**Epipog**

**Specification**

**Storage Interface Family**

**Dec. 13, 2016**

# 1. Interface

The storage object is the interface definition for attaching an underlying physical storage representation implementation to a data store representation. Derived objects are implemented from the interface, depending on their requirement in the database pipeline. This interface is defined in the file Storagre.java. This interface defines the methods and interfaces for how physical storage is specified and accessed for a collection in an Epipog application and the operations that may be performed with the storage object.

## 1.2 Interface Methods

The interface contains the following non-implemented methods

***Getter/Setter***

public void Storage( String volume, String path );

This method assigns a volume and path in the volume for accessing collections with the storage object.

public void DataStoreType( DataStore dataStore );

This method associates a data store representation with this instance of a storage object.

public String DataStoreType();

This method gets the data store representation associated with this storage object.

public ArrayList<String> List();

This method returns an array list of all collections (names) stored in this type of storage at the volume location specified for this instance.

***I/O***

public void Open() throws StorageException;

This method opens access to a collection based on the underlying physical storage representation.

public void Close() throws StorageException;

This method closes access to a collection based on the underlying physical storage representation.

public void Begin() throws StorageException;

This method is seeks to the beginning of the data records for a collection in the underlying physical storage representation.

public long End() throws StorageException;

This method seeks to the end of the data records for a collection in the underlying physical storage representation, 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 underlying physical storage representation.

public void Move( long pos ) throws StorageException;

This method moves to a new location, specified by the argument pos, for a collection in the underlying physical storage representation.

public boolean Eof() throws StorageException;

This method returns true if the current location is the end of the collection in the underlying physical storage representation.

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 in the underlying physical storage representation, 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 in the underlying physical storage representation, according to their data type.

***Schema***

public void Write( Schema schema ) throws StorageException;

This method writes the collection’s schema to the underlying physical storage.

public ArrayList<Pair<String,Integer>> ReadSchema() throws StorageException;

This method reads in a collection’s schema from the underlying physical storage.

## public void Delete() throws StorageException;

This method deletes the collection assigned to this storage object from the underlying physical storage. Once deleted, it cannot be undone.

# 2 StorageSingleFile Implementation

This implementation implements the “Storage” interface. They are implemented in the file StorageSingleFile.java. This implementation implements the methods and interfaces for storing a collection as a single monolithic file.

## 2.1 Fields

The implementation defines the following fields, which are only accessible (private/protected) by this class or a derived class:  
  
private String volume = null; // storage location  
private String path = null; // storage file path  
private RandomAccessFile fd = null; // file pointer for data storage file  
private RandomAccessFile sc = null; // file pointer for schema storage file   
private RandomAccessFile ix = null; // file pointer for index storage file

The volume field specifies the volume in which the collection will be stored. For example, on a DOS filesystem, the volume maybe specified like C:/tmp. The path is the file path (not inclusive of file suffixes) of the collection within the volume (e.g., mycollection). In the example given, the data records, schema and indices would be stored as:

C:/tmp/mycollection.dat  
C:/tmp/mycollection.sch   
C:/tmp/mycollection.idx

The fields, fd, sc and ix are file pointers for random access read/write/seek to the data records, schema and indices, respectively.

## 2.2 Methods

The implementation contains the following methods:

***Getter/Setter***

public void Storage( String volume, String path );

This method assigns a volume and path in the volume for accessing collections with the storage object. For example, on a DOS filesystem the volume might be specified as C:/tmp.

public void DataStoreType( DataStore dataStore );

This method associates a data store representation with this instance of a storage object.

public String DataStoreType();

This method gets the data store representation associated with this storage object.

public ArrayList<String> List();

This method returns an array list of all collections (names) stored in this type of storage at the volume location specified for this instance. For example, if three collections by the name foo, goo and hoo are written to the volume C:/tmp, and the storage volume for this instance is set to C:/tmp, then this method would return foo, goo and hoo.

***I/O***

public void Open() throws StorageException;

This method opens file access to the data records for the collection (i.e., file ends in .dat).

public void Close() throws StorageException;

This method closes the file access to the data records for the collection.

public void Begin() throws StorageException;

This method is seeks to the beginning of the data records for a collection in the corresponding storage file.

public long End() throws StorageException;

This method seeks to the end of the data records for a collection in the corresponding storage file, 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 corresponding storage file.

public void Move( long pos ) throws StorageException;

This method moves to a new location, specified by the argument pos, for a collection in the corresponding storage file.

public boolean Eof() throws StorageException;

This method returns true if the current location is the file end of the data records in the corresponding storage file.

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 in the corresponding file storage, 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 corresponding file storage, according to their data type.

***Note:*** *The data type Char is written/read as a 2-byte UNICODE character.*