The JDBC$^{(tm)}$ API Version 1.20

January 3, 1997

Part 2 - Classes and Exceptions

This document contains a paper copy of the JDBC API online documentation that is distributed with the JDBC package and is also available on http://splash.javasoft.com/jdbc.

It takes the place of the source code comments that were originally included as part of the JDBC specification.

---

package java.sql

Interface Index

- CallableStatement
- Connection
- DatabaseMetaData
- Driver
- PreparedStatement
- ResultSet
- ResultSetMetaData
- Statement

Class Index

- Date
- DriverManager
- DriverPropertyInfo
- Time
- Timestamp
- Types

Exception Index

- DataTruncation
- SQLException
- SQLWarning

Class java.math.BigDecimal

```
java.lang.Object
   |
   +----java.lang.Number
```

```
        |
        +----java.math.BigDecimal
```

---

public class **BigDecimal**
extends Number

Immutable, arbitrary-precision signed decimal numbers. A BigDecimal consists of an arbitrary precision integer value and a non-negative integer scale, which represents the number of decimal digits to the right of the decimal point. (The number represented by the BigDecimal is intVal/10**scale.) BigDecimals provide operations for basic arithmetic, scale manipulation, comparison, format conversion and hashing.

The BigDecimal class gives its user complete control over rounding behavior, forcing the user to explicitly specify a rounding behavior for operations capable of discarding precision (divide and setScale). Eight *rounding modes* are provided for this purpose. Two types of operations are provided for manipulating the scale of a BigDecimal: scaling/rounding operations and decimal point motion operations. Scaling/Rounding operations (SetScale) return a BigDecimal whose value is approximately (or exactly) equal to that of the operand, but whose scale is the specified value; that is, they increases or decreases the precision of the number with minimal effect on its value. Decimal point motion operations (movePointLeft and movePointRight) return a BigDecimal created from the operand by moving the decimal point a specified distance in the specified direction; they change a number’s value without affecting its precision.

See Also:
    BigInteger

Variable Index

- **ROUND_CEILING**
  If the BigDecimal is positive, behave as for ROUND_UP; if negative, behave as for ROUND_DOWN.
- **ROUND_DOWN**
  Never increment the digit prior to a discarded fraction (i.e., truncate).
- **ROUND_FLOOR**
  If the BigDecimal is positive, behave as for ROUND_DOWN; if negative behave as for ROUND_UP.
- **ROUND_HALF_DOWN**
  Behave as for ROUND_UP if the discarded fraction is > .5; otherwise, behave as for ROUND_DOWN.
- **ROUND_HALF_EVEN**
  Behave as for ROUND_HALF_UP if the digit to the left of the discarded fraction is odd; behave as for ROUND_HALF_DOWN if it’s even.
- **ROUND_HALF_UP**
  Behave as for ROUND_UP if the discarded fraction is >= .5; otherwise, behave as for

ROUND_DOWN.

- **ROUND_UNNECESSARY**
  This "pseudo-rounding-mode" is actually an assertion that the requested operation has an exact result, hence no rounding is necessary.
- **ROUND_UP**
  Always increment the digit prior to a non-zero discarded fraction.

Constructor Index

- **BigDecimal**(BigInteger)
  Translates a BigInteger into a BigDecimal.
- **BigDecimal**(BigInteger, int)
  Translates a BigInteger and a scale into a BigDecimal.
- **BigDecimal**(double)
  Translates a double into a BigDecimal.
- **BigDecimal**(String)
  Constructs a BigDecimal from a string containing an optional minus sign followed by a sequence of zero or more decimal digits, optionally followed by a fraction, which consists of a decimal point followed by zero or more decimal digits.

Method Index

- **abs**()
  Returns a BigDecimal whose value is the absolute value of this number, and whose scale is this.scale().
- **add**(BigDecimal)
  Returns a BigDecimal whose value is (this + val), and whose scale is MAX(this.scale(), val.scale).
- **compareTo**(BigDecimal)
  Returns -1, 0 or 1 as this number is less than, equal to, or greater than val.
- **divide**(BigDecimal, int)
  Returns a BigDecimal whose value is (this / val), and whose scale is this.scale().
- **divide**(BigDecimal, int, int)
  Returns a BigDecimal whose value is (this / val), and whose scale is as specified.
- **doubleValue**()
  Converts the number to a double.
- **equals**(Object)
  Returns true iff x is a BigDecimal whose value is equal to this number.
- **floatValue**()
  Converts this number to a float.
- **hashCode**()
  Computes a hash code for this object.
- **intValue**()
  Converts this number to an int.
- **longValue**()
  Converts this number to a long.

- **max**(BigDecimal)
  Returns the BigDecimal whose value is the greater of this and val.
- **min**(BigDecimal)
  Returns the BigDecimal whose value is the lesser of this and val.
- **movePointLeft**(int)
  Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left.
- **movePointRight**(int)
  Moves the decimal point the specified number of places to the right.
- **multiply**(BigDecimal)
  Returns a BigDecimal whose value is (this * val), and whose scale is this.scale() + val.scale.
- **negate**()
  Returns a BigDecimal whose value is -1 * this, and whose scale is this.scale().
- **scale**()
  Returns the scale of this number.
- **setScale**(int)
  Returns a BigDecimal whose scale is the specified value, and whose value is exactly equal to this number's.
- **setScale**(int, int)
  Returns a BigDecimal whose scale is the specified value, and whose integer value is determined by multiplying or dividing this BigDecimal's integer value by the appropriate power of ten to maintain the overall value.
- **signum**()
  Returns the signum function of this number (i.e., -1, 0 or 1 as the value of this number is negative, zero or positive).
- **subtract**(BigDecimal)
  Returns a BigDecimal whose value is (this - val), and whose scale is MAX(this.scale(), val.scale).
- **toBigInteger**()
  Converts this number to a BigInteger.
- **toString**()
  Returns the string representation of this number.
- **valueOf**(long)
  Returns a BigDecimal with the given value and a scale of zero.
- **valueOf**(long, int)
  Returns a BigDecimal with a value of (val/10**scale).

