**Epipog**

**Specification**

**DataStore Class Family**

**Dec. 13, 2016**

# 1. Base Class

The data store object is the base class for storing and accessing data records. Derived objects are extended from the base class, depending on their requirement in the database pipeline. This base class is defined in the file DataStore.java. This base class defines the methods and interfaces for how data is represented in storage in the Epipog application and the operations that may be performed with the data store object.

## 1.1 Fields

The base class defines the following fields:

private Storage storage = null; // data storage  
  
The underlying storage object assigned (bound) to this data store object.  
  
protected Collection collection = null; // assigned collection

The collection object assigned (bound) to this data store object.  
  
protected Data.DataModel dataModel = Data.DataModel.DATA; // data model to use  
  
The data model to use when processing data prior to writing to the underlying storage to/from storage.  
  
protected boolean validate = false; // validate the input according to the data type

A boolean flag indicating if the data values should be validated according to their data type, prior to being written to storage.

## 1.2 Methods

The base class contains the following implemented methods

***Accessors (Getter/Setter)***

public void Collection( Collection collection )

This method binds a collection object to this data store object.

public void Storage( Storage storage )

This method binds a storage object to this data store object.

public StorageStorage()

This method returns the storage object that is bound to this data store object.

public void DataModel( Data.DataModel dataModel );

This method sets the data model for processing data.

public void Validate( boolean validate );

This method sets the flag whether to verifying data, prior to writing to storage, based on its data type.

***I/O***

public void Open() throws DataStoreException;

This method opens the collection using the storage object that has been bound to the data store object. The underlying storage object handles opening the collection, reading in the schema and indices. If an error occurs, a DataStoreException is thrown.

public void Close();

This method closes the collection using the storage object that has been bound to the data store object. The underlying storage object handles closing the collection, written out changes to the schema and indices. If an error occurs, a DataStoreException is thrown.

public void Begin() throws DataStoreException;

This method is seeks to the beginning of the data records for a collection using the storage object bound to the data store object.

public long End() throws DataStoreException;

This method seeks to the end of the data records for a collection using the storage object bound to the data store object, and returns the storage offset (e.g., file pointer position) of that location.

public long Pos() throws StorageException;

This method returns the current position (e.g., last record read or written) for a collection in the storage bound to the data store object.

public void Move( long pos ) throws StorageException;

This method moves to a new location, specified by the argument pos, for a collection in the storage object bound to the data store object.

public boolean Eof() throws StorageException;

This method returns true if the current location is the end of the collection in the storage object bound to the data store object.

public void Write( String value, int length ) throws StorageException;  
public void Write( String value ) throws StorageException;  
public void Write( Character value ) throws StorageException;  
public void Write( byte value ) throws StorageException;  
public void Write( Short value ) throws StorageException;  
public void Write( Integer value ) throws StorageException;  
public void Write( Long value ) throws StorageException;  
public void Write( Float value ) throws StorageException;  
public void Write( Double value ) throws StorageException;  
public void Write( Boolean value ) throws StorageException;  
public void WriteLine( String value ) throws StorageException;

These methods are used to write data records in a collection to the storage object bound to this data store object, based on the data type utilizing method overloading.

public String Read( int length ) throws StorageException;  
public Character ReadChar() throws StorageException;  
public byte ReadByte() throws StorageException;  
public Short ReadShort() throws StorageException;  
public Integer ReadInt() throws StorageException;  
public Long ReadLong() throws StorageException;  
public Float ReadFloat() throws StorageException;  
public Double ReadDouble() throws StorageException;  
public Boolean ReadBoolean() throws StorageException;  
public String ReadLine() throws StorageException;  
  
These methods are used to read data records in a collection from the storage object bound to this data store object , according to their data type.

***CRUD***

public void Delete() throws StorageExc eption;

This method deletes the collection assigned to this data store object from the storage object bound to this data store object. Once deleted, it cannot be undone.

protected String DataCheck( Data.DataModel dataModel, Integer type, String value ) throws DataStoreException

This method verifies a string representation of data according to its BSON data type and data model representation.

protected String StringNoNull( String str );

This method trims all ending null characters. This is used with fixed length strings where any unused space at the end is padded with null bytes.

