PUBLIC
SQL Anywhere Server
Document Version: 17.01.0 – 2021-10-15

SQL Anywhere - XS JavaScript API Reference

THE BEST RUN SAP

Content

1 XS JavaScript Application Programming. 6

1.1 CallableStatement Class. 9

close() Method. 13

execute() Method. 14

getBigInt(Integer) Method. 14

getBlob(Integer) Method. 15

getClob(Integer) Method. 15

getDate(Integer) Method. 16

getDecimal(Integer) Method. 17

getDouble(Integer) Method. 17

getInteger(Integer) Method. 18

getMetaData() Method. 19

getMoreResults() Method. 19

getParameterMetaData() Method. 20

getReal(Integer) Method. 20

getResultSet() Method. 21

getSmallInt(Integer) Method. 21

getString(Integer) Method. 22

getText(Integer) Method. 22

getTime(Integer) Method. 23

getTimestamp(Integer) Method. 24

getTinyInt(Integer) Method. 24

getUnsigned(Integer) Method. 25

isClosed() Method. 26

setBigInt(Integer, Number) Method. 26

setBlob(Integer, Buffer) Method. 27

setBString(Integer, String) Method. 28

setClob(Integer, String) Method. 28

setDate Method. 29

setDecimal(Integer, Number) Method. 30

setDouble(Integer, Number) Method. 31

setInteger(Integer, Integer) Method. 32

setNull(Integer) Method. 32

setReal(Integer, Number) Method. 33

setSmallInt(Integer, Integer) Method. 33

setString(Integer, String) Method. 34

setText(Integer, String) Method. . . . 35
setTime Method. . . . 35
setTimestamp Method. . . . 37
setTinyInt(Integer, Integer) Method. . . . 39
setUnsigned(Integer, Integer) Method. . . . 40
1.2 Connection Class. . . . 40
close() Method. . . . 42
commit() Method. . . . 42
connect Method. . . . 43
disconnect() Method. . . . 45
isClosed() Method. . . . 46
prepareCall(String) Method. . . . 46
prepareStatement(String) Method. . . . 47
rollback() Method. . . . 47
setAutoCommit(Boolean) Method. . . . 48
1.3 ParameterMetaData Class. . . . 48
getParameterCount() Method. . . . 50
getParameterMode(Integer) Method. . . . 51
getParameterName(Integer) Method. . . . 51
getParameterType(Integer) Method. . . . 52
getParameterTypeName(Integer) Method. . . . 53
getPrecision(Integer) Method. . . . 53
getScale(Integer) Method. . . . 54
hasDefault(Integer) Method. . . . 54
isNullable(Integer) Method. . . . 55
isSigned(Integer) Method. . . . 56
1.4 PreparedStatement Class. . . . 56
addBatch() Method. . . . 60
close() Method. . . . 60
execute() Method. . . . 61
executeBatch() Method. . . . 61
executeQuery() Method. . . . 62
executeUpdate() Method. . . . 62
getMetaData() Method. . . . 63
getMoreResults() Method. . . . 63
getParameterMetaData() Method. . . . 63
getResultSet() Method. . . . 64
isClosed() Method. . . . 64
setBatchSize(Integer) Method. . . . 64
setBigInt(Integer, Number) Method. . . . 65
setBlob(Integer, Buffer) Method. . . . 66

setBString(Integer, String) Method. . . . 66
setClob(Integer, String) Method. . . . 67
setDate Method. . . . 67
setDecimal(Integer, Number) Method. . . . 69
setDouble(Integer, Number) Method. . . . 70
setInteger(Integer, Integer) Method. . . . 70
setNull(Integer) Method. . . . 71
setReal(Integer, Number) Method. . . . 72
setSmallInt(Integer, Integer) Method. . . . 72
setString(Integer, String) Method. . . . 73
setText(Integer, String) Method. . . . 73
setTime Method. . . . 74
setTimestamp Method. . . . 76
setTinyInt(Integer, Integer) Method. . . . 77
setUnsigned(Integer, Integer) Method. . . . 78
1.5 ResultSet Class. . . . 79
close() Method. . . . 81
getBigInt(Integer) Method. . . . 81
getBlob(Integer) Method. . . . 82
getClob(Integer) Method. . . . 83
getDate(Integer) Method. . . . 83
getDecimal(Integer) Method. . . . 84
getDouble(Integer) Method. . . . 85
getInteger(Integer) Method. . . . 85
getReal(Integer) Method. . . . 86
getSmallInt(Integer) Method. . . . 87
getString(Integer) Method. . . . 87
getText(Integer) Method. . . . 88
getTime(Integer) Method. . . . 89
getTimestamp(Integer) Method. . . . 89
getTinyInt(Integer) Method. . . . 90
getUnsigned(Integer) Method. . . . 91
isClosed() Method. . . . 91
next() Method. . . . 92
1.6 ResultSetMetaData Class. . . . 92
getCatalogName() Method. . . . 94
getColumnCount() Method. . . . 94
getColumnLabel(Integer) Method. . . . 95
getColumnName(Integer) Method. . . . 95
getColumnType(Integer) Method. . . . 96
getColumnTypeName(Integer) Method. . . . 96