Variables

ROUND_UP

```
public final static int ROUND_UP
```

Always increment the digit prior to a non-zero discarded fraction. Note that this rounding mode never decreases the magnitude. (Rounds away from zero.)

ROUND_DOWN

```
public final static int ROUND_DOWN
```

Never increment the digit prior to a discarded fraction (i.e., truncate). Note that this rounding mode never increases the magnitude. (Rounds towards zero.)

ROUND_CEILING

```
public final static int ROUND_CEILING
```

If the BigDecimal is positive, behave as for ROUND_UP; if negative, behave as for ROUND_DOWN. Note that this rounding mode never decreases the value. (Rounds towards positive infinity.)

ROUND_FLOOR

```
public final static int ROUND_FLOOR
```

If the BigDecimal is positive, behave as for ROUND_DOWN; if negative behave as for ROUND_UP. Note that this rounding mode never increases the value. (Rounds towards negative infinity.)

ROUND_HALF_UP

```
public final static int ROUND_HALF_UP
```

Behave as for ROUND_UP if the discarded fraction is >= .5; otherwise, behave as for ROUND_DOWN. (Rounds towards "nearest neighbor" unless both neighbors are equidistant, in which case rounds up.)

ROUND_HALF_DOWN

```
public final static int ROUND_HALF_DOWN
```

Behave as for ROUND_UP if the discarded fraction is > .5; otherwise, behave as for ROUND_DOWN. (Rounds towards "nearest neighbor" unless both neighbors are equidistant, in which case rounds down.)

ROUND_HALF_EVEN

```
public final static int ROUND_HALF_EVEN
```

Behave as for ROUND_HALF_UP if the digit to the left of the discarded fraction is odd; behave as for ROUND_HALF_DOWN if it's even. (Rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, rounds towards the even neighbor.)

ROUND_UNNECESSARY

```
public final static int ROUND_UNNECESSARY
```

This "pseudo-rounding-mode" is actually an assertion that the requested operation has an exact result, hence no rounding is necessary.

Constructors

● **BigDecimal**

```
public BigDecimal(String val) throws NumberFormatException
```

Constructs a BigDecimal from a string containing an optional minus sign followed by a sequence of zero or more decimal digits, optionally followed by a fraction, which consists of a decimal point followed by zero or more decimal digits. The string must contain at least one digit in the integer or fractional part. The scale of the resulting BigDecimal will be the number of digits to the right of the decimal point in the string, or 0 if the string contains no decimal point. The character-to-digit mapping is provided by Character.digit. Any extraneous characters (including whitespace) will result in a NumberFormatException.

● **BigDecimal**

```
public BigDecimal(double val) throws NumberFormatException
```

Translates a double into a BigDecimal. The scale of the BigDecimal is the smallest value such that (10**scale * val) is an integer. A double whose value is -infinity, +infinity or NaN will result in a NumberFormatException.

● **BigDecimal**

```
public BigDecimal(BigInteger val)
```

Translates a BigInteger into a BigDecimal. The scale of the BigDecimal is zero.

● **BigDecimal**

```
public BigDecimal(BigInteger val,
                  int scale) throws NumberFormatException
```

Translates a BigInteger and a scale into a BigDecimal. The value of the BigDecimal is (BigInteger/10**scale). A negative scale will result in a NumberFormatException.

Methods

● **valueOf**

```
public static BigDecimal valueOf(long val,
                                 int scale) throws NumberFormatException
```

Returns a BigDecimal with a value of (val/10**scale). This factory is provided in preference to a (long) constructor because it allows for reuse of frequently used BigDecimals (like 0 and 1), obviating the need for exported constants. A negative scale will result in a NumberFormatException.

valueOf

```
public static BigDecimal valueOf(long val)
```

Returns a BigDecimal with the given value and a scale of zero. This factory is provided in preference to a (long) constructor because it allows for reuse of frequently used BigDecimals (like 0 and 1), obviating the need for exported constants.

add

```
public BigDecimal add(BigDecimal val)
```

Returns a BigDecimal whose value is (this + val), and whose scale is MAX(this.scale(), val.scale).

subtract

```
public BigDecimal subtract(BigDecimal val)
```

Returns a BigDecimal whose value is (this - val), and whose scale is MAX(this.scale(), val.scale).

multiply

```
public BigDecimal multiply(BigDecimal val)
```

Returns a BigDecimal whose value is (this * val), and whose scale is this.scale() + val.scale.

divide

```
public BigDecimal divide(BigDecimal val,
                         int scale,
                         int roundingMode) throws ArithmeticException, IllegalArgu
```

Returns a BigDecimal whose value is (this / val), and whose scale is as specified. If rounding must be performed to generate a result with the given scale, the specified rounding mode is applied. Throws an ArithmeticException if val == 0 or scale <0. Throws an IllegalArgumentException if roundingMode does not represent a valid rounding mode. dl> **divide**

```
public BigDecimal divide(BigDecimal val,
                         int roundingMode) throws ArithmeticException, Illega
```

Returns a BigDecimal whose value is (this / val), and whose scale is this.scale(). If rounding must be performed to generate a result with the given scale, the specified rounding mode is applied. Throws an ArithmeticException if val == 0. Throws an IllegalArgumentException if roundingMode does not represent a valid rounding mode.

abs

```
public BigDecimal abs()
```

Returns a BigDecimal whose value is the absolute value of this number, and whose scale is this.scale().

negate

```
public BigDecimal negate()
```

Returns a BigDecimal whose value is -1 * this, and whose scale is this.scale().

signum

```
public int signum()
```

Returns the signum function of this number (i.e., -1, 0 or 1 as the value of this number is negative, zero or positive).

scale

```
public int scale()
```

Returns the scale of this number.

setScale

```
public BigDecimal setScale(int scale,
                           int roundingMode) throws ArithmeticException, Illeg
```

