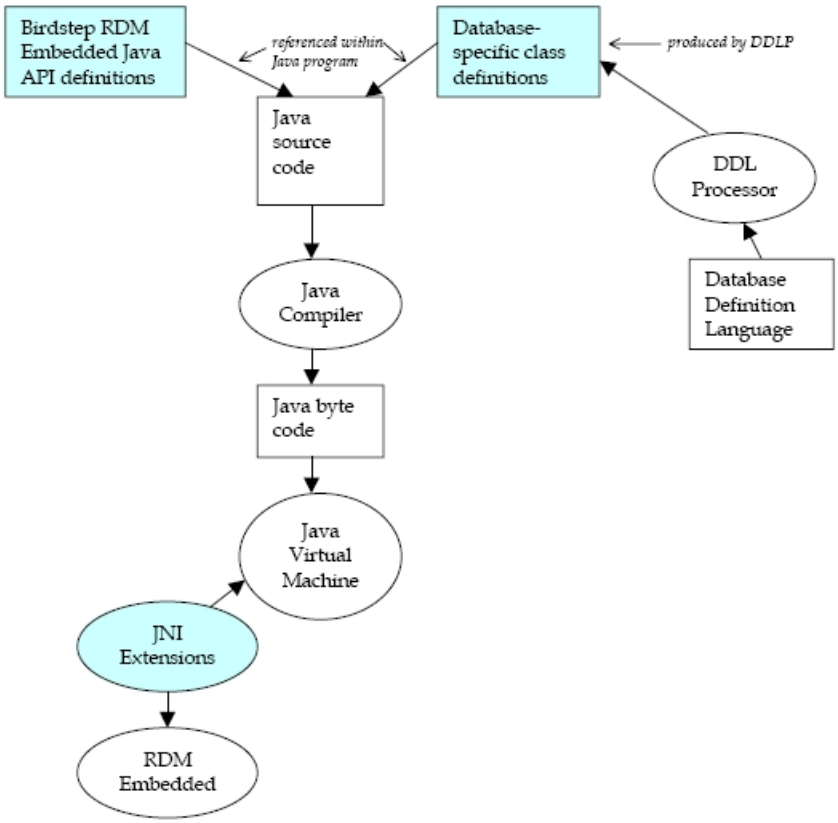
The RDM Embedded Runtime library is made accessible to Java code through the JNI via the Java native methods contained in the native library RDMEJni.

2.2.3 The Runtime Library

Database manipulation and control is performed through calls to C functions contained in the RDM Embedded runtime library. The library is linked with the object code for a C or C++ database application program to produce the executable program.

The runtime library functions contained in the library perform such operations as opening and closing the database, creating and modifying records and fields, connecting and disconnecting sets, searching for records through keys and sets, and controlling multi-user access through interaction with the system lock manager.

2.3 Operational Flow



2.4 JNI Environmental Setup

Extra steps are required in order to use the RDM Embedded JNI since it is not a pure Java client.

First, the RDM Embedded Native shared library and RDM Embedded runtime shared library must be added to the PATH environment variable. These libraries are installed to:

```
%RDME_HOME%/bin/%platform%
```

Secondly, you must place the rdm_embedded.jar file into the classpath.

2.5 Introductory Example

The following code briefly describes data retrieval using set navigation with the RDM Embedded Java API.

```
Task taskObj = new Task();
Database dbObj = taskObj.open("tims", "o");
AUTHOR name = new AUTHOR();
INFO irec = new INFO();
int stat;

// walk through the list of authors
for (stat = dbObj.findfm(tims.AUTHOR_LIST); stat == RDME.S_OKAY;
       stat = dbObj.findnm(tims.AUTHOR_LIST)) {

       stat = dbObj.recread(name);
       for (stat = dbObj.findfm(tims.HAS_PUBLISHED); stat == RDME.S_OKAY;
              stat = dbObj.findnm(tims.HAS_PUBLISHED)) {

              stat = dbObj.recread(irec);

              System.out.println("id_code : " + irec.getID_CODE());
              System.out.println("author : " + name.getNAME());
              System.out.println("title : " + irec.getINFO_TITLE());
              System.out.println("publc. : " + irec.getPUBLISHER() + ", " +
irec.getPUB_DATE());
       }
}
```

Note: The entire code for the Tims example is listed in Appendix C.

Java Class and Package Definitions

3.1 Package com.birdstep.rdme.jni

3.2 Task Class

The Task class collects the RDM Embedded functions into the group that requires a task parameter (DB_TASK*) but not a database number. They are the functions that aren't necessarily tied to a particular database, but rather the task (or session) in which one or more databases will be opened and transactions will be performed.

The Task class defines a number of public methods for use by the Java programmer, and a number of private native methods. These are the functions, which are actually implemented in C/C++. The native functions will receive parameters as specified in this class, plus two additional parameters that allow them to look up function addresses that interact with the Java runtime environment, including the instance of the Task object from which they are being invoked. The definition of the Task class is given in Appendix A.

3.3 Database Class

The Database class collects the RDM Embedded functions into the group that requires both a task parameter (DB_TASK*) and database number. The implementation of the java methods and the native methods follow the design of the Task class. The definition of the Database class is given in Appendix A.

3.4 DB_ADDR Class

The DB _ADDR class wraps the unique record slot within a database. The DB_ADDR class represents the structure of the database address as composed by a file and slot number. The file number is an unsigned value between zero and 254 and identifies the file where the record is stored. The slot number is an unsigned value between one and 16,777,214 (there is no slot zero).

Method Summary

```
void setAddr(short parameterFile, int parameterSlot)
```

Sets the database address

```
void setAddr(DB_ADDR parameterDB_ADDR)
```

Sets the database address

```
 void setFile(short parameterFile)
```

Sets the database file number

```
void setSlot(short parameterSlot)
```

Sets the database slot number

```
short getFile()
```

Retrieves the value of the database file number

```
int getSlot()
```

Retrieves the value of the database slot number

boolean isNull()

Retrieves whether the database address is null [0:0]

String toString()

Returns a String object representing this DB_ADDR's value.

3.5 LockRequest Class

The LockRequest class wraps the group lock request structure to the d_lock API to lock a group of record or set types.

Method Summary

```
void setItem(int parameterItem)
```

Sets the record or set type to be locked

```
   void setType(char parameterType)
```

Sets the type of lock to be applied as follows:

```
  'r' (read lock)
  'w' (write lock)
  'x' (exclusive lock)
  'k' (keep lock)
```

```
   int getItem()
```

Retrieves the value of the record or set type to be locked

```
   char getType()
```

Retrieves the value of the type of lock to be applied

3.6 LockType Class

The LockType class wraps a value of a primitive type char in an object. An object of type LockType contains a single field of type char. The LockType object is a parameter to d_reclstat, d_setlstat and d_keylstat to obtain the lock status for the type.

Method Summary

```
   void setLockType(char parameterLockType)
```

Sets the value of the char field

```
   char getLockType()
```

Retrieves char value of the object field

3.7 MirDiff Class

The MirDiff class wraps a value of a primitive type short in an object. An object of type MirDiff contains a single field of type short. The MirDiff object is a parameter to d_mirdbcompare method to compare the contents of two databases.

Method Summary

```
void setMirDiff(short parameterMirDiff)
```

Sets the value of the short field

```
short getMirDiff()
```

Retrieves short value of the object field

```
boolean dbIdentical()
```

Tests the comparison of the two databases

3.8 MirState Class

The MirState class wraps a value of a primitive type int in an object. An object of type MirState contains a single field of type int. The MirState object is a parameter to d_mirgetstate method to obtain the status of the mirroring system.

Field Summary

```
static MIR_PASSIVE
```

Mirroring is not launched

```
static MIR_SUSPENDED
```

Mirroring is suspended

```
    static MIR_ACTIVE
```

Mirroring is active

```
    static MIR_DISABLED
```

Mirroring is disabled

```
    static MIR_ERROR
```

Internal error, mirroring aborted

Method Summary

```
   void setMirState(int parameterMirDiff)
```

Sets the value of the int field

```
   int getMirState()
```

Retrieves int value of the object field

3.9 RDMERecType Class

The RDMERecType class wraps a value of a primitive type int in an object. An object of type RDMERecType contains a single field of type int. The RDMERecType object is a parameter to the d_cmtype, d_cotype and d_crtype to obtain the record type.

Method Summary

```
   void setRecordType(int parameterRDMERecType)
```

Sets the value of the integer field

```
   int getRecordType()
```

Retrieves integer value of the object field

3.10 Timestamp Class

The Timestamp class wraps a value of a primitive type int in an object. An object of type Timestamp contains a single field of type int. The Timestamp object is a parameter to the currency timestamp methods.

Method Summary

```
void setTimestamp(int parameterTimestamp)
```

Sets the value of the integer field

```
int getTimestamp()
```

Retrieves integer value of the object field

3.11 Package com.birdstep.rdm

3.12 Count Class

The Count class wraps a value of a primitive type int in an object. An object of type Count contains a single field of type int. The Count object is a parameter to d_recnum, d_setnum, d_fldnum and d_members.

Method Summary

```
void setCount(int parameterCount)
```

Sets the value of the integer field

```
int getCount()
```

Retrieves integer value of the object field

The UserStat class wraps a value of a primitive type int in an object. An object of type UserStat contains a single field of type int. The UserStat object is a parameter to d_lmstat method to obtain the status of a user from the user id.

