WorkingWithBasicCrud

The RDB DAS provides a simple, command-oriented approach for reading and writing to the database.

A simple read

The following illustrates the simplest possible read:

```
DAS das = DAS.FACTORY.createDAS(getConnection());
Command readCustomers = das.createCommand("select * from CUSTOMER where ID = 22590");
DataObject root = readCustomers.executeQuery();
```

The first line creates an instance of the DAS that is primed with a given JDBC Connection object. The Connection can be procured in a number of different ways. This example is from the RDB DAS test suite and the implentation of getConnection() simply uses the DriverManager.getConnection() method.

The second line uses the das instance as a factory to create a Command instance from the provided SQL SELECT string. The third line executes the command which returns a "root" SDO DataObject. This root DataObject is sometimes referred to as a "dummy" root and acts as a container for the graph of data returned from the query. The RDB DAS always returns a dummy root from executing a query.

Once we have a handle to the root we can pull information from the returned customer object using the dynamic SDO apis like this:

```
DataObject cust = root.getDataObject("CUSTOMER[1]");
String name = cust.getString("LASTNAME");
```

The first line retreives the first Customer from the root's list of returned Customers (since we queried by ID, thereis only one). The second line pulls the LASTNAME value from the customer.

A simple write

The DAS leverages the SDO Change Summary to provide a very simple mechanism to flush data changes back to the database. We can illustrate this by continuing the previous example:

```
customer.set("LASTNAME", "Pavick");
das.applyChanges(root);
```

The first line uses the SDO dynamic APIs to change the LASTNAME property of the Customer instance and the second line directs the das to flush graph changes back to the database. At this point, the das scans the SDO ChangeSummary and generates the required UPDATE statement necessary to synchronize this change with the database.

Although this simple example demonstrates a single property change to a single object, the das will process any numbers of changes made to any number of objects in a graph. Even if we read 1000 customers from the database and made several changes to each, flushing those changes back to the database is still just one call to das.applyChanges().

A write commmand

Writes to the database can also be done without using the change summary. To do this, we simply use the das instance as a factory to create a command initialized with a CREATE/UPDATE or DELETE statement.

```
DAS das = DAS.FACTORY.createDAS(getConnection());
Command insert = das.createCommand("insert into CUSTOMER values (10, 'Williams', '5555 Maine
Ave.')");
insert.execute();
```

Although allowing the das to generate write statements from the change summary is recommended, this low level ability to execute aribtrary SQL is another option.