Returns a BigDecimal whose scale is the specified value, and whose integer value is determined by multiplying or dividing this BigDecimal's integer value by the appropriate power of ten to maintain the overall value. If the scale is reduced by the operation, the integer value must be divided (rather than multiplied), and precision may be lost; in this case, the specified rounding mode is applied to the division. Throws an ArithmeticException if scale is negative. Throws an IllegalArgumentException if roundingMode does not represent a valid rounding mode.

setScale

```
public BigDecimal setScale(int scale) throws ArithmeticException, IllegalArgu
```

Returns a BigDecimal whose scale is the specified value, and whose value is exactly equal to this number's. Throws an ArithmeticException if this is not possible. This call is typically used to increase the scale, in which case it is guaranteed that there exists a BigDecimal of the specified scale and the correct value. The call can also be used to reduce the scale if the caller knows that the number has sufficiently many zeros at the end of its fractional part (i.e., factors of ten in its integer value) to allow for the rescaling without loss of precision. Note that this call returns the same result as the two argument version of setScale, but saves the caller the trouble of specifying a rounding mode in cases where it is irrelevant.

movePointLeft

```
public BigDecimal movePointLeft(int n)
```

Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left. If n is non-negative, the call merely adds n to the scale. If n is negative, the call is equivalent to movePointRight(-n). (The BigDecimal returned by this call has value (this * 10**-n) and scale MAX(this.scale()+n, 0).)

● movePointRight

```
public BigDecimal movePointRight(int n)
```

Moves the decimal point the specified number of places to the right. If this number's scale is >= n, the call merely subtracts n from the scale; otherwise, it sets the scale to zero, and multiplies the integer value by 10 ** (n - this.scale). If n is negative, the call is equivalent to movePointLeft(-n). (The BigDecimal returned by this call has value (this * 10**n) and scale MAX(this.scale()-n, 0).)

● compareTo

```
public int compareTo(BigDecimal val)
```

Returns -1, 0 or 1 as this number is less than, equal to, or greater than val. Two BigDecimals that are equal in value but have a different scale (e.g., 2.0, 2.00) are considered equal by this method. This method is provided in preference to individual methods for each of the six boolean comparison operators (<,>, >=, !=, <=). The suggested idiom for performing these comparisons is: x.compareTo(y) op> 0), where is one of the six comparison operators.

● equals

```
public boolean equals(Object x)
```

Returns true iff x is a BigDecimal whose value is equal to this number. This method is provided so that BigDecimals can be used as hash keys. Unlike compareTo, this method considers two BigDecimals equal only if they are equal in value and scale.
Overrides:
equals in class Object

● min

```
public BigDecimal min(BigDecimal val)
```

Returns the BigDecimal whose value is the lesser of this and val. If the values are equal (as defined by the compareTo operator), either may be returned.

● max

```
public BigDecimal max(BigDecimal val)
```

Returns the BigDecimal whose value is the greater of this and val. If the values are equal (as defined by the compareTo operator), either may be returned.

● hashCode

```
public int hashCode()
```

Computes a hash code for this object. Note that two BigDecimals that are numerically equal but differ in scale (e.g., 2.0, 2.00) will not generally have the same hash code.
Overrides:
hashCode in class Object

● toString

```
public String toString()
```

Returns the string representation of this number. The digit-to- character mapping provided by Character.forDigit is used. The minus sign and decimal point are used to indicate sign and scale. (This representation is compatible with the (String, int) constructor.)
Overrides:
toString in class Object

● toBigInteger

```
public BigInteger toBigInteger()
```

Converts this number to a BigInteger. Standard narrowing primitive conversion as per The Java Language Specification. In particular, note that any fractional part of this number will be truncated.

● intValue

```
public int intValue()
```

Converts this number to an int. Standard narrowing primitive conversion as per The Java Language Specification. In particular, note that any fractional part of this number will be truncated.
Overrides:
intValue in class Number

● longValue

```
public long longValue()
```

Converts this number to a long. Standard narrowing primitive conversion as per The Java Language Specification. In particular, note that any fractional part of this number will be truncated.
Overrides:
longValue in class Number

● floatValue

```
public float floatValue()
```

Converts this number to a float. Similar to the double-to-float narrowing primitive conversion defined in The Java Language Specification: if the number has too great a magnitude to represent as a float, it will be converted to infinity or negative infinity, as

appropriate.
Overrides:
floatValue in class Number

doubleValue

```
public double doubleValue()
```

Converts the number to a double. Similar to the double-to-float narrowing primitive conversion defined in The Java Language Specification: if the number has too great a magnitude to represent as a double, it will be converted to infinity or negative infinity, as appropriate.
Overrides:
doubleValue in class Number

Class java.math.BigInteger

```
java.lang.Object
   |
   +----java.lang.Number
           |
           +----java.math.BigInteger
```

---

public class **BigInteger**
extends Number

This version of BigInteger is just a wrapper class for long and its purpose is to only to support a JDK 1.0.2 version of BigDecimal.

See Also:
BigDecimal

Constructor Index

- **BigInteger**(String)
  Translates a string containing an optional minus sign followed by a sequence of one or more decimal digits into a BigInteger.
- **BigInteger**(String, int)
  Translates a string containing an optional minus sign followed by a sequence of one or more digits in the specified radix into a BigInteger.