Field Summary

```
static U_EMPTY
```

The user does not exist to the lock manager

```
static U_LIVE
```

The user is in a normal state

```
static U_DEAD
```

The user 'died' and has not been cleaned up

```
static U_BEGIN_REC
```

The user is being recovered

```
static U_RECOVERING
```

The user is recovering for another user

```
static U_REC_MYSELF
```

The user has re-entered and is recovering itself

```
static U_HOLDING_X
```

The user has left exclusive locks when closing

Method Summary

```
void setUserStat(int parameterUserStat)
```

Sets the value of the int field

```
int getUserStat()
```

Retrieves int value of the object field

Data Exchange and Representation Using ddlp

The ddlp utility includes a new command-line option to specify the output of the java database classes. This new -jni option will work together with existing options. Specifying this option will suppress the generation of a C header file. The -jni flag will optionally take a package name.

```
usage: ddlp [-jni [pkgname]][-d] [-n] [-r][-x] [-z] [-sql] [-s[-]] ddlfile
ddlp -jni moviedb.ddl
ddlp -jni com.mycompany.myproduct moviedb.ddl
```

The ddlp utility will create Java class files for the database, each record type and key field, and for record data fields, which are defined in the DDL as C structures. The Java class file created for the database contains the record, field, and set name constants.

To illustrate what is created by ddlp , assume that moviedb is used, and a portion of the DDL that defines movierec looks like this:

```
record movierec {
        long mid;
        char mname[31];
        compound unique key mkey {
                mid asc;
        }
        compound key mnkey {
                mname asc;
        }
}
```

The Java file created to represent this record type will look like this (and will be called movierec.java):

movierec.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;

public class movierec extends RDMTable
{
        public static final short ID = 0;
```

```
        // default constructor
        public movierec()
        {

                super(ID);
        // Allocate/Initialize data for mid field
        mid = 0;

        // Allocate/Initialize data for mname field
        mname = null;
}



// copy constructor
public movierec(movierec data) throws DBException
{
        super(ID);
        set(data);
}

// set method
public void set(movierec data) throws DBException
{
        this.setmid(data.getmid());
        this.setmname(data.getmname());
}



// getmid method
public int getmid() throws DBException
{
        return mid;
}

// setmid method
public void setmid(int mid) throws DBException
{
        this.mid = mid;
}
```

```
// getmname method
public String getmname() throws DBException
{
       return new String(mname);
}

// setmname method
public void setmname(String mname) throws DBException
{
       validateString(mname, 31);


       this.mname = new String(mname);
}

// private data members
       private int mid;
       private String mname;
}
```

Our convention for Java access to data in a record object is to create set and get methods to access the private data contained on the object. So for each field, a pair of methods will be defined by prefixing the field name with set and get. For arrayed fields additional set/get methods will be defined to access individual elements of the field array.

4.1 For Each Key Type

Also generated will be classes for holding the keys, movierecmkey.java and movierecmnkey.java:

movierecmkey.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;

public class movierecmkey extends RDMKey
{
       public static final int ID = 0;

       // default constructor
```

```
        public movierecmkey()
        {
                super(ID);

                // Allocate/initialize data for mid field
                mid = 0;
        }



        // copy constructor
        public movierecmkey(movierecmkey data) throws DBException
        {

                super(ID);
                set(data);
        }

        // set method
        public void set(movierecmkey data) throws DBException
        {
                setmid(getmid());
        }



        // getmid method
        public int getmid() throws DBException
        {
                return mid;
        }

        // setmid method
        public void setmid(int mid) throws DBException
        {
                this.mid = mid;
        }

        // private data members
        private int mid;
}
```

movierecmnkey.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;

public class movierecmnkey extends RDMKey
{

	public static final int ID = 1;

	// default constructor
	public movierecmnkey()
	{

		super(ID);



		// Allocate/initialize data for mname field
		mname = null;
	}

// copy constructor
public movierecmnkey(movierecmnkey data) throws DBException
{
	super(ID);
	set(data);
}

// set method
public void set(movierecmnkey data) throws DBException
{
	setmname(getmname());
}



// getmname method
public String getmname() throws DBException
{
	return new String(mname);
}

// setmname method
public void setmname(String mname) throws DBException
```

```
{
        validateString(mname, 31);

this.mname = new String(mname);
}

// private data members
private String mname;
}
```

4.2 For Each Structured Field

C structures have no equivalent in Java, so it is necessary to define Java classes that represent the structure data field. A Java class will be generated for each structure field with the naming convention of RECORD-NAMDFIELDNAMEStruct. The record will define get and set methods to access the structure class, but not the individual elements. Access to the structure elements will be handled through the structure class.

To illustrate, assume the following schema:

```
database structdb {

        data file "structdb.d01" contains rec1;

        record rec1 {
                int int_array[3][6][9];
                struct {
                        int fd1[5][7][10];
                        char fd2[30];
                } s[2][4][6];
        }
}
```

The Java classes generated by ddlp will consist of the database class (structdb.java), record class (REC1.java) and the struct data field class (REC1SStruct.java) as listed below.

structdb.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;
```

```
public class structdb
{
	// Record name constants
	public static final short REC1 = 0x0000;

	// Field name constants
	public static final int REC1_INT_ARRAY = 0x00000000;
	public static final int REC1_S = 0x00000001;
	public static final int REC1_FD1 = 0x00000002;
	public static final int REC1_FD2 = 0x00000003;
}
```

REC1.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;

public class rec1 extends RDMTable
{
	public static final short ID = 0;

	// default constructor
	public rec1()
	{
		super(ID);



		// Allocate/Initialize data for int_array field
		int_array = new int[3][][];
		for (int ii = 0; ii < 3; ii++) {
			int_array[ii] = new int[6][];
			for (int jj = 0; jj < 6; jj++) {
				int_array[ii][jj] = new int[9];
				for (int kk = 0; kk < 9; kk++) {
					int_array[ii][jj][kk] = 0;
				}
			}
		}



		// Allocate/Initialize data for s field
```

```
                s = new rec1sStruct[2][][];
                for (int ii = 0; ii < 2; ii++) {
                        s[ii] = new rec1sStruct[4][];
                        for (int jj = 0; jj < 4; jj++) {
                                s[ii][jj] = new rec1sStruct[6];
                                for (int kk = 0; kk < 6; kk++) {
                                        s[ii][jj][kk] = new rec1sStruct();
                                }
                        }
                }
        }



 // copy constructor
public rec1(rec1 data) throws DBException
{
        super(ID);
        set(data);
}

// set method
public void set(rec1 data) throws DBException
{
        this.setint_array(data.getint_array());
        this.sets(data.gets());
}

// getint_array method
public int[][][] getint_array() throws DBException
{
        int[][][] tempint_array;

        tempint_array = new int[3][][];
        for (int ii = 0; ii < 3; ii++) {
                tempint_array[ii] = new int[6][];
                for (int jj = 0; jj < 6; jj++) {
                        tempint_array[ii][jj] = new int[9];
                for (int kk = 0; kk < 9; kk++) {
                        tempint_array[ii][jj][kk] = int_array[ii][jj][kk];
                        }
                }
        }

        return tempint_array;
}

// getint_arrayElement method
```

```
        public int getint_arrayElement(int elem1, int elem2, int elem3) throws
DBException
{
        validateElement(elem1, 3);
        validateElement(elem2, 6);
        validateElement(elem3, 9);

        return int_array[elem1][elem2][elem3];
}

// setint_array method
public void setint_array(int[][][] int_array) throws DBException
{
        validateObject(int_array);

        int maxii = Math.min(3, int_array.length);
        for (int ii = 0; ii < maxii; ii++) {
                        validateObject(int_array[ii]);



                        int maxjj = Math.min(6, int_array[ii].length);
           for (int jj = 0; jj < maxjj; jj++) { validateObject(int_array[ii][jj]);
                        int maxkk = Math.min(9, int_array[ii][jj].length);
                        for (int kk = 0; kk < maxkk; kk++) {
                        this.int_array[ii][jj][kk] = int_array[ii][jj][kk];
                        }
                }
        }
}


        // setint_arrayElement method
        public void setint_arrayElement(int elem1, int elem2, int elem3, int
int_array) throws DBException
{
        validateElement(elem1, 3);
        validateElement(elem2, 6);
        validateElement(elem3, 9);
        this.int_array[elem1][elem2][elem3] = int_array;
}

        // gets method
        public rec1sStruct[][][] gets() throws DBException
{
        rec1sStruct[][][] temps;
```

```
        temps = new rec1sStruct[2][][];
        for (int ii = 0; ii < 2; ii++) {
                temps[ii] = new rec1sStruct[4][];
                for (int jj = 0; jj < 4; jj++) {
                        temps[ii][jj] = new rec1sStruct[6];
                        for (int kk = 0; kk < 6; kk++) {
                             temps[ii][jj][kk] = new rec1sStruct(s[ii][jj][kk]);
                }
        }
}
return temps;
}


 // getsElement method
public rec1sStruct getsElement(int elem1, int elem2, int elem3) throws
DBException
{
        validateElement(elem1, 2);
        validateElement(elem2, 4);
        validateElement(elem3, 6);

        return new rec1sStruct(s[elem1][elem2][elem3]);
}



// sets method
public void sets(rec1sStruct[][][] s) throws DBException
{
        validateObject(s);

        int maxii = Math.min(2, s.length);
        for (int ii = 0; ii < maxii; ii++) {
                validateObject(s[ii]);

        int maxjj = Math.min(4, s[ii].length);
        for (int jj = 0; jj < maxjj; jj++) {
                validateObject(s[ii][jj]);

                int maxkk = Math.min(6, s[ii][jj].length);
                for (int kk = 0; kk < maxkk; kk++) {
                        validateObject(s[ii][jj][kk]);