## 1.4 Abstract Methods (Interface)

The base class contains abstract methods, which must be implemented by the derived classes, for the following:

***CRUD***

public abstract void Insert( ArrayList<Pair<String,String>> keyVals ) throws DataStoreException, StorageException;

This method is implemented in the derived classes to insert a data record into a data store represented by a list of key-value pairs, where the first element is the key (i.e., column name) and the second element is the value in string representation. Dependent on the data store type, the value maybe converted to a binary representation, based on the data type indicated in the schema. If an error occurs, an exception is thrown.

public abstract void InsertC( ArrayList<String> values ) throws DataStoreException, StorageException;

This method is implemented in the derived classes to insert a data record into a data store represented by an ordered list of values, where the ordered list matches a predetermined ordering in a schema assigned to the data store. Each value is in string is string representation. Dependent on the data store type, the value maybe converted to a binary representation, based on the data type indicated in the schema. If an error occurs, an exception is thrown.

public abstract ArrayList<Data[]> Select( ArrayList<String> fields, ArrayList<Where> where ) throws DataStoreException, StorageException;

This method is implemented in the derived classes to select data records from the data store. The fields argument specifies the fields in the data records to retrieve, and the argument where are one or more where clauses to filter the selection. If the fields argument is ‘\*’, then all fields in the data record are retrieved.

The following where operation are supported, unless otherwise noted in the derived class:

= equal  
 != not equal  
 > greater than  
 < less than  
 >= greater than or equal  
 <= less than or equal  
 in value is in a specified set

The return array of Data values is in the same order as specified in the fields argument, or if an ‘\*’ (all) was specified, in a predetermined order specified by the schema.

Dependent on the data store type, the values are converted into the corresponding Epipog Data class types, based on the data type indicated in the schema. If an error occurs, a DataStoreException is thrown.

# 2 DataStoreBinary Derived Class

This derived class extends base class “DataStore”, and implements the methods for a binary store fixed record format data representation, as typically found in an RDBMS database. They are implemented in the file DataStoreBinary.java. This extended class implements the methods and interfaces for accessing and storing data records.

As in traditional RDBMS databases, each record is of fixed length and each record is of a fixed offset within the record. All data is stored in its binary representation. For example, an integer is stored as a 32-bit binary value.

## 2.1 Methods

The extended class contains the implementation of the following methods:

***CRUD***

public void Insert( ArrayList<Pair<String,String>> keyVals ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by a list of key-value pairs, where the first element is the key (i.e., column name) and the second element is the value in string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

Each value is converted from string representation into a binary format, based on the BSON data type specified in the associated schema.

The first byte of each record indicates if the data record is clean (0) or dirty (1). Dirty (e.g., marked for deletion) are ignored by CRUD operations and are eventually deleted during a vacuum operation.

public void InsertC( ArrayList<String> values ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by an ordered list of values, where the ordered list matches a predetermined ordering in a schema assigned to the data store. Each value is in string is string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

Each value is converted from string representation into a binary format, based on the BSON data type specified in the associated schema.

public ArrayList<Data[]> Select( ArrayList<String> fields, ArrayList<Where> where ) throws DataStoreException, StorageException;

This method selects data records from the data store. The argument fields specifies the fields (columns) in the data records to retrieve, and the argument where are one or more where clauses to filter the selection. If the fields argument is ‘\*’, then all fields in the data record are retrieved. All where clauses, as documented in the base class, are supported.

The return array of Data values is in the same order as specified in the fields argument, or if an ‘\*’ (all) was specified, in a predetermined order specified by the schema.

The values are converted into the corresponding Epipog Data class types, based on the data type indicated in the schema. If an error occurs, a DataStorException is thrown.

# 3 DataStoreJSON Derived Class

This derived class extends the base class “DataStore”, and implements the methods for a document object based store, such as found in some modern NoSQL database. They are implemented in the file DataStoreJSON.java. This extended class implements the methods and interfaces for accessing and storing data records.

## 3.1 Methods

The extended class contains the implementation of the following methods:

***CRUD***

public void Insert( ArrayList<Pair<String,String>> keyVals ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by a list of key-value pairs, where the first element is the key (i.e., column name) and the second element is the value in string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

If the validate flag is set, then each value is validated, based on the BSON data type specified in the associated schema.

The first field (clx) of each record indicates if the data record is clean (0) or dirty (1). Dirty (e.g., marked for deletion) are ignored by CRUD operations and are eventually deleted during a vacuum operation.

Each record appears in storage in the following format, following by a newline:

{ clx: 0, field: “value”, field: “value”, … }

public void InsertC( ArrayList<String> values ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by an ordered list of values, where the ordered list matches a predetermined ordering in a schema assigned to the data store. Each value is in string is string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

If the validate flag is set, then each value is validated, based on the BSON data type specified in the associated schema.

The first field (clx) of each record indicates if the data record is clean (0) or dirty (1). Dirty (e.g., marked for deletion) are ignored by CRUD operations and are eventually deleted during a vacuum operation.

Each record appears in storage in the following format, following by a newline:

{ clx: 0, field: “value”, field: “value”, … }

public ArrayList<Data[]> Select( ArrayList<String> fields, ArrayList<Where> where ) throws DataStoreException, StorageException;

This method selects data records from the data store. The argument fields specifies the fields (columns) in the data records to retrieve, and the argument where are one or more where clauses to filter the selection. If the fields argument is ‘\*’, then all fields in the data record are retrieved. All where clauses, as documented in the base class, are supported.