Method Index

- **abs**()
  Returns a BigInteger whose value is the absolute value of this number.
- **add**(BigInteger)
  Returns a BigInteger whose value is (this + val).
- **compareTo**(BigInteger)
  Returns -1, 0 or 1 as this number is less than, equal to, or greater than val.
- **divide**(BigInteger)
  Returns a BigInteger whose value is (this / val).
- **divideAndRemainder**(BigInteger)
  Returns an array of two BigIntegers.
- **doubleValue**()
  Converts the number to a double.
- **equals**(Object)
  Returns true iff x is a BigInteger whose value is equal to this number.
- **floatValue**()
  Converts this number to a float.
- **hashCode**()
  Computes a hash code for this object.
- **intValue**()
  Converts this number to an int.
- **longValue**()
  Converts this number to a long.
- **max**(BigInteger)
  Returns the BigInteger whose value is the greater of this and val.
- **min**(BigInteger)
  Returns the BigInteger whose value is the lesser of this and val.
- **multiply**(BigInteger)
  Returns a BigInteger whose value is (this * val).
- **negate**()
  Returns a BigInteger whose value is (-1 * this).
- **pow**(int)
  Returns a BigInteger whose value is (this ** exponent).
- **remainder**(BigInteger)
  Returns a BigInteger whose value is (this % val).
- **signum**()
  Returns the signum function of this number (i.e., -1, 0 or 1 as the value of this number is negative, zero or positive).
- **subtract**(BigInteger)
  Returns a BigInteger whose value is (this - val).
- **testBit**(int)
  Returns true iff the designated bit is set.
- **toString**()
  Returns the string representation of this number, radix 10.
- **toString**(int)

Returns the string representation of this number in the given radix.

valueOf(long)
Returns a BigInteger with the specified value.

Constructors

BigInteger

```
public BigInteger(String val,
                  int radix) throws NumberFormatException
```

Translates a string containing an optional minus sign followed by a sequence of one or more digits in the specified radix into a BigInteger. The character-to-digit mapping is provided by Character.digit. Any extraneous characters (including whitespace), or a radix outside the range from Character.MIN_RADIX(2) to Character.MAX_RADIX(36), inclusive, will result in a NumberFormatException.

BigInteger

```
public BigInteger(String val) throws NumberFormatException
```

Translates a string containing an optional minus sign followed by a sequence of one or more decimal digits into a BigInteger. The character-to-digit mapping is provided by Character.digit. Any extraneous characters (including whitespace) will result in a NumberFormatException.

Methods

valueOf

```
public static BigInteger valueOf(long val)
```

Returns a BigInteger with the specified value. This factory is provided in preference to a (long) constructor because it allows for reuse of frequently used BigIntegers (like 0 and 1), obviating the need for exported constants.

add

```
public BigInteger add(BigInteger val) throws ArithmeticException
```

Returns a BigInteger whose value is (this + val).

subtract

```
public BigInteger subtract(BigInteger val)
```

Returns a BigInteger whose value is (this - val).

multiply

```
public BigInteger multiply(BigInteger val) throws ArithmeticException
```

Returns a BigInteger whose value is (this * val).

divide

```
public BigInteger divide(BigInteger val) throws ArithmeticException
```

Returns a BigInteger whose value is (this / val). Throws an ArithmeticException if val == 0.

remainder

```
public BigInteger remainder(BigInteger val) throws ArithmeticException
```

Returns a BigInteger whose value is (this % val). Throws an ArithmeticException if val == 0.

divideAndRemainder

```
public BigInteger[] divideAndRemainder(BigInteger val) throws ArithmeticException
```

Returns an array of two BigIntegers. The first ([0]) element of the return value is the quotient (this / val), and the second ([1]) element is the remainder (this % val). Throws an ArithmeticException if val == 0.

testBit

```
public boolean testBit(int n) throws ArithmeticException
```

Returns true iff the designated bit is set. (Computes ((this & (1< **pow**

```
public BigInteger pow(int exponent) throws ArithmeticException
```

Returns a BigInteger whose value is (this ** exponent). Throws an ArithmeticException if exponent <0 as the operation would yield a non-integer value). Note that exponent is an integer rather than a BigInteger. dl> **abs**

```
public BigInteger abs()
```

Returns a BigInteger whose value is the absolute value of this number.

negate

```
public BigInteger negate()
```

Returns a BigInteger whose value is (-1 * this).

signum

```
public int signum()
```

Returns the signum function of this number (i.e., -1, 0 or 1 as the value of this number is negative, zero or positive).

● **compareTo**

```
public int compareTo(BigInteger val)
```

Returns -1, 0 or 1 as this number is less than, equal to, or greater than val. This method is provided in preference to individual methods for each of the six boolean comparison operators (<,>, >=, !=, <=). The suggested idiom for performing these comparisons is: x.compareTo(y) op> 0), where is one of the six comparison operators.

● **equals**

```
public boolean equals(Object x)
```

Returns true iff x is a BigInteger whose value is equal to this number. This method is provided so that BigIntegers can be used as hash keys.
Overrides:
equals in class Object

● **min**

```
public BigInteger min(BigInteger val)
```

Returns the BigInteger whose value is the lesser of this and val. If the values are equal, either may be returned.

● **max**

```
public BigInteger max(BigInteger val)
```

Returns the BigInteger whose value is the greater of this and val. If the values are equal, either may be returned.

● **hashCode**

```
public int hashCode()
```

Computes a hash code for this object.
Overrides:
hashCode in class Object

● **toString**

```
public String toString(int radix)
```

Returns the string representation of this number in the given radix. If the radix is outside the range from Character.MIN_RADIX(2) to Character.MAX_RADIX(36) inclusive, it will default to 10 (as is the case for Integer.toString). The digit-to-character mapping provided by Character.forDigit is used, and a minus sign is prepended if appropriate. (This representation is compatible with the (String, int) constructor.)

● toString

```
public String toString()
```

Returns the string representation of this number, radix 10. The digit-to-character mapping provided by Character.forDigit is used, and a minus sign is prepended if appropriate. (This representation is compatible with the (String) constructor, and allows for string concatenation with Java's + operator.)

Overrides:
toString in class Object

● intValue

```
public int intValue()
```

Converts this number to an int. Standard narrowing primitive conversion as per The Java Language Specification.

Overrides:
intValue in class Number

● longValue

```
public long longValue()
```

Converts this number to a long. Standard narrowing primitive conversion as per The Java Language Specification.