                        this.s[ii][jj][kk].setStruct(s[ii][jj][kk]);
                        }
```

```
                }
        }
}



        // setsElement method
        public void setsElement(int elem1, int elem2, int elem3, rec1sStruct s)
throws DBException
{
        validateElement(elem1, 2);
        validateElement(elem2, 4);
        validateElement(elem3, 6);
        validateObject(s);
        this.s[elem1][elem2][elem3].setStruct(s);
}

        // private data members
        private int[][][] int_array;
        private rec1sStruct[][][] s;
}
```

REC1SStruct.java

```
<optional package name>

import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;

public class rec1sStruct extends RDMObject
{

        // default constructor
        public rec1sStruct()
        {



                // Allocate/Initialize data for fd1 field
                fd1 = new int[5][][];
                for (int ii = 0; ii < 5; ii++) {
                        fd1[ii] = new int[7][];
                        for (int jj = 0; jj < 7; jj++) {
                                fd1[ii][jj] = new int[10];
```

```
                                for (int kk = 0; kk < 10; kk++) {
                                        fd1[ii][jj][kk] = 0;
                                }
                        }
                }



        // Allocate/Initialize data for fd2 field
        fd2 = null;
}

// copy constructor
public rec1sStruct(rec1sStruct data) throws DBException
{
        setStruct(data);
}




// setStruct method
public void setStruct(rec1sStruct data) throws DBException
{
        setfd1(data.getfd1());
        setfd2(data.getfd2());
}

// getfd1 method
public int[][][] getfd1() throws DBException
{
        int[][][] tempfd1;

        tempfd1 = new int[5][][];
        for (int ii = 0; ii < 5; ii++) {
                tempfd1[ii] = new int[7][];
                for (int jj = 0; jj < 7; jj++) {
                        tempfd1[ii][jj] = new int[10];
                        for (int kk = 0; kk < 10; kk++) {
                                tempfd1[ii][jj][kk] = fd1[ii][jj][kk];
                        }
                }
        }

        return tempfd1;
}

// getfd1Element method
```

```
public int getfd1Element(int elem1, int elem2, int elem3) throws
DBException
{
       validateElement(elem1, 5);
       validateElement(elem2, 7);
       validateElement(elem3, 10);

       return fd1[elem1][elem2][elem3];
}

// setfd1 method
public void setfd1(int[][][] fd1) throws DBException
{
               validateObject(fd1);

               int maxii = Math.min(5, fd1.length);
               for (int ii = 0; ii < maxii; ii++) {
                       validateObject(fd1[ii]);

               int maxjj = Math.min(7, fd1[ii].length);
               for (int jj = 0; jj < maxjj; jj++) {
                       validateObject(fd1[ii][jj]);

               int maxkk = Math.min(10, fd1[ii][jj].length);
               for (int kk = 0; kk < maxkk; kk++) {
                       this.fd1[ii][jj][kk] = fd1[ii][jj][kk];
                       }
               }
       }


 }



       // setfd1Element method
       public void setfd1Element(int elem1, int elem2, int elem3, int fd1)
       throws DBException
{
       validateElement(elem1, 5);
       validateElement(elem2, 7);
       validateElement(elem3, 10);
       this.fd1[elem1][elem2][elem3] = fd1;
}
// getfd2 method
public String getfd2() throws DBException
{
       return new String(fd2);
```

```
  }



  // setfd2 method
  public void setfd2(String fd2) throws DBException
  {
         validateString(fd2, 30);
         this.fd2 = new String(fd2);
  }

  // private data members
         private int[][][] fd1;
         private String fd2;
```

Core APL Reference

A major design goal for the RDM Embedded Java API was to maintain consistency with the Core API. Thus enabling existing users accustomed to the Core API the ease of using a familiar API in their Java applications. With the close similarities between the RDM Embedded Java API and the RDM Embedded Core API, the use of the Core API Reference materials can be used with minimal translation to the Java Programmer. A notable difference to highlight is in the naming of RDM Embedded Java methods. The public methods of the RDM Embedded Java API, do not contain the d_ prefix to the Core API function name. Additionally, functions in RDM Embedded Core API come in one of three forms: no handle, task handle only, or task handle and database number. Rather than create task handle and database number objects, we will encapsulate this information in Task and Database objects.

For example, a task will be created by instantiating a new object:

```
Task taskObj = new Task();
```

The new Task object, taskObj, contains methods that require a task parameter. One method is dbdpath, which requires only one parameter in Java, but two in C, because the task parameter is contained in the object:

```
taskObj.dbdpath("../dictdir");
```

Database objects (type Database) can be created in one of three ways:

```
// Through the Task object Task taskObj = new Task();
Database dbObj = taskObj.open("tims", "x");
```

or:

```
// Through the Database 'open' method Database dbObj = new Database(taskObj);
dbObj.open("tims", "x");
```

or:

```
// Through the Database constructor
Database dbObj = new Database(taskObj, "tims", "x");
```

The new Database object, dbObj, contains both the task handle and database number. Methods within the object are invoked without either parameter from Java, as shown below:

```
AUTHOR name  = new AUTHOR (); name.setNAME("Knuth, D.");
dbObj.fillnew(tims.AUTHOR, name);
```

Also shown in the example above is the usage of database-specific classes. In this case, AUTHOR is a class for which the definition was generated by the DDL processor.

Appendix

Appendix A Class Contents

5.1 Task Class contents

```
package com.birdstep.rdme.jni;

import com.birdstep.rdm.DBException;
import java.util.*;

public class Task {
	private boolean isValidTask;
	private int hTask;
	private int stat = 0;

	private native int d_opentask();
	private native int d_closetask(int hTask);

	private native int d_locktimeout(int read_secs, int write_secs, int
	hTask);
	private native int d_freeall(int hTask);
	private native int d_destroy(String dbname, int hTask);

	private native int d_lmclear(String user, String lockmgr,
	                             String lockcomm_type, int hTask);
	private native int d_lmstat(String user, UserStat stat, int hTask);
	private native int d_lockcomm(String lockcomm_type, int hTask);
	private native int d_lockmgr(String lockmgrname, int hTask);

	private native int d_timeout(int secs, int hTask);
	private native int d_trabort(int hTask);
	private native int d_trbegin(String tid, int hTask);
	private native int d_trend(int hTask);




	private native int d_mapchar(String inchar, String outchar, String
	sort_as,
	                                    String sub_sort, int hTask);
	private native int d_mircopymaintomirror(int hTask);
	private native int d_mircopymirrortomain(short sApplyPending, int hTask);
	private native int d_mircreate(String mainDbName, String mainDbPath,
	                               String mirDbName, String mirDbPath,
	                               short copy, int hTask);
```

```
        private native int d_mirdbcompare(MirDiff Identical, int hTask);
        private native int d_mirdelete(int hTask);
        private native int d_mirdisable(int hTask);
        private native int d_mirenable(int hTask);
        private native int d_mirexit(int hTask);
        private native int d_mirgetstate(MirState state, int hTask);
        private native int d_mirkill(int hTask);
        private native int d_mirlaunchsynchronous(int hTask);
        private native int d_mirlaunchthread(int hTask);
        private native int d_mirresume(int hTask);
        private native int d_mirsetpriority(int priority, int hTask);
        private native int d_mirsuspend(int hTask);

        private native int d_checkid(String id, int hTask);
        private native int d_ctbpath(String path, int hTask);
        private native int d_dbdpath(String path, int hTask);
        private native int d_dbfpath(String path, int hTask);
        private native int d_dbini(String dir_name, int hTask);
        private native int d_dblog(String log, int hTask);
        private native int d_dbnum(String dbname, int hTask);
        private native int d_dbtaf(String taf, int hTask);
        private native int d_dbtmp(String tmp, int hTask);
        private native int d_dbuserid(String id, int hTask);
        private static native int d_dbver(String fmt, StringBuffer buf, int len);
        private native int d_setpages(int dbpages, int ovflpages, int hTask);

        private native int d_def_opt(int optflag, int hTask);
        private native int d_on_opt(int optflag, int hTask);
        private native int d_off_opt(int optflag, int hTask);

        private native int d_renclean(int hTask);
        private native int d_setfiles(int num, int hTask);

        static {
                System.loadLibrary ("RDMEjni");
        }

        public Task() throws DBException
        {
                hTask = 0;
                isValidTask = false;
                validate(Opentask());
        }

        private int Opentask() throws DBException
        {
                int stat;
                if (isValidTask)
```

```
                        throw new DBException("Task has all ready been created!");

        stat = d_opentask();
        if (stat < 0)
                throw new DBException(stat);

        isValidTask = true;
        return stat;
}


public Database open(String dbnames, String omode) throws DBException
{
        Database db = new Database(this, dbnames, omode);
        return db;
}


public int closetask() throws DBException
{
        int stat;

        if (!isValidTask)
                return 0;

        isValidTask = false;
        stat = d_closetask(hTask);
        if (stat < 0)
                throw new DBException(stat);

        return stat;
}

public int locktimeout(int read_secs, int write_secs) throws DBException
{
        return validate(d_locktimeout(read_secs, write_secs, hTask));
}


public int freeall() throws DBException
{
        return validate(d_freeall(hTask));
}


        public int destroy(String dbname) throws DBException
{
        return validate(d_destroy(dbname, hTask));
```

