Table of Contents

JVx uses IStorage to transfer table data between client and server in standardized form. JVx contains an existing implementation for databases, DBStorage, and for dynamically assembled data, AbstractMemStorage. At the client, data is accessed via RemoteDataBooks.

To avoid repeated implementation of familiar features such as metadata caching, server-side triggers, sorting, master/detail, and export, existing base classes can be reused.

The class AbstractStorage implements the handling of server-side triggers. AbstractCachedStorage is a derivative of AbstractStorage and implements the metadata cache.

DBStorage is derived from AbstractCachedStorage and implements the required methods:

```
public abstract void writeCSV(...) throws Exception;

public abstract MetaData executeGetMetaData() throws DataSourceException;

public abstract Object[] executeRefetchRow(...) throws DataSourceException;

public abstract List<Object[]> executeFetch(...) throws DataSourceException;

public abstract Object[] executeInsert(...) throws DataSourceException;

public abstract Object[] executeUpdate(...) throws DataSourceException;

public abstract void executeDelete(...) throws DataSourceException;
```

specifically for databases.

If data has to be assembled dynamically, AbstractMemStorage can be used. This class uses MemDataBook to save the data.

One example of use would be for email queries. The data could be provided to the client in email storage. Another example would be TwitterStorage.

The following is an excerpt from a possible TwitterStorage implementation:

```
@Override
public RowDefinition getRowDefinition() throws ModelException
{
   RowDefinition rowdef = new RowDefinition();
   rowdef.addColumnDefinition(new ColumnDefinition("ID", new
BigDecimalDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("USER", new
StringDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("IMAGE", new
BinaryDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("MESSAGE", new
StringDataType()));

   rowdef.setPrimaryKeyColumnNames(new String[] {"ID"});

   rowdef.setColumnView(null, new ColumnView(rowdef));
```

```
    return rowdef;
}

@Override
public void loadData(MemDataBook pBook, ICondition pFilter) throws
ModelException
{
    pBook.close();
    pBook.open();

    try
    {
       List<Status> liStati = twitter.getUserTimeline(new Paging(1,
iMaxMessages));

        for (Status status : liStati)
        {
           pBook.insert(false);
           pBook.setValues(new String[] {"ID", "USER", "IMAGE", "MESSAGE"},
                           new Object[] {BigDecimal.valueOf(status.getId()),
                                         status.getUser().getName(),
getImage(status.getUser().getProfileImageURL()),
                                         status.getText()});
        }
     }
     catch (Exception ex)
     {
        throw new ModelException("Load data failed!", ex);
     }
}

@Override
public void insert(DataBookEvent pEvent) throws ModelException
{
    try
    {
       Status status =
twitter.updateStatus((String)pEvent.getChangedDataBook().
                                           getValue("MESSAGE"));

       pEvent.getChangedDataBook().setValues(new String[] {"ID", "USER",
"IMAGE"},
                                             new Object[]
{BigDecimal.valueOf(
status.getId()),
status.getUser().getName(),
getImage(status.getUser())});
    }
    catch (TwitterException te)
    {
```

```
      throw new ModelException("Insert failed", te);
   }
}

@Override
public void update(DataBookEvent pEvent) throws ModelException
{
   throw new ModelException("Update is not supported!");
}

@Override
public void delete(DataBookEvent pEvent) throws ModelException
{
   BigDecimal bdId = (BigDecimal)pEvent.getOriginalDataRow().getValue("ID");

   try
   {
      twitter.destroyStatus(bdId.longValue());
   }
   catch (TwitterException te)
   {
      throw new ModelException("Delete failed!", te);
   }
}
```