**Epipog**

**Specification**

**Data Class Family**

**Nov. 1, 2016**

# 1. Base Class Family

The data object consists of a family of base classes, each extending beyond the other. Derived objects are extended from one of the base classes in the base class family, depending on their requirement in the database pipeline. The base class family is as follows:

Data

This base class is the most streamed-lined. It is used for the fastest computational and smallest footprint in memory. The data object does not contain any information on validation (preflight checks) or semantic meaning (linked data, namespace).  
  
DataState (extends Data)

This base class is the next most streamed-lined. It has an additional byte per data value in memory. The data object contains information on validation (preflight checks) but does not contain any information on semantic meaning (linked data, namespace).

DataMeta (Extends DataState)

This base class is the least streamed-lined. It contains additional fields per data value in memory. The data object contains information on validation (preflight checks) and on semantic meaning (linked data, namespace).

# 2. Data Base Class ( Data.java)

This base class is part of the base class family for the Data Class family ‘Data’, and is defined in the file Data.java. This base class defines the methods and interfaces for how data is represented within the Epipog application and the operations that may be performed on a data object.

## Fields

The base class defines the following fields:

protected Object value;

This field holds the value of the data object in a native Java data type. Access to the data value is limited to the derived object (protected).

## Methods

The base class contains no implemented methods.

## Abstract Methods (Interface)

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

***Typeof***

public abstract String Type();

The method returns the string representation of the name of the data type.  
  
public abstract Integer Size();

The method returns the byte size of the data value.

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

public abstract Object Get();

This method returns the value of the data object in the corresponding native java data type. For example, a date would be returned as a java Date object.

public abstract void Set( Object v );

This method sets the value of the data object. The argument v is the value of the object in the corresponding native java data type. For example, if the data object is a date, the argument v would be a java Date object.

***String***

public abstract String AsString();

This method returns the value of the data object in a string representation. For example, the integer value 101 would be returned as the string “101”.

public abstract void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. If the string representation is invalid, a DataException is thrown.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog data type. For example, the comparison methods for an Epipog Date object take as an argument another Epipog Date object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# 

# 3. DataState Base Class (DataState.java)

This base class is part of the base class family for the Data Class family ‘Data’, and is defined in the file DataState.java. It extends the base class Data. This base class defines the methods and interfaces for how data is represented within the Epipog application and the operations that may be performed on a data object.

## 3.1 Fields

The base class defines the following fields:

private byte state = 0x0;

This single byte field contains bitwise flags for setting states of the data object, as follows:

0x01 data item is undefined (e.g., like null in MySQL)  
 0x02 date item is validated (e.g., passed thru data wrangling and safe to use)  
 0x04 date item is not valid (e.g., not within expected range)

## Methods

The base class contains the following implemented methods:

***State***public void Undefined()  
public boolean IsUndefined()

The above methods are accessor methods for setting and getting the ‘undefined’ state of the data value. By default, data values are defined. For example, a null entry in a table column can be represented by setting the corresponding data value to undefined.

public void Validated()  
public boolean IsValidated()

The above methods are accessor methods for setting and getting the ‘validated’ state of the data value. By default, data values are not validated. For example, the state can be used to indicate that the data value was validated against some requirement during a preflight stage (e.g., a phone number field is verified to be of correct form and range).

public void NotValid()  
public Boolean IsNotValid()

The above methods are accessor methods for setting and getting the ‘not valid’ state of the date value. By default, data values are valid. For example, the state can be used to indicate the data value failed a verification check in a preflight stage (e.g., a phone number field is verified to be of correct form and range).   
  
public void Clear();

The above method clears the state bits.

## Abstract Methods (Interface)

The base class has the following abstract methods, which would need to be implemented in a derived class.

public abstract Object Base();  
  
This method will clone the existing data state object with the lessor data object of the same data type. This can be used to reduce the memory footprint in later stages of the database pipeline when state information is no longer needed.