Overrides:
longValue in class Number

● floatValue

```
public float floatValue()
```

Converts this number to a float. Similar to the double-to-float narrowing primitive conversion defined in The Java Language Specification: if the number has too great a magnitude to represent as a float, it will be converted to infinity or negative infinity, as appropriate.

Overrides:
floatValue in class Number

● doubleValue

```
public double doubleValue()
```

Converts the number to a double. Similar to the double-to-float narrowing primitive conversion defined in The Java Language Specification: if the number has too great a magnitude to represent as a double, it will be converted to infinity or negative infinity,

as appropriate.
Overrides:
doubleValue in class Number

Class java.sql.Date

```
java.lang.Object
   |
   +----java.util.Date
           |
           +----java.sql.Date
```

---

public class **Date**
extends Date

This class is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL DATE value. It adds formatting and parsing operations to support the JDBC escape syntax for date values.

Constructor Index

- **Date**(int, int, int)
  Construct a Date
- **Date**(long)
  Construct a Date using a milliseconds time value

Method Index

- **getHours**()

- **getMinutes**()

- **getSeconds**()

- **setHours**(int)

- **setMinutes**(int)

- **setSeconds**(int)

- **setTime**(long)
  Set a Date using a milliseconds time value
- **toString**()
  Format a date in JDBC date escape format
- **valueOf**(String)
  Convert a string in JDBC date escape format to a Date value

Constructors

Date

```
public Date(int year,
            int month,
            int day)
```

Construct a Date
Parameters:
year - year-1900
month - 0 to 11
day - 1 to 31

Date

```
public Date(long date)
```

Construct a Date using a milliseconds time value
Parameters:
date - milliseconds since January 1, 1970, 00:00:00 GMT

Methods

setTime

```
public void setTime(long date)
```

Set a Date using a milliseconds time value
Parameters:
date - milliseconds since January 1, 1970, 00:00:00 GMT
Overrides:
setTime in class Date

valueOf

```
public static Date valueOf(String s)
```

Convert a string in JDBC date escape format to a Date value
Parameters:

s - date in format "yyyy-mm-dd"

Returns:

corresponding Date

● toString

```
public String toString()
```

Format a date in JDBC date escape format

Returns:

a String in yyyy-mm-dd format

Overrides:

toString in class Date

● getHours

```
public int getHours()
```

Overrides:

getHours in class Date

● getMinutes

```
public int getMinutes()
```

Overrides:

getMinutes in class Date

● getSeconds

```
public int getSeconds()
```

Overrides:

getSeconds in class Date

● setHours

```
public void setHours(int i)
```

Overrides:

setHours in class Date

● setMinutes

```
public void setMinutes(int i)
```

Overrides:

setMinutes in class Date

● setSeconds

```
public void setSeconds(int i)
```

Overrides:
setSeconds in class Date

Class java.sql.DriverManager

```
java.lang.Object
   |
   +----java.sql.DriverManager
```

---

public class **DriverManager**
extends Object

The DriverManager provides a basic service for managing a set of JDBC drivers.

As part of its initialization, the DriverManager class will attempt to load the driver classes referenced in the "jdbc.drivers" system property. This allows a user to customize the JDBC Drivers used by their applications. For example in your ~/.hotjava/properties file you might specify: `jdbc.drivers=foo.bah.Driver:wombat.sql.Driver:bad.taste.ourDriver` A program can also explicitly load JDBC drivers at any time. For example, the my.sql.Driver is loaded with the following statement: `Class.forName("my.sql.Driver");`

When getConnection is called the DriverManager will attempt to locate a suitable driver from amongst those loaded at initialization and those loaded explicitly using the same classloader as the current applet or application.

See Also:
Driver, Connection

Method Index

- **deregisterDriver**(Driver)
  Drop a Driver from the DriverManager’s list.
- **getConnection**(String)
  Attempt to establish a connection to the given database URL.
- **getConnection**(String, Properties)
  Attempt to establish a connection to the given database URL.
- **getConnection**(String, String, String)
  Attempt to establish a connection to the given database URL.
- **getDriver**(String)

Attempt to locate a driver that understands the given URL.

- **getDrivers**()
  Return an Enumeration of all the currently loaded JDBC drivers which the current caller has access to.
- **getLoginTimeout**()
  Get the maximum time in seconds that all drivers can wait when attempting to log in to a database.
- **getLogStream**()
  Get the logging/tracing PrintStream that is used by the DriverManager and all drivers.
- **println**(String)
  Print a message to the current JDBC log stream
- **registerDriver**(Driver)
  A newly loaded driver class should call registerDriver to make itself known to the DriverManager.
- **setLoginTimeout**(int)
  Set the maximum time in seconds that all drivers can wait when attempting to log in to a database.
- **setLogStream**(PrintStream)
  Set the logging/tracing PrintStream that is used by the DriverManager and all drivers.

Methods

getConnection

```
public static synchronized Connection getConnection(String url,
                                                     Properties info) throws SQLExc
```

Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

url - a database url of the form jdbc:*subprotocol*:*subname*
info - a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included

Returns:

a Connection to the URL

getConnection

```
public static synchronized Connection getConnection(String url,
                                                     String user,
                                                     String password) throws SQLExc
```

Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

url - a database url of the form jdbc:*subprotocol*:*subname*
user - the database user on whose behalf the Connection is being made
password - the user’s password

Returns:

a Connection to the URL

getConnection

```
public static synchronized Connection getConnection(String url) throws SQLExceptio
```

Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:
url - a database url of the form jdbc:*subprotocol*:*subname*

Returns:
a Connection to the URL

getDriver

```
public static Driver getDriver(String url) throws SQLException
```

Attempt to locate a driver that understands the given URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:
url - a database url of the form jdbc:*subprotocol*:*subname*

Returns:
a Driver that can connect to the URL

registerDriver

```
public static synchronized void registerDriver(Driver driver) throws SQLException
```

A newly loaded driver class should call registerDriver to make itself known to the DriverManager.