```
}


public int lmclear(String user, String lockmgr, String lockcomm_type)
       throws DBException
{
       return validate(d_lmclear(user, lockmgr, lockcomm_type, hTask));
}

public int lmstat(String user, UserStat stat) throws DBException
{
       return validate(d_lmstat(user, stat, hTask));
}

public int lockcomm(String lockcomm_type) throws DBException
{
       return validate(d_lockcomm(lockcomm_type, hTask));
}

public int lockmgr(String lockmgrname) throws DBException
{
       return validate(d_lockmgr(lockmgrname, hTask));
}

public int timeout(int secs) throws DBException
{
       return validate(d_timeout(secs, hTask));
}

public int trabort() throws DBException
{
       return validate(d_trabort(hTask));
}

public int trbegin(String tid) throws DBException
{
       return validate(d_trbegin(tid, hTask));
}

public int trend() throws DBException
{
       return validate(d_trend(hTask));
}

public int mapchar(String inchar, String outchar, String sort_as, String
sub_sort)
       throws DBException
       {
```

```
                return validate(d_mapchar(inchar, outchar, sort_as, sub_sort,
hTask));
}

int mircopymaintomirror() throws DBException
{
       return validate(d_mircopymaintomirror(hTask));
}


public int copymirrortomain(short sApplyPending) throws DBException
{
       return validate(d_mircopymirrortomain(sApplyPending, hTask));
}


public int mircreate(String mainDbName, String mainDbPath, String mirDbName,
String mirDbPath, short copy)
       throws DBException
{
       return validate(d_mircreate(mainDbName, mainDbPath, mirDbName,
                      mirDbPath, copy, hTask));
}


public int mirdbcompare(MirDiff Identical) throws DBException
{
       return validate(d_mirdbcompare(Identical, hTask));
}



public int mirdelete() throws DBException
{
       return validate(d_mirdelete(hTask));
}


public int mirdisable() throws DBException
{
       return validate(d_mirdisable(hTask));
}


public int mirenable() throws DBException
{
       return validate(d_mirenable(hTask)); }
```

```
public int mirexit() throws DBException
{
       return validate(d_mirexit(hTask));
}


public int mirgetstate(MirState state) throws DBException
{
       return validate(d_mirgetstate(state, hTask));
}



public int mirkill() throws DBException
{
       return validate(d_mirkill(hTask));
}

public int mirlaunchsynchronous() throws DBException
{
       return validate(d_mirlaunchsynchronous(hTask));
}


public int mirlaunchthread() throws DBException
{
       return validate(d_mirlaunchthread(hTask));
}



public int mirresume() throws DBException
{
       return validate(d_mirresume(hTask));
}


public int mirsetpriority(int priority) throws DBException
{
       return validate(d_mirsetpriority(priority, hTask));
}



public int mirsuspend() throws DBException
{
       return validate(d_mirsuspend(hTask));
```

```
}


public int checkid(String id) throws DBException
{
       return validate(d_checkid(id, hTask));
}


public int ctbpath(String path) throws DBException
{
       return validate(d_ctbpath(path, hTask));
}


public int dbdpath(String path) throws DBException
{
       return validate(d_dbdpath(path, hTask));
}


public int dbfpath(String path) throws DBException
{
       return validate(d_dbfpath(path, hTask));
}


public int dbini(String dir_name) throws DBException
{
       return validate(d_dbini(dir_name, hTask));
}

public int dblog(String log) throws DBException
{
       return validate(d_dblog(log, hTask));
}


public int dbnum(String dbname) throws DBException
{
       return validate(d_dbnum(dbname, hTask));
}


public int dbtaf(String taf) throws DBException
{
       return validate(d_dbtaf(taf, hTask));
}
```

```
public int dbtmp(String tmp) throws DBException
{
       return validate(d_dbtmp(tmp, hTask));
}


public int dbuserid(String id) throws DBException
{
       return validate(d_dbuserid(id, hTask));
}


public static int dbver(String fmt, StringBuffer buf) throws DBException
{
       return staticValidate(d_dbver(fmt, buf, buf.capacity()));
}

public int setpages(int dbpages, int ovflpages) throws DBException
{
       return validate(d_setpages(dbpages, ovflpages, hTask));
}


public int def_opt(int optflag) throws DBException
{
       return validate(d_def_opt(optflag, hTask));
}


public int on_opt(int optflag) throws DBException
{
       return validate(d_on_opt(optflag, hTask));
}


public int off_opt(int optflag) throws DBException
{
       return validate(d_off_opt(optflag, hTask));
}


public int renclean() throws DBException
{
       return validate(d_renclean(hTask));
}
```

```
public int setfiles(int num) throws DBException
{
       return validate(d_setfiles(num, hTask));
}

public void setTask(int hTask) throws DBException
{
       if (hTask == 0)
              throw new DBException("Setting invalid Task!");

       this.hTask = hTask;
}

public int getTask() throws DBException
{
       if (!isValidTask)
              throw new DBException("Task has not been created!");

       return hTask; }



private static int staticValidate(int stat) throws DBException
{
       if (stat < 0)
       throw new DBException(stat);

       return stat;

}

private int validate(int stat) throws DBException
{
       if (stat < 0)
              throw new DBException(stat);

       return stat;
       }
}
```

5.2 Database Class contents

```
package com.birdstep.rdme.jni;
import com.birdstep.rdm.*;
public class Database {
public static final int CURR_DB = -1;
private Task hTask;
private boolean isOpen;
private int dbn;
private native int d_open(String dbnames, String omode, int hTask);
private native int d_iopen(String dbnames, int hTask);
private native int d_close(int hTask);
private native int d_closeall(int hTask);
private native int d_iclose(int hTask, int dbn);


 private native int d_cmstat(int set, int hTask, int dbn);
private native int d_cmtype(int set, RDMERecType rectype, int hTask, int
dbn);
private native int d_connect(int set, int hTask, int dbn);
private native int d_costat(int set, int hTask, int dbn);
private native int d_cotype(int set, RDMERecType rectype, int hTask, int
dbn);
private native int d_crget(DB_ADDR dba, int hTask, int dbn); /* DB_ADDR
class */
private native int d_crread_RDMObject(int field, RDMObject data, int
hTask, int dbn);
private native int d_crread_ByteArray(int field, byte[] data, int hTask,
int dbn);
private native int d_crset(DB_ADDR dba, int hTask, int dbn);
private native int d_crstat(int hTask, int dbn);
private native int d_crtype(RDMERecType rectype, int hTask, int dbn);
private native int d_crwrite_RDMObject(int field, RDMObject data, int
hTask, int dbn);
private native int d_crwrite_ByteArray(int field, byte[] data, int hTask,
int dbn);
private native int d_csmget(int set, DB_ADDR dba, int hTask, int dbn);
private native int d_csmread_RDMObject(int set, int field, RDMObject
data, int hTask, int dbn);
private native int d_csmread_ByteArray(int set, int field, byte[] data,
int hTask, int dbn);
private native int d_csmset(int set, DB_ADDR dba, int hTask, int dbn);
private native int d_csmwrite_RDMObject(int set, int field, RDMObject
data, int hTask, int dbn);
private native int d_csmwrite_ByteArray(int set, int field, byte[] data,
int hTask, int dbn);
```

```
private native int d_csoget(int set, DB_ADDR dba, int hTask, int dbn);
private native int d_csoread_RDMObject(int set, int field, RDMObject
data, int hTask, int dbn);
private native int d_csoread_ByteArray(int set, int field, byte[] data,
int hTask, int dbn);
private native int d_csoset(int set, DB_ADDR dba, int hTask, int dbn);
private native int d_csowrite_RDMObject(int set, int field, RDMObject
data, int hTask, int dbn);
private native int d_csowrite_ByteArray(int set, int field, byte[] data,
int hTask, int dbn);
private native int d_csstat(int set, int hTask, int dbn);
private native int d_ctscm(int set, Timestamp ts, int hTask, int dbn);
private native int d_ctsco(int set, Timestamp ts, int hTask, int dbn);
private native int d_ctscr(Timestamp ts, int hTask, int dbn);
private native int d_curkey(int hTask, int dbn);
private native int d_dbnum(String dbname, int hTask, int dbn);
private native int d_delete(int hTask, int dbn);
private native int d_discon(int set, int hTask, int dbn);
private native int d_disdel(int hTask, int dbn);
private native int d_fillnew(int rec, RDMTable data, int hTask, int dbn);
private native int d_findco(int set, int hTask, int dbn);
private native int d_findfm(int set, int hTask, int dbn);
private native int d_findlm(int set, int hTask, int dbn);
private native int d_findnm(int set, int hTask, int dbn);
private native int d_findpm(int set, int hTask, int dbn);
private native int d_fldnum(Count index, int field, int hTask, int dbn);

 private native int d_gtscm(int set, Timestamp ts, int hTask, int dbn);
private native int d_gtsco(int set, Timestamp ts, int hTask, int dbn);
private native int d_gtscr(Timestamp ts, int hTask, int dbn);
private native int d_gtscs(int set, Timestamp ts, int hTask, int dbn);
private native int d_ismember(int set, int hTask, int dbn);
private native int d_isowner(int set, int hTask, int dbn);
private native int d_keydel(int field, int hTask, int dbn);
private native int d_keyexist(int field, int hTask, int dbn);
private native int d_keyfind(int field, RDMKey data, int hTask, int dbn);
private native int d_keyfrst(int field, int hTask, int dbn);
private native int d_keylast(int field, int hTask, int dbn);
private native int d_keynext(int field, int hTask, int dbn);
private native int d_keyprev(int field, int hTask, int dbn);
private native int d_keyread(RDMKey data, int hTask);
private native int d_keystore(int field, int hTask, int dbn);
private native int d_makenew(int rec, int hTask, int dbn);
private native int d_members(int set, Count count, int hTask, int dbn);
private native int d_rdcurr(DB_ADDR dba, int currsize, int hTask, int
dbn);
private native int d_recfrst(int rec, int hTask, int dbn);
private native int d_recnext(int hTask, int dbn);
```