***String***

public abstract void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. Additionally, the method will set the validated bit. If the string is empty, if will additionally set the undefined bit. If the string representation is invalid, the not valid bit is set.

Unlike the same method derived from the Data base class, this method overrides the base method and does not throw an exception.

# 4. DataMeta Base Class (DataMeta.java)

This base class is part of the base class family for the Data Class family ‘Data’, and is defined in the file DataMeta.java. It extends the base class Data. This base class defines the methods and interfaces for how data is represented within the Epipog application and the operations that may be performed on a data object.

## 4.1 Fields

The base class defines the following fields:

private short ontology = 0;   
private short id = 0;

The field ontology is an internal identifier that is mapped to an external data ontology or namespace. A data ontology is a specification that defines the semantic meaning (context) of a named field within the ontology. For example, it might define a domain of values (i.e., range or set), unit and adornments such as a currency symbol.

## Methods

The base class contains the following implemented methods:

public void Set( short ontology, short id )

public void Set( String ontology, String id )

In Progress

## Abstract Methods (Interface)

The base class contains no additional abstract methods, which would need to be implemented in a derived class.

**TBD**

# Data(Short, Integer,Long) Derived Classes

These derived classes extend base class “Data”, and implement the data model for handling integers. They are implemented in the files DataShort.java, DataInteger.java and DataLong.java. These extended classes implement the methods and interfaces for how integer data is represented within the Epipog application and the operations that may be performed on a data object.

DataShort : 16bit integers  
 DataInteger : 32bit integers  
 DataLong : 64bit integers

## 5.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: short, integer, or long.  
  
public Integer Size();

The method returns the byte size of the data value: 2 (short), 4 (integer), or 8 (long).

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

public Object Get();

This method returns the integer value of the data object in the corresponding native java integer data type.

public void Set( Object v );

This method sets the integer value of the data object. The argument v is the value of the object in the corresponding native java integer data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the integer value 101 would be returned as the string “101”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. If the string representation is invalid, a DataException is thrown . This method expects the strings to be base 10 integers. Any other representation will (e.g., comma for 1000 units, period for decimal point, hex numbers) will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog integer data type. For example, the comparison methods for an Epipog short integer object take as an argument another Epipog short integer object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

## Abstract Methods (Interface)

The extended classes contain no abstract methods.

# Data(Float,Double) Derived Classes

These derived classes extend base class “Data”, and implement the data model for handling floating point numbers. They are implemented in the files DataFloat.java and DataDouble.java. These extended classes implement the methods and interfaces for how floating point data is represented within the Epipog application and the operations that may be performed on a data object.  
  
 DataFloat : 32bit floating point  
 DataDouble : 64bit floating point

## 6.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: float, or double.  
  
public Integer Size();

The method returns the byte size of the data value: 4 (float), or 8 (double).

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

public Object Get();

This method returns the floating point value of the data object in the corresponding native java floating point data type.

public void Set( Object v );

This method sets the floating point value of the data object. The argument v is the value of the object in the corresponding native java floating point data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the floating point value 1.01 would be returned as the string “1.01”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. If the string representation is invalid, a DataException is thrown . This method expects the strings to be in base 10 floating point syntax. Any other representation will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog floating point data type. For example, the comparison methods for an Epipog double floating point object take as an argument another Epipog double floating point object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

## Abstract Methods (Interface)

The extended classes contain no abstract methods.

# DataBoolean Derived Class

This derived class extends the base class “Data”, and implements the data model for handling boolean values. They are implemented in the file DataBoolean.java. This extended class implements the methods and interfaces for how boolean data is represented within the Epipog application and the operations that may be performed on a data object.

## 7.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: boolean.  
  
public Integer Size();

The method returns the byte size of the data value: 1.