Parameters:
driver - the new JDBC Driver

deregisterDriver

```
public static void deregisterDriver(Driver driver) throws SQLException
```

Drop a Driver from the DriverManager's list. Applets can only deregister Drivers from their own classloader.

Parameters:
driver - the JDBC Driver to drop

getDrivers

```
public static Enumeration getDrivers()
```

Return an Enumeration of all the currently loaded JDBC drivers which the current caller has access to.

Note: The classname of a driver can be found using `d.getClass().getName()`

Returns:
the list of JDBC Drivers loaded by the caller's class loader

● setLoginTimeout

```
public static void setLoginTimeout(int seconds)
```

Set the maximum time in seconds that all drivers can wait when attempting to log in to a database.
Parameters:
seconds - the driver login time limit

● getLoginTimeout

```
public static int getLoginTimeout()
```

Get the maximum time in seconds that all drivers can wait when attempting to log in to a database.
Returns:
the driver login time limit

● setLogStream

```
public static void setLogStream(PrintStream out)
```

Set the logging/tracing PrintStream that is used by the DriverManager and all drivers.
Parameters:
out - the new logging/tracing PrintStream; to disable, set to null

● getLogStream

```
public static PrintStream getLogStream()
```

Get the logging/tracing PrintStream that is used by the DriverManager and all drivers.
Returns:
the logging/tracing PrintStream; if disabled, is null

● println

```
public static void println(String message)
```

Print a message to the current JDBC log stream
Parameters:
message - a log or tracing message

Class java.sql.DriverPropertyInfo

```
java.lang.Object
   |
   +----java.sql.DriverPropertyInfo
```

---

public class **DriverPropertyInfo**
extends Object

The DriverPropertyInfo class is only of interest to advanced programmers who need to interact with a Driver via getDriverProperties to discover and supply properties for connections.

Variable Index

- **choices**
  If the value may be selected from a particular set of values, then this is an array of the possible values.
- **description**
  A brief description of the property.
- **name**
  The name of the property.
- **required**
  "required" is true if a value must be supplied for this property during Driver.connect.
- **value**
  "value" specifies the current value of the property, based on a combination of the information supplied to getPropertyInfo, the Java environment, and driver supplied default values.

Constructor Index

- **DriverPropertyInfo**(String, String)
  Constructor a DriverPropertyInfo with a name and value; other members default to their initial values.

Variables

name

```
public String name
```

The name of the property.

description

```
public String description
```

A brief description of the property. This may be null.

required

```
public boolean required
```

"required" is true if a value must be supplied for this property during Driver.connect. Otherwise the property is optional.

value

```
public String value
```

"value" specifies the current value of the property, based on a combination of the information supplied to getPropertyInfo, the Java environment, and driver supplied default values. This may be null if no value is known.

choices

```
public String choices[]
```

If the value may be selected from a particular set of values, then this is an array of the possible values. Otherwise it should be null.

Constructors

DriverPropertyInfo

```
public DriverPropertyInfo(String name,
                          String value)
```

Constructor a DriverPropertyInfo with a name and value; other members default to their initial values.

Parameters:

name - the name of the property
value - the current value, which may be null

Class java.sql.Time

```
java.lang.Object
   |
   +----java.util.Date
           |
           +----java.sql.Time
```

---

public class **Time**
extends Date

This class is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIME value. It adds formatting and parsing operations to support the JDBC escape syntax for time values.

Constructor Index

- **Time**(int, int, int)
  Construct a Time Object
- **Time**(long)
  Construct a Time using a milliseconds time value

Method Index

- **getDate**()

- **getDay**()

- **getMonth**()

- **getYear**()

- **setDate**(int)

- **setMonth**(int)

- **setTime**(long)
  Set a Time using a milliseconds time value
- **setYear**(int)

- **toString**()
  Format a time in JDBC date escape format
- **valueOf**(String)
  Convert a string in JDBC time escape format to a Time value

Constructors

Time

```
public Time(int hour,
            int minute,
            int second)
```

Construct a Time Object
Parameters:
hour - 0 to 23

minute - 0 to 59
second - 0 to 59

Time

```
public Time(long time)
```

Construct a Time using a milliseconds time value
Parameters:
time - milliseconds since January 1, 1970, 00:00:00 GMT

Methods

setTime

```
public void setTime(long time)
```

Set a Time using a milliseconds time value
Parameters:
time - milliseconds since January 1, 1970, 00:00:00 GMT
Overrides:
setTime in class Date

valueOf

```
public static Time valueOf(String s)
```

Convert a string in JDBC time escape format to a Time value
Parameters:
s - time in format "hh:mm:ss"
Returns:
corresponding Time

toString

```
public String toString()
```

Format a time in JDBC date escape format
Returns:
a String in hh:mm:ss format
Overrides:
toString in class Date

getYear

```
public int getYear()
```

Overrides:
getYear in class Date

● **getMonth**

```
public int getMonth()
```

Overrides:
getMonth in class Date

● **getDay**

```
public int getDay()
```

Overrides:
getDay in class Date

● **getDate**

```
public int getDate()
```

Overrides:
getDate in class Date

● **setYear**

```
public void setYear(int i)
```

Overrides:
setYear in class Date

● **setMonth**

```
public void setMonth(int i)
```

Overrides:
setMonth in class Date

● **setDate**

```
public void setDate(int i)
```

Overrides:
setDate in class Date

Class java.sql.Timestamp

```
java.lang.Object
   |
   +----java.util.Date
```

```
          |
          +----java.sql.Timestamp
```

---

public class **Timestamp**
extends Date

This class is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIMESTAMP value. It adds the ability to hold the SQL TIMESTAMP nanos value and provides formatting and parsing operations to support the JDBC escape syntax for timestamp values.

Note: This type is a composite of a java.util.Date and a separate nanos value. Only integral seconds are stored in the java.util.Date component. The fractional seconds - the nanos - are separate. The getTime method will only return integral seconds. If a time value that includes the fractional seconds is desired you must convert nanos to milliseconds (nanos/1000000) and add this to the getTime value. Also note that the hashcode() method uses the underlying java.util.Data implementation and therefore does not include nanos in its computation.