```
  private native int d_recnum(Count index, int rec, int hTask, int dbn);
  private native int d_recprev(int hTask, int dbn);
  private native int d_recread(RDMTable data, int hTask, int dbn);
  private native int d_recset(int rec, int hTask, int dbn);
  private native int d_recstat(RDMTable data, int rts, int hTask, int dbn);
  private native int d_recwrite(RDMTable data, int hTask, int dbn);
  private native int d_rerdcurr(DB_ADDR dba, int hTask, int dbn);
  private native int d_setdb(int db, int hTask);
  private native int d_setkey(int field, RDMKey data, int hTask, int dbn);
  private native int d_setmm(int to_set, int from_set, int hTask, int dbn);
  private native int d_setmo(int member_set, int owner_set, int hTask, int
  dbn);
  private native int d_setmr(int set, int hTask, int dbn);
  private native int d_setnum(Count index, int set, int hTask, int dbn);
  private native int d_setom(int owner_set, int member_set, int hTask, int
  dbn);
  private native int d_setoo(int to_set, int from_set, int hTask, int dbn);
  private native int d_setor(int set, int hTask, int dbn);
  private native int d_setrm(int set, int hTask, int dbn);
  private native int d_setro(int set, int hTask, int dbn);
  private native int d_stscm(int set, Timestamp ts, int hTask, int dbn);
  private native int d_stsco(int set, Timestamp ts, int hTask, int dbn);
  private native int d_stscr(Timestamp ts, int hTask, int dbn);
  private native int d_stscs(int set, Timestamp ts, int hTask, int dbn);
  private native int d_utscm(int set, Timestamp ts, int hTask, int dbn);
  private native int d_utsco(int set, Timestamp ts, int hTask, int dbn);
  private native int d_utscr(Timestamp ts, int hTask, int dbn);
  private native int d_utscs(int set, Timestamp ts, int hTask, int dbn);
  private native int d_wrcurr(DB_ADDR dba, int hTask, int dbn);
  private native int d_lock(int count, LockRequest[] lrp, int hTask, int
  dbn);
  private native int d_initialize(int hTask, int dbn);
  private native int d_initfile(short fileid, int hTask, int dbn);
  private native int d_keybuild(int hTask, int dbn);
  private native int d_keyfree(int field, int hTask, int dbn);

   private native int d_keylock(int field, String type, int hTask, int dbn);
  private native int d_keylstat(int field, LockType type, int hTask, int
  dbn);
  private native int d_recfree(int rec, int hTask, int dbn);
  private native int d_reclock(int rec, String type, int hTask, int dbn);
  private native int d_reclstat(int rec, LockType type, int hTask, int
  dbn);
  private native int d_rlbclr(int hTask, int dbn);
  private native int d_rlbset(int hTask, int dbn);
  private native int d_rlbtst(int hTask, int dbn);
  private native int d_setfree(int set, int hTask, int dbn);
  private native int d_setlock(int set, String type, int hTask, int dbn);
```

```
private native int d_setlstat(int set, LockType type, int hTask, int
dbn);


 static {
System.loadLibrary("RDMEjni");
}
public Database(Task task) throws DBException
{
this.dbn = 0;
this.isOpen = false;
this.hTask = task;
}
public Database(Task task, String dbnames, String omode) throws
DBException
{
int stat;
this.hTask = task;
stat = open(dbnames, omode);
if (stat != 0)
throw new DBException(stat);
}
public int open(String dbnames, String omode) throws DBException
{
return validate(d_open(dbnames, omode, hTask.getTask()));
}
public int iopen(String dbnames) throws DBException
{
return validate(d_iopen(dbnames, hTask.getTask()));
}
public int close() throws DBException
{
return validate(d_close(hTask.getTask()));
}
public int iclose(int dbn) throws DBException
{
return validate(d_iclose(dbn, hTask.getTask()));
}
public int closeall() throws DBException
{
return validate(d_closeall(hTask.getTask()));
}
public int cmstat(int set) throws DBException
{
return validate(d_cmstat(set, hTask.getTask(), this.dbn));
}
```

```
 public int cmtype(int set, RDMERecType rectype) throws DBException
{
return validate(d_cmtype(set, rectype, hTask.getTask(), this.dbn));
}
public int connect(int set) throws DBException
{
return validate(d_connect(set, hTask.getTask(), this.dbn));
}
public int costat(int set) throws DBException
{
return validate(d_costat(set, hTask.getTask(), this.dbn));
}
public int cotype(int set, RDMERecType rectype) throws DBException
{
return validate(d_cotype(set, rectype, hTask.getTask(), this.dbn));
}
public int crget(DB_ADDR dba) throws DBException
{
return validate(d_crget(dba, hTask.getTask(), this.dbn));
}
public int crread(int field, RDMTable data) throws DBException
{
return validate(d_crread_RDMObject(field, data, hTask.getTask(),
this.dbn));
}
public int crread(int field, byte[] data) throws DBException
{
return validate(d_crread_ByteArray(field, data, hTask.getTask(),
this.dbn));
}
public int crset(DB_ADDR dba) throws DBException
{
return validate(d_crset(dba, hTask.getTask(), this.dbn));
}
public int crstat() throws DBException
{
return validate(d_crstat(hTask.getTask(), this.dbn));
}
public int crtype(RDMERecType rectype) throws DBException
{
return validate(d_crtype(rectype, hTask.getTask(), this.dbn));
}
public int crwrite(int field, RDMObject data) throws DBException
{
return validate(d_crwrite_RDMObject(field, data, hTask.getTask(),
this.dbn));

 }
```

```
public int crwrite(int field, byte[] data) throws DBException
{
return validate(d_crwrite_ByteArray(field, data, hTask.getTask(),
this.dbn));
}
public int csmget(int set, DB_ADDR dba) throws DBException
{
return validate(d_csmget(set, dba, hTask.getTask(), this.dbn));
}
public int csmread(int set, int field, RDMObject data) throws DBException
{
return validate(d_csmread_RDMObject(set, field, data,
hTask.getTask(), this.dbn));
}
public int csmread(int set, int field, byte[] data) throws DBException
{
return validate(d_csmread_ByteArray(set, field, data,
hTask.getTask(), this.dbn));
}
public int csmset(int set, DB_ADDR dba) throws DBException
{
return validate(d_csmset(set, dba, hTask.getTask(), this.dbn));
}
public int csmwrite(int set, int field, RDMObject data) throws
DBException
{
return validate(d_csmwrite_RDMObject(set, field, data,
hTask.getTask(), this.dbn));
}
public int csmwrite(int set, int field, byte[] data) throws DBException
{
return validate(d_csmwrite_ByteArray(set, field, data,
hTask.getTask(), this.dbn));
}
public int csoget(int set, DB_ADDR dba) throws DBException
{
return validate(d_csoget(set, dba, hTask.getTask(), this.dbn));
}
public int csoread(int set, int field, RDMObject data) throws DBException
{
return validate(d_csoread_RDMObject(set, field, data,
hTask.getTask(), this.dbn));
}
public int csoread(int set, int field, byte[] data) throws DBException
{
return validate(d_csoread_ByteArray(set, field, data,

hTask.getTask(), this.dbn));
```

```
}
public int csoset(int set, DB_ADDR dba) throws DBException
{
return validate(d_csoset(set, dba, hTask.getTask(), this.dbn));
}
public int csowrite(int set, int field, RDMObject data) throws
DBException
{
return validate(d_csowrite_RDMObject(set, field, data,
hTask.getTask(), this.dbn));
}
public int csowrite(int set, int field, byte[] data) throws DBException
{
return validate(d_csowrite_ByteArray(set, field, data,
hTask.getTask(), this.dbn));
}
public int csstat(int set) throws DBException
{
return validate(d_csstat(set, hTask.getTask(), this.dbn));
}
public int ctscm(int set, Timestamp ts) throws DBException
{
return validate(d_ctscm(set, ts, hTask.getTask(), this.dbn));
}
public int ctsco(int set, Timestamp ts) throws DBException
{
return validate(d_ctsco(set, ts, hTask.getTask(), this.dbn));
}
public int ctscr(Timestamp ts) throws DBException
{
return validate(d_ctscr(ts, hTask.getTask(), this.dbn));
}
public int curkey() throws DBException
{
return validate(d_curkey(hTask.getTask(), this.dbn));
}
public int dbnum(String dbname) throws DBException
{
return validate(d_dbnum(dbname, hTask.getTask(), this.dbn));
}
public int delete() throws DBException
{
return validate(d_delete(hTask.getTask(), this.dbn));
}
public int discon(int set) throws DBException