The return array of Data values is in the same order as specified in the fields argument, or if an ‘\*’ (all) was specified, in a predetermined order specified by the schema.

The values are converted into the corresponding Epipog Data class types, based on the data type indicated in the schema. If an error occurs, a DataStoreException is thrown.

# 4 DataStoreSV Derived Class

This derived class extends the base class “DataStore”, and implements the methods for a character sequenced separated (e.g., CSV) based store, such as found in some columnar databases. They are implemented in the file DataStoreSV.java. This extended class implements the methods and interfaces for accessing and storing data records.

## 4.1 Fields

private byte separator; // field separator

This field is the ASCII character that will be used for storing and parsing separated values when accessed to/from storage.

## 4.2 Methods

The extended class contains the implementation of the following methods:

***Constructor***

public DataStoreSV( byte separator )

The argument separator specifies the character sequence (e.g., comma) to use for separating ordered values when storing and accessing the values to/from storage.

***CRUD***

public void Insert( ArrayList<Pair<String,String>> keyVals ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by a list of key-value pairs, where the first element is the key (i.e., column name) and the second element is the value in string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

If the validate flag is set, then each value is validated, based on the BSON data type specified in the associated schema.

The first field (clx) of each record indicates if the data record is clean (0) or dirty (1). Dirty (e.g., marked for deletion) are ignored by CRUD operations and are eventually deleted during a vacuum operation.

Each record appears in storage in the following format, following by a newline:

clx, value, value, value, ….

public void InsertC( ArrayList<String> values ) throws DataStoreException, StorageException;

This method inserts a data record into a data store represented by an ordered list of values, where the ordered list matches a predetermined ordering in a schema assigned to the data store. Each value is in string is string representation. The associated collection must have a schema and the key (column) names must match the column names in the schema, or a DataStoreException is thrown, and the transaction is rolled back.

If the validate flag is set, then each value is validated, based on the BSON data type specified in the associated schema.

The first field (clx) of each record indicates if the data record is clean (0) or dirty (1). Dirty (e.g., marked for deletion) are ignored by CRUD operations and are eventually deleted during a vacuum operation.

Each record appears in storage in the following format, following by a newline:

clx, value, value, value, …

public ArrayList<Data[]> Select( ArrayList<String> fields, ArrayList<Where> where ) throws DataStoreException, StorageException;

This method selects data records from the data store. The argument fields specifies the fields (columns) in the data records to retrieve, and the argument where are one or more where clauses to filter the selection. If the fields argument is ‘\*’, then all fields in the data record are retrieved. All where clauses, as documented in the base class, are supported.

The return array of Data values is in the same order as specified in the fields argument, or if an ‘\*’ (all) was specified, in a predetermined order specified by the schema.

The values are converted into the corresponding Epipog Data class types, based on the data type indicated in the schema. If an error occurs, a DataStoreException is thrown.

# 5 DataStoreCSV Derived Class

This derived class extends the class “DataStoreSV”, and implements the methods for a comma separated (i.e., CSV) based store, such as found in some columnar databases. They are implemented in the file DataStoreCSV.java. This extended class implements the methods and interfaces for accessing and storing data records.

## 5.1 Methods

The extended class contains the implementation of the following methods:

***Constructor***

public DataStoreCSV()

This constructor instantiates the DataStoreSV() constructor, passing the argument value ‘,’ (comma) to initialize the data store object as a pipe separated value data store.

# 6 DataStorePSV Derived Class

This derived class extends the class “DataStoreSV”, and implements the methods for a pipe separated (i.e., PSV) based store, such as found in some columnar databases. They are implemented in the file DataStorePSV.java. This extended class implements the methods and interfaces for accessing and storing data records.

## 6.1 Methods

The extended class contains the implementation of the following methods:

***Constructor***

public DataStorePSV()

This constructor instantiates the DataStoreSV() constructor, passing the argument value ‘|’ (pipe) to initialize the data store object as a pipe separated value data store.

# 7 DataStoreTSV Derived Class

This derived class extends the class “DataStoreSV”, and implements the methods for a tab separated (i.e., TSV) based store, such as found in some columnar databases. They are implemented in the file DataStoreTSV.java. This extended class implements the methods and interfaces for accessing and storing data records.

## 5.1 Methods

The extended class contains the implementation of the following methods:

***Constructor***

public DataStoreCSV()

This constructor instantiates the DataStoreSV() constructor, passing the argument value ‘\t’ (tab) to initialize the data store object as a tab separated value data store.