Constructor Index

- **Timestamp**(int, int, int, int, int, int, int)
  Construct a Timestamp Object
- **Timestamp**(long)
  Construct a Timestamp using a milliseconds time value.

Method Index

- **after**(Timestamp)
  Is this timestamp later than the timestamp argument?
- **before**(Timestamp)
  Is this timestamp earlier than the timestamp argument?
- **equals**(Timestamp)
  Test Timestamp values for equality
- **getNanos**()
  Get the Timestamp’s nanos value
- **setNanos**(int)
  Set the Timestamp’s nanos value
- **toString**()
  Format a timestamp in JDBC timestamp escape format
- **valueOf**(String)
  Convert a string in JDBC timestamp escape format to a Timestamp value

Constructors

Timestamp

```
public Timestamp(int year,
                 int month,
                 int date,
                 int hour,
                 int minute,
                 int second,
                 int nano)
```

Construct a Timestamp Object

Parameters:

year - year-1900
month - 0 to 11
day - 1 to 31
hour - 0 to 23
minute - 0 to 59
second - 0 to 59
nano - 0 to 999,999,999

Timestamp

```
public Timestamp(long time)
```

Construct a Timestamp using a milliseconds time value. The integral seconds are stored in the underlying date value; the fractional seconds are stored in the nanos value.

Parameters:

time - milliseconds since January 1, 1970, 00:00:00 GMT

Methods

valueOf

```
public static Timestamp valueOf(String s)
```

Convert a string in JDBC timestamp escape format to a Timestamp value

Parameters:

s - timestamp in format "yyyy-mm-dd hh:mm:ss.fffffffff"

Returns:

corresponding Timestamp

toString

```
public String toString()
```

Format a timestamp in JDBC timestamp escape format

Returns:
a String in yyyy-mm-dd hh:mm:ss.fffffffff format
Overrides:
toString in class Date

● getNanos

```
public int getNanos()
```

Get the Timestamp’s nanos value
Returns:
the Timestamp’s fractional seconds component

● setNanos

```
public void setNanos(int n)
```

Set the Timestamp’s nanos value
Parameters:
n - the new fractional seconds component

● equals

```
public boolean equals(Timestamp ts)
```

Test Timestamp values for equality
Parameters:
ts - the Timestamp value to compare with

● before

```
public boolean before(Timestamp ts)
```

Is this timestamp earlier than the timestamp argument?
Parameters:
ts - the Timestamp value to compare with

● after

```
public boolean after(Timestamp ts)
```

Is this timestamp later than the timestamp argument?
Parameters:
ts - the Timestamp value to compare with

Class java.sql.Types

```
java.lang.Object
   |
   +----java.sql.Types
```

---

public class **Types**
extends Object

This class defines constants that are used to identify SQL types. The actual type constant values are equivalent to those in XOPEN.

Variable Index

- **BIGINT**
- **BINARY**
- **BIT**
- **CHAR**
- **DATE**
- **DECIMAL**
- **DOUBLE**
- **FLOAT**
- **INTEGER**
- **LONGVARBINARY**
- **LONGVARCHAR**
- **NULL**
- **NUMERIC**
- **OTHER**
  OTHER indicates that the SQL type is database specific and gets mapped to a Java object which can be accessed via getObject and setObject.
- **REAL**

- **SMALLINT**
- **TIME**
- **TIMESTAMP**
- **TINYINT**
- **VARBINARY**
- **VARCHAR**

Variables

- **BIT**

```
public final static int BIT
```

- **TINYINT**

```
public final static int TINYINT
```

- **SMALLINT**

```
public final static int SMALLINT
```

- **INTEGER**

```
public final static int INTEGER
```

- **BIGINT**

```
public final static int BIGINT
```

- **FLOAT**

```
public final static int FLOAT
```

- **REAL**

```
public final static int REAL
```

- **DOUBLE**

```
public final static int DOUBLE
```

- **NUMERIC**

```
public final static int NUMERIC
```

● DECIMAL

```
public final static int DECIMAL
```

● CHAR

```
public final static int CHAR
```

● VARCHAR

```
public final static int VARCHAR
```

● LONGVARCHAR

```
public final static int LONGVARCHAR
```

● DATE

```
public final static int DATE
```

● TIME

```
public final static int TIME
```

● TIMESTAMP

```
public final static int TIMESTAMP
```

● BINARY

```
public final static int BINARY
```

● VARBINARY

```
public final static int VARBINARY
```

● LONGVARBINARY

```
public final static int LONGVARBINARY
```

● NULL

```
public final static int NULL
```

● OTHER

```
public final static int OTHER
```

OTHER indicates that the SQL type is database specific and gets mapped to a Java object which can be accessed via getObject and setObject.

Class java.sql.DataTruncation

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
                           |
                           +----java.sql.SQLWarning
                                   |
                                   +----java.sql.DataTruncation
```

---

public class **DataTruncation**
extends SQLWarning

When JDBC unexpectedly truncates a data value, it reports a DataTruncation warning (on reads) or throws a DataTruncation exception (on writes).

The SQLstate for a DataTruncation is "01004".

Constructor Index

- **DataTruncation**(int, boolean, boolean, int, int)
  Create a DataTruncation object.

Method Index

- **getDataSize**()
  Get the number of bytes of data that should have been transferred.
- **getIndex**()
  Get the index of the column or parameter that was truncated.
- **getParameter**()
  Is this a truncated parameter value?
- **getRead**()
  Was this a read truncation?
- **getTransferSize**()
  Get the number of bytes of data actually transferred.

Constructors

DataTruncation