 {
```

```
return validate(d_discon(set, hTask.getTask(), this.dbn));
}
public int disdel() throws DBException
{
return validate(d_disdel(hTask.getTask(), this.dbn));
}
public int fillnew(int rec, RDMTable data) throws DBException
{
return validate(d_fillnew(rec, data, hTask.getTask(), this.dbn));
}
public int findco(int set) throws DBException
{
return validate(d_findco(set, hTask.getTask(), this.dbn));
}
public int findfm(int set) throws DBException
{
return validate(d_findfm(set, hTask.getTask(), this.dbn));
}
public int findlm(int set) throws DBException
{
return validate(d_findlm(set, hTask.getTask(), this.dbn));
}
public int findnm(int set) throws DBException
{
return validate(d_findnm(set, hTask.getTask(), this.dbn));
}
public int findpm(int set) throws DBException
{
return validate(d_findpm(set, hTask.getTask(), this.dbn));
}
public int fldnum(Count index, int field) throws DBException
{
return validate(d_fldnum(index, field, hTask.getTask(), this.dbn));
}
public int gtscm(int set, Timestamp ts) throws DBException
{
return validate(d_gtscm(set, ts, hTask.getTask(), this.dbn));
}
public int gtsco(int set, Timestamp ts) throws DBException
{
return validate(d_gtsco(set, ts, hTask.getTask(), this.dbn));
}
public int gtscr(Timestamp ts) throws DBException
{
return validate(d_gtscr(ts, hTask.getTask(), this.dbn));

 }
public int gtscs(int set, Timestamp ts) throws DBException
```

```
{
return validate(d_gtscs(set, ts, hTask.getTask(), this.dbn));
}
public int ismember(int set) throws DBException
{
return validate(d_ismember(set, hTask.getTask(), this.dbn));
}
public int isowner(int set) throws DBException
{
return validate(d_isowner(set, hTask.getTask(), this.dbn));
}
public int keydel(int field) throws DBException
{
return validate(d_keydel(field, hTask.getTask(), this.dbn));
}
public int keyexist(int field) throws DBException
{
return validate(d_keyexist(field, hTask.getTask(), this.dbn));
}
public int keyfind(int field, RDMKey data) throws DBException
{
return validate(d_keyfind(field, data, hTask.getTask(), this.dbn));
}
public int keyfrst(int field) throws DBException
{
return validate(d_keyfrst(field, hTask.getTask(), this.dbn));
}
public int keylast(int field) throws DBException
{
return validate(d_keylast(field, hTask.getTask(), this.dbn));
}
public int keynext(int field) throws DBException
{
return validate(d_keynext(field, hTask.getTask(), this.dbn));
}
public int keyprev(int field) throws DBException
{
return validate(d_keyprev(field, hTask.getTask(), this.dbn));
}
public int keyread(RDMKey data) throws DBException
{
return validate(d_keyread(data, hTask.getTask()));
}


 public int keystore(int field) throws DBException
{
return validate(d_keystore(field, hTask.getTask(), this.dbn));
```

```
}
public int makenew(int rec) throws DBException
{
return validate(d_makenew(rec, hTask.getTask(), this.dbn));
}
public int members(int set, Count count) throws DBException
{
return validate(d_members(set, count, hTask.getTask(), this.dbn));
}
public int rdcurr(DB_ADDR dba, int currsize) throws DBException
{
return validate(d_rdcurr(dba, currsize, hTask.getTask(), this.dbn));
}
public int recfrst(int rec) throws DBException
{
return validate(d_recfrst(rec, hTask.getTask(), this.dbn));
}
public int recnext() throws DBException
{
return validate(d_recnext(hTask.getTask(), this.dbn));
}
public int recnum(Count index, int rec) throws DBException
{
return validate(d_recnum(index, rec, hTask.getTask(), this.dbn));
}
public int recprev() throws DBException
{
return validate(d_recprev(hTask.getTask(), this.dbn));
}
public int recread(RDMTable data) throws DBException
{
return validate(d_recread(data, hTask.getTask(), this.dbn));
}
public int recset(int rec) throws DBException
{
return validate(d_recset(rec, hTask.getTask(), this.dbn));
}
public int recstat(RDMTable data, int rts) throws DBException
{
return validate(d_recstat(data, rts, hTask.getTask(), this.dbn));
}
public int recwrite(RDMTable data) throws DBException
{

 return validate(d_recwrite(data, hTask.getTask(), this.dbn));
}
public int rerdcurr(DB_ADDR dba) throws DBException
{
```

```
return validate(d_rerdcurr(dba, hTask.getTask(), this.dbn));
}
public int setdb(int db) throws DBException
{
if (db < 0)
throw new DBException("Setting invalid database");
this.dbn = db;
return validate(d_setdb(db, hTask.getTask()));
}
public int setkey(int field, RDMKey data) throws DBException
{
return validate(d_setkey(field, data, hTask.getTask(), this.dbn));
}
public int setmm(int to_set, int from_set) throws DBException
{
return validate(d_setmm(to_set, from_set, hTask.getTask(),
this.dbn));
}
public int setmo(int member_set, int owner_set) throws DBException
{
return validate(d_setmo(member_set, owner_set, hTask.getTask(),
this.dbn));
}
public int setmr(int set) throws DBException
{
return validate(d_setmr(set, hTask.getTask(), this.dbn));
}
public int setnum(Count index, int set) throws DBException
{
return validate(d_setnum(index, set, hTask.getTask(), this.dbn));
}
public int setom(int owner_set, int member_set) throws DBException
{
return validate(d_setom(owner_set, member_set, hTask.getTask(),
this.dbn));
}
public int setoo(int to_set, int from_set) throws DBException
{
return validate(d_setoo(to_set, from_set, hTask.getTask(),
this.dbn));
}



 public int setor(int set) throws DBException
{
return validate(d_setor(set, hTask.getTask(), this.dbn));
}
```

```
public int setrm(int set) throws DBException
{
return validate(d_setrm(set, hTask.getTask(), this.dbn));
}
public int setro(int set) throws DBException
{
return validate(d_setro(set, hTask.getTask(), this.dbn));
}
public int stscm(int set, Timestamp ts) throws DBException
{
return validate(d_stscm(set, ts, hTask.getTask(), this.dbn));
}
public int stsco(int set, Timestamp ts) throws DBException
{
return validate(d_stsco(set, ts, hTask.getTask(), this.dbn));
}
public int stscr(Timestamp ts) throws DBException
{
return validate(d_stscr(ts, hTask.getTask(), this.dbn));
}
public int stscs(int set, Timestamp ts) throws DBException
{
return validate(d_stscs(set, ts, hTask.getTask(), this.dbn));
}
public int utscm(int set, Timestamp ts) throws DBException
{
return validate(d_utscm(set, ts, hTask.getTask(), this.dbn));
}
public int utsco(int set, Timestamp ts) throws DBException
{
return validate(d_utsco(set, ts, hTask.getTask(), this.dbn));
}
public int utscr(Timestamp ts) throws DBException
{
return validate(d_utscr(ts, hTask.getTask(), this.dbn));
}
public int utscs(int set, Timestamp ts) throws DBException
{
return validate(d_utscs(set, ts, hTask.getTask(), this.dbn));
}
public int wrcurr(DB_ADDR dba) throws DBException
{


 return validate(d_wrcurr(dba, hTask.getTask(), this.dbn));
}
public int lock(int count, LockRequest[] lrp) throws DBException
{
```

```
return validate(d_lock(count, lrp, hTask.getTask(), this.dbn));
}


 public int initialize() throws DBException
{
return validate(d_initialize(hTask.getTask(), this.dbn));
}
public int initfile(short fileid) throws DBException
{
return validate(d_initfile(fileid, hTask.getTask(), this.dbn));
}
public int keybuild(int dbn) throws DBException
{
return validate(d_keybuild(dbn, hTask.getTask()));
}
public int keyfree(int field) throws DBException
{
return validate(d_keyfree(field, hTask.getTask(), this.dbn));
}
public int keylock(int field, String type) throws DBException
{
return validate(d_keylock(field, type, hTask.getTask(), this.dbn));
}
public int keylstat(int field, LockType type) throws DBException
{
return validate(d_keylstat(field, type, hTask.getTask(), this.dbn));
}
public int recfree(int rec) throws DBException
{
return validate(d_recfree(rec, hTask.getTask(), this.dbn));
}
public int reclock(int rec, String type) throws DBException
{
return validate(d_reclock(rec, type, hTask.getTask(), this.dbn));
}
public int reclstat(int rec, LockType type) throws DBException
{
return validate(d_reclstat(rec, type, hTask.getTask(), this.dbn));
}
public int rlbclr() throws DBException
{
return validate(d_rlbclr(hTask.getTask(), this.dbn));
}
public int rlbset() throws DBException
{
return validate(d_rlbset(hTask.getTask(), this.dbn));
}
```