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

public Object Get();

This method returns the boolean value of the data object in the corresponding native java boolean data type.

public void Set( Object v );

This method sets the boolean value of the data object. The argument v is the value of the object in the corresponding native java boolean data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the boolean value true would be returned as the string “true”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. If the string representation is invalid, a DataException is thrown . This method expects the strings to be “true” or “false”, and may be case insensitive. Any other representation will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog boolean data type. For example, the comparison methods for an Epipog boolean object take as an argument another Epipog boolean object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataChar Derived Class

This derived class extends the base class “Data”, and implements the data model for handling (UTF-8) character values. They are implemented in the file DataChar.java. This extended class implements the methods and interfaces for how character data is represented within the Epipog application and the operations that may be performed on a data object.

## 8.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: char.  
  
public Integer Size();

The method returns the character size (not bytes) of the data value: 1.

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

public Object Get();

This method returns the character value of the data object in the corresponding native java character data type.

public void Set( Object v );

This method sets the character value of the data object. The argument v is the value of the object in the corresponding native java character data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the character value ‘c’ would be returned as the string “c”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. If the string representation is invalid, a DataException is thrown . This method expects the strings to be a single (UTF-8) character. Any other representation will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog character data type. For example, the comparison methods for an Epipog character object take as an argument another Epipog character object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# Data(String,StringFixed) Derived Class

These derived classes extend the base class “Data”, and implements the data model for handling (UTF-8) string values. They are implemented in the files DataString.java and DataStringFixed.java. These extended classes implements the methods and interfaces for how string data is represented within the Epipog application and the operations that may be performed on a data object.

DataString : variable length strings  
 DataStringFixed : fixed length strings

## 9.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Constructor***

public DataStringFixed( int size);

The derived class DataStringFixed overrides the default constructor and takes a single argument, which is the maximum (fixed) size of the string. If the argument is zero or less than, a DataException is thrown.

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: string (variable length) or string(N), where N is a numeric number (fixed length strings).  
  
public Integer Size();

The method returns the character size (not bytes) of the data value: the string length if variable length string or max length if fixed length string. For example, if a 10 character fixed length string holds a three character value, it will return 10.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java string data type.

public void Set( Object v );

This method sets the string value of the data object. The argument v is the value of the object in the corresponding native java string data type. If the data type is DataStringFixed and the string would exceed the maximum length, then a DataException is thrown.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the string value ‘hello world’ would be returned as the string “hello world”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. This method expects the strings to be a (UTF-8) string. If the data type is DataStringFixed and the string would exceed the maximum length, then a DataException is thrown.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog string data type. For example, the comparison methods for an Epipog string object take as an argument another Epipog string object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataTime Derived Class

This derived class extends the base class “Data”, and implements the data model for handling time values. They are implemented in the file DataTime.java. This extended class implements the methods and interfaces for how time data is represented within the Epipog application and the operations that may be performed on a data object.

## 10.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: time.  
  
public Integer Size();

The method returns the byte size of the data value: 8.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java time data type.

public void Set( Object v );

This method sets the time value of the data object. The argument v is the value of the object in the corresponding native java time data type.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the time value 12:02 would be returned as the string “12:02”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. This method expects the strings to be in the form HH:mm:ss (2 digit hours, 2 digit minutes, 2 digit seconds) or HH:mm (2 digit hours, 2 digit minutes). Any other representation will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog time data type. For example, the comparison methods for an Epipog time object take as an argument another Epipog time object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataDate Derived Class

This derived class extends the base class “Data”, and implements the data model for handling date values. They are implemented in the file DataDate.java. This extended class implements the methods and interfaces for how time data is represented within the Epipog application and the operations that may be performed on a data object.