```
public DataTruncation(int index,
                      boolean parameter,
                      boolean read,
                      int dataSize,
                      int transferSize)
```

Create a DataTruncation object. The SQLState is initialized to 01004, the reason is set to "Data truncation" and the vendorCode is set to the SQLException default.

Parameters:

index - The index of the parameter or column value
parameter - true if a parameter value was truncated
read - true if a read was truncated
dataSize - the original size of the data
transferSize - the size after truncation

Methods

getIndex

```
public int getIndex()
```

Get the index of the column or parameter that was truncated.

This may be -1 if the column or parameter index is unknown, in which case the "parameter" and "read" fields should be ignored.

Returns:

the index of the truncated paramter or column value.

getParameter

```
public boolean getParameter()
```

Is this a truncated parameter value?

Returns:

True if the value was a parameter; false if it was a column value.

getRead

```
public boolean getRead()
```

Was this a read truncation?

Returns:

True if the value was truncated when read from the database; false if the data was truncated on a write.

getDataSize

```
public int getDataSize()
```

Get the number of bytes of data that should have been transferred. This number may be approximate if data conversions were being performed. The value may be "-1" if the size is unknown.
Returns:
the number of bytes of data that should have been transferred

getTransferSize

```
public int getTransferSize()
```

Get the number of bytes of data actually transferred. The value may be "-1" if the size is unknown.
Returns:
the number of bytes of data actually transferred

Class java.sql.SQLException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
```

---

public class **SQLException**
extends Exception

The SQLException class provides information on a database access error.

Each SQLException provides several kinds of information:

- a string describing the error. This is used as the Java Exception message, and is available via the getMesage() method
- A "SQLstate" string which follows the XOPEN SQLstate conventions. The values of the SQLState string as described in the XOPEN SQL spec.
- An integer error code that is vendor specific. Normally this will be the actual error code returned by the underlying database.
- A chain to a next Exception. This can be used to provided additional error information.

Constructor Index

- **SQLException**()
  Construct an SQLException; reason defaults to null, SQLState defaults to null and vendorCode defaults to 0.
- **SQLException**(String)
  Construct an SQLException with a reason; SQLState defaults to null and vendorCode defaults to 0.
- **SQLException**(String, String)
  Construct an SQLException with a reason and SQLState; vendorCode defaults to 0.
- **SQLException**(String, String, int)
  Construct a fully-specified SQLException

Method Index

- **getErrorCode**()
  Get the vendor specific exception code
- **getNextException**()
  Get the exception chained to this one.
- **getSQLState**()
  Get the SQLState
- **setNextException**(SQLException)
  Add an SQLException to the end of the chain.

Constructors

SQLException

```
public SQLException(String reason,
                    String SQLState,
                    int vendorCode)
```

Construct a fully-specified SQLException
Parameters:
reason - a description of the exception
SQLState - an XOPEN code identifying the exception
vendorCode - a database vendor specific exception code

SQLException

```
public SQLException(String reason,
                    String SQLState)
```

Construct an SQLException with a reason and SQLState; vendorCode defaults to 0.
Parameters:
reason - a description of the exception
SQLState - an XOPEN code identifying the exception

SQLException

```
public SQLException(String reason)
```

Construct an SQLException with a reason; SQLState defaults to null and vendorCode defaults to 0.
Parameters:
reason - a description of the exception

SQLException

```
public SQLException()
```

Construct an SQLException; reason defaults to null, SQLState defaults to null and vendorCode defaults to 0.

Methods

getSQLState

```
public String getSQLState()
```

Get the SQLState
Returns:
the SQLState value

getErrorCode

```
public int getErrorCode()
```

Get the vendor specific exception code
Returns:
the vendor’s error code

getNextException

```
public SQLException getNextException()
```

Get the exception chained to this one.
Returns:
the next SQLException in the chain, null if none

setNextException

```
public synchronized void setNextException(SQLException ex)
```

Add an SQLException to the end of the chain.
Parameters:
ex - the new end of the SQLException chain

Class java.sql.SQLWarning

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
                           |
                           +----java.sql.SQLWarning
```

public class **SQLWarning**
extends SQLException

The SQLWarning class provides information on a database access warnings. Warnings are silently chained to the object whose method caused it to be reported.

See Also:
getWarnings, getWarnings, getWarnings

Constructor Index

- **SQLWarning**()
  Construct an SQLWarning ; reason defaults to null, SQLState defaults to null and vendorCode defaults to 0.
- **SQLWarning**(String)
  Construct an SQLWarning with a reason; SQLState defaults to null and vendorCode defaults to 0.
- **SQLWarning**(String, String)
  Construct an SQLWarning with a reason and SQLState; vendorCode defaults to 0.
- **SQLWarning**(String, String, int)
  Construct a fully specified SQLWarning.

Method Index

- **getNextWarning**()
  Get the warning chained to this one
- **setNextWarning**(SQLWarning)
  Add an SQLWarning to the end of the chain.

Constructors

SQLWarning

```
public SQLWarning(String reason,
                  String SQLstate,
                  int vendorCode)
```

Construct a fully specified SQLWarning.
Parameters:
reason - a description of the warning
SQLState - an XOPEN code identifying the warning
vendorCode - a database vendor specific warning code

SQLWarning

```
public SQLWarning(String reason,
                  String SQLstate)
```

Construct an SQLWarning with a reason and SQLState; vendorCode defaults to 0.
Parameters:
reason - a description of the warning
SQLState - an XOPEN code identifying the warning

SQLWarning

```
public SQLWarning(String reason)
```

Construct an SQLWarning with a reason; SQLState defaults to null and vendorCode defaults to 0.
Parameters:
reason - a description of the warning

SQLWarning

```
public SQLWarning()
```

Construct an SQLWarning ; reason defaults to null, SQLState defaults to null and vendorCode defaults to 0.

Methods

getNextWarning

```
public SQLWarning getNextWarning()
```

Get the warning chained to this one
Returns:
the next SQLException in the chain, null if none

● setNextWarning

```
public void setNextWarning(SQLWarning w)
```

Add an SQLWarning to the end of the chain.

Parameters:

w - the new end of the SQLException chain