```
public int rlbtst() throws DBException


 {
return validate(d_rlbtst(hTask.getTask(), this.dbn));
}
public int setfree(int set) throws DBException
{
return validate(d_setfree(set, hTask.getTask(), this.dbn));
}
public int setlock(int set, String type) throws DBException
{
return validate(d_setlock(set, type, hTask.getTask(), this.dbn));
}
public int setlstat(int set, LockType type) throws DBException
{
return validate(d_setlstat(set, type, hTask.getTask(), this.dbn));
}
public void setdbn(int db)
{
this.dbn = db;
}
public int getdbn()
{
return this.dbn;
}
private int validate(int stat) throws DBException
{
if (stat < 0)
    throw new DBException(stat);
return stat;
}
}
```

Function not supported in the RDM Embedded Java API

- d_dbstat
- d_internals
- SQLAllocHandle
- SQLBindCol
- SQLBindParameter
- SQLCloseCursor
- SQLConnect

SQLDescribeCol

SQLDescribeParam

SQLDisconnect

SQLEndTran

SQLExecute

SQLExecDirect

SQLFetch

SQLFreeHandle

SQLGetCursorName

SQLGetDiagField

SQLGetDiagRec

SQLNumParams

SQLNumResultCols

SQLPrepare

SQLRowCount

SQLSetCursorName

```
import com.birdstep.rdme.jni.*;
import com.birdstep.rdm.*;
import java.io.*;
import examples.jnitims.*;
class Prompt
{
BufferedReader br;


 Prompt()
{
br = new BufferedReader(new InputStreamReader(System.in));
}
String getStr(String prompt)
{
String line;
System.out.print(prompt);
```

```
 try {
line = br.readLine();
}
catch (IOException e) {
System.out.println("IOException");
return null;
}
if (line.length() == 0)
return null;
return line;
}
}


class TimsRun
{
Task taskObj;
Database dbObj;
Prompt p;
public TimsRun(Prompt p) throws DBException
{
this.p = p;
try {
taskObj = new Task();
}
catch (DBException e) {
throw e;
}
dbObj = null;
try {
dbObj = taskObj.open("tims", "o");
}
catch (DBException e) {
System.out.println("Failed to open tims database: " +
e.getErrorNumber());
taskObj.closetask();
throw e;
}
}
void displayCommands()
{
System.out.println("");
System.out.println("JNITIMS Commands:");
System.out.println(" 1 - Display list of key words");
System.out.println(" 2 - Display list of authors");
System.out.println(" 3 - List publications by key word");
System.out.println(" 4 - List publications by author");
System.out.println(" 5 - Enter technical information");
```

```
System.out.println(" 6 - Delete technical information");
System.out.println(" 7 - Loan book");
System.out.println(" 8 - Return loaned book");
System.out.println(" 9 - List borrowed books");
System.out.println(" q - Quit");
}
void listKeys() throws DBException
{
int stat;
boolean all = false;
boolean partial = false;
String key;
KEY_WORDKWORD key_word = new KEY_WORDKWORD();
key = p.getStr("start key: ");
if (key != null) {

 key_word.setKWORD(key);
if ((stat = dbObj.keyfind(tims.KWORD, key_word)) ==
RDME.S_NOTFOUND) {
stat = dbObj.keynext(tims.KWORD);
}
}
else
stat = dbObj.keyfrst(tims.KWORD);


 /* scan thru keys */
while (stat == 0) {
dbObj.keyread(key_word);
System.out.println(" " + key_word.getKWORD());
stat = dbObj.keynext(tims.KWORD);
}
p.getStr("--- press <enter> to continue");
}
void listAuthors() throws DBException
{
int stat;
String start;
AUTHOR name = new AUTHOR();


 start = p.getStr("start name: ");
if (start != null) {
/* scan for first name */
for(stat = dbObj.findfm(tims.AUTHOR_LIST); stat == RDME.S_OKAY;
stat = dbObj.findnm(tims.AUTHOR_LIST)) {
stat = dbObj.recread(name);
if (start.compareTo(name.getNAME()) <= 0)
```

```
break;
}
}
else
stat = dbObj.findfm(tims.AUTHOR_LIST);


 while (stat == 0) {
stat = dbObj.recread(name);
System.out.println(" " + name.getNAME());
stat = dbObj.findnm(tims.AUTHOR_LIST);
}
p.getStr("--- press <enter> to continue");
}


 void pr_keywords() throws DBException
{
Count count = new Count();
KEY_WORD keyword;
DB_ADDR dba = new DB_ADDR();
/* the current member of the has_published set is the info record
whose
key word are to be listed */
dbObj.setom(tims.INFO_TO_KEY, tims.HAS_PUBLISHED);
/* fetch the number of members of info_to_key */
dbObj.members(tims.INFO_TO_KEY, count);
/* list the key words, if any */
if (count.getCount() > 0) {


 /* save current member of key_to_info because it's going to
change and
we may be currently scanning through that set */
keyword = new KEY_WORD();
//dba = new DB_ADDR();
dbObj.csmget(tims.KEY_TO_INFO, dba);


 System.out.println("key words:");
System.out.println("----------");
/* find each intersect member record */
while (dbObj.findnm(tims.INFO_TO_KEY) == RDME.S_OKAY) {
/* find, read and print corresponding key_word */
dbObj.findco(tims.KEY_TO_INFO);
dbObj.recread(keyword);
System.out.println(" " + keyword.getKWORD());
}
```

```
}


 System.out.println("");
/* reset key_to_info current member and owner */
if (!dba.isNull())
dbObj.csmset(tims.KEY_TO_INFO, dba);
}
void pr_abstract() throws DBException
{
Count count = new Count(); /* number of abstract members */
INFOTEXT text;
/* the current member of has_published is the info record whose
abstract
is to be printed */
dbObj.setom(tims.ABSTRACT, tims.HAS_PUBLISHED);
/* fetch number of lines in abstract */
dbObj.members(tims.ABSTRACT, count);


 /* print abstract if one exists */
if (count.getCount() > 0) {
text = new INFOTEXT();
System.out.println("abstract:");
System.out.println("---------");
/* find, read and print each abstract text line */
while (dbObj.findnm(tims.ABSTRACT) != RDME.S_EOS) {
dbObj.recread(text);
System.out.println(" " + text.getLINE());
}
}
System.out.println("");
}
void byKey() throws DBException
{
int stat;
String key;
AUTHOR name = new AUTHOR(); /* author's name */
INFO irec = new INFO(); /* key word */
KEY_WORDKWORD key_word = new KEY_WORDKWORD();
key = p.getStr("key word: ");
key_word.setKWORD(key);
if (dbObj.keyfind(tims.KWORD, key_word) == RDME.S_NOTFOUND) {
System.out.println("no records found");
}
else {
/* scan thru key_to_info set */
stat = dbObj.setor(tims.KEY_TO_INFO);
```

```
for (stat = dbObj.findfm(tims.KEY_TO_INFO); stat == RDME.S_OKAY;
stat = dbObj.findnm(tims.KEY_TO_INFO)) {
/* find current owner (info) of current record (intersect) */
stat = dbObj.findco(tims.INFO_TO_KEY);
/* read contents of info record */
stat = dbObj.recread(irec);
/* find author of info record */
stat = dbObj.findco(tims.HAS_PUBLISHED);
stat = dbObj.recread(name);
/* output results */
System.out.println("id_code: " + irec.getID_CODE());
System.out.println("author : " + name.getNAME());
System.out.println("title : " + irec.getINFO_TYPE());
System.out.println("publ. : " + irec.getPUBLISHER() + ", " +
irec.getPUB_DATE());
pr_keywords();
pr_abstract();
p.getStr("--- press <enter> to continue");
}
}

 }


 void byAuthor() throws DBException
{
int stat;
String authorname;
INFO irec;
AUTHOR name;
/* find author record */
authorname = p.getStr("author: ");
if (authorname == null)
return;
irec = new INFO();
name = new AUTHOR();
for (stat = dbObj.findfm(tims.AUTHOR_LIST); stat == RDME.S_OKAY;
stat = dbObj.findnm(tims.AUTHOR_LIST)) {
stat = dbObj.recread(name);
if (authorname.compareTo(name.getNAME()) == 0) {
stat = dbObj.setor(tims.HAS_PUBLISHED);
for (stat = dbObj.findfm(tims.HAS_PUBLISHED); stat ==
RDME.S_OKAY;
stat = dbObj.findnm(tims.HAS_PUBLISHED)) {
stat = dbObj.recread(irec);
/* read and print info record */
System.out.println("id_code: " + irec.getID_CODE());
System.out.println("author : " + name.getNAME());
```

```
System.out.println("title : " + irec.getINFO_TITLE());
System.out.println("publc. : " + irec.getPUBLISHER() + ",
" + irec.getPUB_DATE());
pr_keywords();
pr_abstract();
p.getStr("--- press <enter> to continue");
}