## 11.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: date.  
  
public Integer Size();

The method returns the byte size of the data value: 8.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java date data type.

public void Set( Object v );

This method sets the time value of the data object. The argument v is the value of the object in the corresponding native java date data type.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the date value April 5, 2016 would be returned as the string “2016-04-15”.

public void Parse( String s ) throws DataException;

This method parses a string representation of the data value and sets the value of the data object accordingly. This method expects the strings to be in the form yyyy-MM-dd (4 digit year, 2 digit month, 2 digit day). Any other representation will result in an exception.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog date data type. For example, the comparison methods for an Epipog date object take as an argument another Epipog date object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataState(Short, Integer,Long) Derived Classes

These derived classes extend base class “DataState”, and implement the data model for handling integers, where the input is preprocessed for input validation. They are implemented in the files DataStateShort.java, DataStateInteger.java and DataStateLong.java. These extended classes implement the methods and interfaces for how integer data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

DataStateShort : 16bit integers  
 DataStateInteger : 32bit integers  
 DataStateLong : 64bit integers

## 12.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: short, integer, or long.  
  
public Integer Size();

The method returns the byte size of the data value: 2 (short), 4 (integer), or 8 (long).

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

public Object Get();

This method returns the integer value of the data object in the corresponding native java integer data type.

public void Set( Object v );

This method sets the integer value of the data object. The argument v is the value of the object in the corresponding native java integer data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the integer value 101 would be returned as the string “101”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* base 10 integer value
* base16 integer value, proceeded by a 0X or 0x prefix.
* Comma in thousands units (e.g, 1,000,000).
* Multiplier suffixes: K = thousand, M = million, B = billion

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog integer data type. For example, the comparison methods for an Epipog short integer object take as an argument another Epipog short integer object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataState(Float,Double) Derived Classes

These derived classes extend base class “DataState”, and implement the data model for handling floating point values, where the input is preprocessed for input validation. They are implemented in the files DataStateFloat.java and DataStateDouble.java. These extended classes implement the methods and interfaces for how floating point data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

DataFloat : 32bit floating point  
 DataDouble : 64bit floating point

## 13.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: float, or double.  
  
public Integer Size();

The method returns the byte size of the data value: 4 (float), or 8 (double).

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

public Object Get();

This method returns the floating point value of the data object in the corresponding native java floating point data type.

public void Set( Object v );

This method sets the floating point value of the data object. The argument v is the value of the object in the corresponding native java floating point data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the floating point value 1.01 would be returned as the string “1.01”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* Floating point notation with decimal point
* Exponent suffix (e or E)

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog floating point data type. For example, the comparison methods for an Epipog double floating point object take as an argument another Epipog double floating point object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataStateBoolean Derived Class

This derived class extends the base class “DataState”, and implements the data model for handling boolean values, where the input is preprocessed for input validation. They are implemented in the file DataStateBoolean.java. This extended class implements the methods and interfaces for how boolean data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

## 14.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: boolean.  
  
public Integer Size();

The method returns the byte size of the data value: 1.

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

public Object Get();

This method returns the boolean value of the data object in the corresponding native java boolean data type.

public void Set( Object v );

This method sets the Boolean value of the data object. The argument v is the value of the object in the corresponding native java boolean data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the Boolean value true would be returned as the string “true”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* True or False, case-insensitive
* T or F, case-insensitive
* 0 or 1

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog boolean data type. For example, the comparison methods for an Epipog boolean object take as an argument another Epipog boolean point object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataChar Derived Class

This derived class extends the base class “DataState”, and implements the data model for handling (UTF-8) character values, where the input is preprocessed for input validation. They are implemented in the file DataStateChar.java. This extended class implements the methods and interfaces for how character data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

## 15.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: char.

public Integer Size();

The method returns the character size (not bytes) of the data value: 1.

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

public Object Get();

This method returns the character value of the data object in the corresponding native java character data type.

public void Set( Object v );

This method sets the character value of the data object. The argument v is the value of the object in the corresponding native java character data type

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the character value ‘c’ would be returned as the string “c”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* Single character
* UNICODE representation \uXXXX or \UXXXX

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog character data type. For example, the comparison methods for an Epipog character object take as an argument another Epipog character point object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# DataState(String,StringFixed) Derived Class

These derived classes extend the base class “DataState”, and implements the data model for handling (UTF-8) string values, where the input is preprocessed for input validation. They are implemented in the files DataStateString.java and DataStateStringFixed.java. These extended classes implements the methods and interfaces for how string data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

DataStateString : variable length strings  
 DataStateStringFixed : fixed length strings

## 16.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Constructor***

public DataStateStringFixed( int size);

The derived class DataStateStringFixed overrides the default constructor and takes a single argument, which is the maximum (fixed) size of the string. If the argument is zero or less than, a DataException is thrown.

