java API

Connecting to the database

Connecting to an ArcSDE server

<sxh java> public static void main (String args[])throws Exception {

```
  SeConnection conn=null;
  String server = “sdeserver”;
  int instance = 5151;
  String database = “sdedb”;
  String user = “user”;
  String password = “passwd”;
  try {
      conn = new SeConnection(server, instance, database, user, password);
  }catch (SeException e) {
      SeError error = e.getSeError();
      System.out.println(error.getSdeErrMsg());
  }
```

} </sxh>

Connection and database information

<sxh java> Vector layerList = conn.getLayers(); for(int index = 0 ; index < layerList.size() ; index++) {

```
  SeLayer layer = (SeLayer)layerList.elementAt(index);
  // Displays the layer’s name
  System.out.println( layer.getName() );
  // Displays the layer’s ID
  System.out.println( layer.getID().longValue() );
  // Displays the layer’s spatial column name
  System.out.println( layer.getSpatialColumn() );
```

} </sxh>

Fetching data

<sxh java> *1. Create an SeSqlConstruct object that contains the name of the table/layer to be queried. To specify a where clause constraint on the query use the second SeSqlConstruct constructor. SeLayer layer = new SeLayer(conn, layerName, spatialColumn); SeSqlConstruct sqlConstruct = new SeSqlConstruct(layer.getName());* 2. Create a java.lang.String array containing

the names of the table columns to be queried. String[] cols = new String[2]; cols[0] = new String(“ColumnOneName”); cols[1] = layer.getSpatialColumn();

3. Define, prepare and execute the query. SeQuery query = new SeQuery(conn, cols, sqlConstruct); query.prepareQuery(); query.execute(); 4. Fetch the first row retrieved by the query into an SeRow object. SeRow row = query.fetch();

5. Retrieve the column definitions of the row retrieved. Get the definitions of all the columns retrieved SeColumnDefinition[] colDefs = SeRow.getColumns();

6. For each column definition, determine the ArcSDE data type of the column. Then retrieve the data in that column. Retrieve data from the first column int colNum = 0; int dataType = colDefs[colNum].getType;

Assuming that the columns we retrieve are either string or shape data type Add more case statements to retrieve all possible SDE data types. *(See Working with Layers Example) switch(dataType) { case SeColumnDefinition.TYPESTRING: System.out.println(colDef.getName()+ row.getString(colNum)); break; case SeColumnDefinition.TYPESHAPE: System.out.println(colDef.getName()); SeShape shape = row.getShape(colNum);* Call a function to retrieve the shape attributes

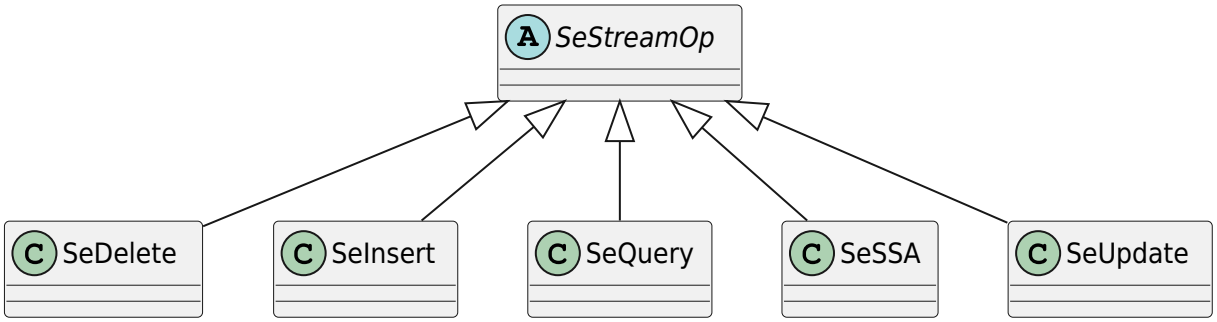
```
        break;
```

}

7. Close the query. </sxh> ## Inserting data <sxh java> 1. Create a java.lang.String array containing the names of the table columns to be populated. Note the array index value of each column; this index value will be used in the SeRow.set* methods later on. The index value for the string column is 0 and the index value for the shape column is 1. String[] cols = new String[2]; cols[0] = new String(“ColumnOneName”); cols[1] = layer.getSpatialColumn();

2. Create an SeInsert object using the current connection handle. This creates an insert stream between the client and the server. Then specify the columns of the table or layer to insert into. Set the SeInsert object to write mode. SeInsert insert = new SeInsert(conn); insert.intoTable(layer.getName(),cols); insert.setWriteMode(true); 3. Obtain the SeRow object that is to be filled, from the insert object. Then set the row object with the data to be inserted. This is done using the SeRow.set* methods. Use the index values from step 1 for the columnPosition parameter of each set method. SeRow row = insert.getRowToSet(); row.setString(0, “Shape Number One”); row.setShape(1,shape);

4. Invoke the execute method of the Insert object to insert the row into the layer. Close the insert stream to finish insert operation. If no more operations are to be done with the current connection, then close the connection as well. insert.execute(); insert.close(); </sxh> ## Stream Function



<sxh java> SeInsert insert = null; try { The stream constructor creates a new stream which consumes

```
  // system resources both on the server and client computers.
  insert = new SeInsert(connection);
```

```
  // Call stream methods here.
  ...
  ...
```

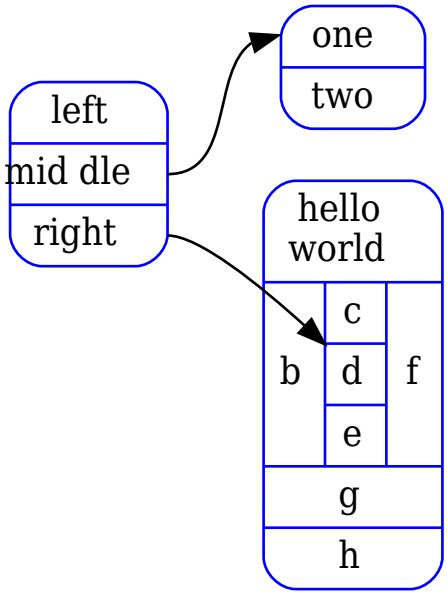
} catch (SeExceptione) {

```
  // Handle exceptions here.
  System.out.println(e.getSeError().getErrDesc() );
```

} finally {

```
  // Close the stream.
  // Calling the stream close function inside the finally clause ensures
  // that the stream will always be closed, even in the event that an
  // exception occurs during the execution of a stream method.
  try {
      insert.close();
  } catch(SeExceptione ) {
      // Handle exception thrown during stream close operation.
      System.out.println(" ERROR : Couldn't close the stream ");
      System.out.println(" e.getSeError().getErrDesc()");
      e.printStackTrace();
  }
```

} </sxh>



관련 문서

Plugin Backlinks: 아무 것도 없습니다.