 }
else if (authorname.compareTo(name.getNAME()) < 0) {
System.out.println("author record not found");
}
}
}


 boolean get_info(INFO irec, AUTHOR author) throws DBException
{
short type;
String str;
if ((str = p.getStr("author : ")) == null)
return false;
author.setNAME(str);
for ( ; ; ) {
irec.setID_CODE(p.getStr("id_code : "));
irec.setINFO_TITLE(p.getStr("title : "));
irec.setPUBLISHER(p.getStr("publisher: "));
irec.setPUB_DATE(p.getStr("pub. date: "));
for ( ; ; ) {
str = p.getStr("info type: ");
type = Short.parseShort(str);
if (type >= 0 && type <= 2)
break;
System.out.println("invalid info type - correct types are:");
System.out.println("0 - book, 1 - magazine, 2 = article");
}
irec.setINFO_TYPE(type);
str = p.getStr("enter data (y/n)? ");
if (str.charAt(0) == 'y' || str.charAt(0) == 'Y')
return true;
}
}


 void enter_key_words(short info_type) throws DBException
{
String str;
```

```
INTERSECT isect = new INTERSECT();
KEY_WORD kw = new KEY_WORD();
KEY_WORDKWORD word = new KEY_WORDKWORD();


 isect.setINT_TYPE(info_type);
while ((str = p.getStr("key word: ")) != null) {
word.setKWORD(str);
/* see if key word record exists */
if (dbObj.keyfind(tims.KWORD, word) == RDME.S_NOTFOUND) {
kw.setKWORD(word.getKWORD());
/* create new key word record */
dbObj.fillnew(tims.KEY_WORD, kw);
}
dbObj.setor(tims.KEY_TO_INFO);
/* create intersection record */
dbObj.fillnew(tims.INTERSECT, isect);
dbObj.connect(tims.KEY_TO_INFO);
dbObj.connect(tims.INFO_TO_KEY);
}
}


 void enter_abstract() throws DBException
{
String str;
INFOTEXT text = new INFOTEXT();
while ((str = p.getStr("abstract: ")) != null) {
text.setLINE(str);
dbObj.fillnew(tims.INFOTEXT, text);
dbObj.connect(tims.ABSTRACT);
}
}


 void enterInfo() throws DBException
{
int stat;
String str;
INFO irec = new INFO();
AUTHOR name = new AUTHOR();



 while (get_info(irec, name)) {
str = name.getNAME();
/* see if author exists */
for (stat = dbObj.findfm(tims.AUTHOR_LIST); stat == RDME.S_OKAY;
```

```
stat = dbObj.findnm(tims.AUTHOR_LIST)) {
dbObj.recread(name);
if (str.compareTo(name.getNAME()) == 0)
break;
}
if (stat == 1) {
name.setNAME(str);
/* author not on file -- create record and connect to author
list */
dbObj.fillnew(tims.AUTHOR, name);
dbObj.connect(tims.AUTHOR_LIST);
}
/* make author current owner of has_published set */
dbObj.setor(tims.HAS_PUBLISHED);


 /* create new tech. info record */
if (dbObj.fillnew(tims.INFO, irec) == RDME.S_NOTFOUND)
System.out.println("duplicate id_code: " +
irec.getID_CODE());
else {
/* connect to author record */
dbObj.connect(tims.HAS_PUBLISHED);
/* set current owner for key words and abstract */
dbObj.setor(tims.INFO_TO_KEY);
dbObj.setor(tims.ABSTRACT);
enter_key_words(irec.getINFO_TYPE());
enter_abstract();
}
}
}

 void deleteInfo() throws DBException
{
int stat;
String str;
Count count;
INFO irec;
AUTHOR name;
INFOID_CODE id = new INFOID_CODE();
/* find info to be deleted */
str = p.getStr("id_code: ");
id.setID_CODE(str);
if (dbObj.keyfind(tims.ID_CODE, id) == RDME.S_NOTFOUND) {
System.out.println("id_code " + str + " not on file");
return;
}
irec = new INFO();
```

```
name = new AUTHOR();
dbObj.recread(irec);


 /* get author name */
dbObj.findco(tims.HAS_PUBLISHED);
dbObj.recread(name);
/* confirm delete request */
System.out.println("author: " + name.getNAME());
System.out.println("title : " + irec.getINFO_TITLE());
if ((str = p.getStr("delete (y/n)? ")) == null)
return;
if (str.charAt(0) != 'y' && str.charAt(0) != 'Y')
return;
/* disconnect any listed articles */
dbObj.setom(tims.ARTICLE_LIST, tims.HAS_PUBLISHED);
for (stat = dbObj.findfm(tims.ARTICLE_LIST); stat == RDME.S_OKAY;
stat = dbObj.discon(tims.ARTICLE_LIST))
;
/* disconnect and delete borrowers */
dbObj.setom(tims.LOANED_BOOKS, tims.HAS_PUBLISHED);
while (dbObj.findfm(tims.LOANED_BOOKS) == RDME.S_OKAY) {
dbObj.discon(tims.LOANED_BOOKS);
dbObj.delete();
}
/* disconnect and delete abstract */
dbObj.setom(tims.ABSTRACT, tims.HAS_PUBLISHED);
while (dbObj.findfm(tims.ABSTRACT) == RDME.S_OKAY) {
dbObj.discon(tims.ABSTRACT);
dbObj.delete();
}


 /* disconnect and delete intersect and (possibly) key word */
count = new Count();
dbObj.setom(tims.INFO_TO_KEY, tims.HAS_PUBLISHED);
while (dbObj.findfm(tims.INFO_TO_KEY) == RDME.S_OKAY) {
dbObj.discon(tims.INFO_TO_KEY);
dbObj.setmr(tims.KEY_TO_INFO);
dbObj.discon(tims.KEY_TO_INFO);
dbObj.delete();
dbObj.members(tims.KEY_TO_INFO, count);
if (count.getCount() == 0) {
/* delete key word */
dbObj.setro(tims.KEY_TO_INFO);
dbObj.delete();
}
}
```

```
/* disconnect info record from author and delete */
dbObj.discon(tims.HAS_PUBLISHED);
dbObj.delete();
/* delete author too, if authur has no other pubs */
dbObj.members(tims.HAS_PUBLISHED, count);
if (count.getCount() == 0) {
dbObj.setmo(tims.AUTHOR_LIST, tims.HAS_PUBLISHED);
dbObj.discon(tims.AUTHOR_LIST);
dbObj.delete();
}
}


 void loanBook() throws DBException
{
int stat;
String str;
INFO irec = new INFO();
BORROWER brec = new BORROWER();
INFOID_CODE id = new INFOID_CODE();
str = p.getStr("id_code of book to be loaned: ");
if (str == null)
return;
id.setID_CODE(str);
if (dbObj.keyfind(tims.ID_CODE, id) == RDME.S_NOTFOUND)
System.out.println("id_code " + str + " not on file");
else {
dbObj.recread(irec);
System.out.println("Title is '" + irec.getINFO_TITLE() + "'");
dbObj.setor(tims.LOANED_BOOKS);
brec.setFRND(p.getStr("name of borrower: "));
brec.setDATE_BORROWED(Integer.parseInt(p.getStr("date borrowed
: ")));
brec.setDATE_RETURNED(0);
dbObj.fillnew(tims.BORROWER, brec);
dbObj.connect(tims.LOANED_BOOKS);
}
}


 void returnBook() throws DBException
{
String str;
BORROWER brec;
INFOID_CODE id = new INFOID_CODE();
if ((str = p.getStr("id_code of returned book: ")) == null)
return;
id.setID_CODE(str);
```

```
if (dbObj.keyfind(tims.ID_CODE, id) == RDME.S_NOTFOUND) {
System.out.println("id_code " + str + " not on file");
return;
}
brec = new BORROWER();
dbObj.setor(tims.LOANED_BOOKS);
str = p.getStr("name of borrower: ");
while (dbObj.findnm(tims.LOANED_BOOKS) == RDME.S_OKAY) {
dbObj.recread(brec);
if (str.compareTo(brec.getFRND()) == 0) {
if (brec.getDATE_RETURNED() == 0) {
System.out.println("book borrower on: " +
brec.getDATE_BORROWED());
brec.setDATE_RETURNED(Integer.parseInt(p.getStr("date
returned : ")));
dbObj.recwrite(brec);
return;
}
}
}
System.out.println("borrower not on file");
}


 void listLoaners() throws DBException
{
int stat;
INFO irec = new INFO();
BORROWER brec = new BORROWER();
for (stat = dbObj.findfm(tims.LOAN_HISTORY); stat == RDME.S_OKAY;
stat = dbObj.findnm(tims.LOAN_HISTORY)) {
dbObj.recread(brec);
if (brec.getDATE_RETURNED() == 0) {
dbObj.findco(tims.LOANED_BOOKS);
dbObj.recread(irec);
System.out.println("id_code: " + irec.getID_CODE());
System.out.println("title : " + irec.getINFO_TITLE());
System.out.println("borrower by " + brec.getFRND() + " on " +
brec.getDATE_BORROWED());
}
}
}


 boolean executeCommand(String cmd) throws DBException
{
if (cmd == null || cmd.length() == 0)
return true;
```

```
switch (cmd.charAt(0)) {
case '1':
listKeys();
break;
case '2':
listAuthors();
break;
case '3':
byKey();
break;


 case '4':
byAuthor();
break;
case '5':
enterInfo();
break;
case '6':
deleteInfo();
break;
case '7':
loanBook();
break;
case '8':
returnBook();
break;
case '9':
listLoaners();
break;


 case 'q':
case 'Q':
return false;
default:
System.out.println("*** incorrect command -- re-enter");
break;
}
return true;
}
public void run() throws DBException
{
String cmd;
do {
displayCommands();
cmd = p.getStr("enter command: ");
} while (executeCommand(cmd));
```

```
dbObj.close();
taskObj.closetask();
}
}


public class TimsExample
{
public static void main(String[] args) throws ClassNotFoundException
{
Prompt p = new Prompt();
TimsRun t;
try {
t = new TimsRun(p);
t.run();
}
catch (DBException e) {
System.out.println("DBException:");
e.printStackTrace(System.out);
}
}
}
```