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: string (variable length) or string(N), where N is a numeric number (fixed length strings).  
  
public Integer Size();

The method returns the character size (not bytes) of the data value: the string length if variable length string or max length if fixed length string. For example, if a 10 character fixed length string holds a three character value, it will return 10.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java string data type.

public void Set( Object v );

This method sets the string value of the data object. The argument v is the value of the object in the corresponding native java string data type. If the data type is DataStringFixed and the string would exceed the maximum length, then the state is set to Not Valid.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the string value ‘hello world’ would be returned as the string “hello world”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* If variable length (DataStateString), any non-zero length string
* If fixed length, any non-zero length string less than or equal to the maximum length

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog string data type. For example, the comparison methods for an Epipog string object take as an argument another Epipog string object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# 17 DataStateTime Derived Class

This derived class extends the base class “DataState”, and implements the data model for handling time values, where the input is preprocessed for input validation. They are implemented in the file DataStateTime.java. This extended class implements the methods and interfaces for how time data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

## 17.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: time.  
  
public Integer Size();

The method returns the byte size of the data value: 8.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java time data type.

public void Set( Object v );

This method sets the time value of the data object. The argument v is the value of the object in the corresponding native java time data type.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the time value 12:02:00 (hours, minutes, seconds) would be returned as the string “12:02:00”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* Time format HH:mm (hours, minutes)
* Time format HH:mm:ss (hours, minutes, seconds)

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog time data type. For example, the comparison methods for an Epipog time object take as an argument another Epipog time object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set

# 18 DataStateTime Derived Class

This derived class extends the base class “DataState”, and implements the data model for handling date values, where the input is preprocessed for input validation. They are implemented in the file DataStateDate.java. This extended class implements the methods and interfaces for how date data is represented, and preprocessed for validation within the Epipog application and the operations that may be performed on a data object.

## 18.1 Fields

The extended class does not define any additional fields.

## Methods

The extended classes contain the implementation of the following methods:

***Typeof***

public String Type();

The method returns the string representation of the name of the data type: date.  
  
public Integer Size();

The method returns the byte size of the data value: 8.

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

public Object Get();

This method returns the string value of the data object in the corresponding native java date data type.

public void Set( Object v );

This method sets the date value of the data object. The argument v is the value of the object in the corresponding native java date data type.

***String***

public String AsString();

This method returns the value of the data object in a string representation. For example, the date value 2016-07-03 (year,month,day) would be returned as the string “2016-07-03”.

public void Parse( String s );

This method parses a string representation of the data value and sets the value of the data object accordingly. In this data model, derived from the DataState base class, the method performs validation on the data input.

The method does not throw any exceptions, but instead sets state (bitwise) flags according to the validation results. If the input is null or an empty string, the data value is set to undefined. Otherwise, the input string is validated against the following representations:

* Date Format: yyyy-mm-dd (year, month, day)

If validation fails, the date value is set to not valid.

***Comparison***

The following methods perform comparison operation with a data value in the Epipog time data type. For example, the comparison methods for an Epipog time object take as an argument another Epipog time object.

public abstract boolean EQ( Object v ); // equal  
public abstract boolean NE( Object v ); // not equal  
public abstract boolean LT( Object v ); // less than  
public abstract boolean GT( Object v ); // greater than  
public abstract boolean LE( Object v ); // less than or equal  
public abstract boolean GT( Object v ); // greater than or equal  
public abstract boolean IN(Object[] v); // in set