getDisplaySize(Integer) Method. 97
getPrecision(Integer) Method. 97
getScale(Integer) Method. 98
getTableName(Integer) Method. 98

1 XS JavaScript Application Programming

The SQL Anywhere XS JavaScript driver can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

The SQL Anywhere XS JavaScript driver allows users to interact with the database from the JavaScript environment. The SQL Anywhere XS API is based on the SAP HANA XS JavaScript Database API provided by the SAP HANA XS engine. Drivers are available for various versions of Node.js.

The XS JavaScript driver also supports server-side applications written in JavaScript using SQL Anywhere external environment support.

The SQL Anywhere .XS JavaScript API reference is available in the *SQL Anywhere- XS JavaScript API Reference* at https://help.sap.com/viewer/78bac38a62a2496996d5a27467561290/LATEST/en-US.

i Note

In addition to the XS JavaScript driver, a lightweight, minimally-featured Node.js driver is available that can handle small result sets quite well. Node.js application programming using this driver is described elsewhere in the documentation. The lightweight driver is perfect for simple web applications that need a quick and easy connection to the database server to retrieve small result sets. However, if your JavaScript application needs to handle large results, have greater control, or have access to a fuller application programming interface, then you should use the XS JavaScript driver.

Node.js must be installed on your computer and the folder containing the Node.js executable should be included in your PATH. Node.js software is available at nodejs.org .

For Node.js to locate the driver, make sure that the NODE_PATH environment variable includes the location of the XS JavaScript driver. The following is an example for Microsoft Windows:

```
SET NODE_PATH=%SQLANY17%\Node
```

i Note

On macOS 10.11 or a later version, set the SQLANY_API_DLL environment variable to the full path for `libdbcapi_r.dylib`.

The following illustrates a simple Node.js application that uses the XS JavaScript driver.

```
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server      : 'demo',
             UserID      : 'DBA',
             Password    : 'sql'
           };
var conn = sqla.createConnection( cstr );
conn.connect();
console.log( 'Connected' );
stmt = conn.prepareStatement( "SELECT * FROM Customers" );
stmt.execute();
var result = stmt.getResultSet();
while ( result.next() )
{
    console.log( result.getString(1) +
             " " + result.getString(3) +
             " " + result.getString(2) );
}
conn.disconnect();
console.log( 'Disconnected' );
```

This program connects to the sample database, executes a SQL SELECT statement, and displays only the first three columns of the result set, namely the ID, GivenName, and Surname of each customer. It then disconnects from the database.

Suppose this JavaScript code was stored in the file `xs-sample.js`. To run this program, open a command prompt and execute the following statement. Make sure that the NODE_PATH environment variable is set appropriately.

```
node xs-sample.js
```

The following JavaScript example illustrates the use of prepared statements and batches. It creates a database table and populates it with the first 100 positive integers.

```
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server     : 'demo',
             UserID     : 'DBA',
             Password   : 'sql'
           };
var conn = sqla.createConnection( cstr );
conn.connect();
conn.prepareStatement( "CREATE OR REPLACE TABLE mytable( col1 INT)" ).execute();
var stmt = conn.prepareStatement( "INSERT INTO mytable VALUES(?)" );
var BATCHSIZE = 100;
stmt.setBatchSize(BATCHSIZE);
for ( var i = 1; i <= BATCHSIZE; i++ )
{
    stmt.setInteger( 1, i );
    stmt.addBatch();
}
stmt.executeBatch();
stmt.close();
conn.commit();
conn.disconnect();
```

The following JavaScript example illustrates the use of prepared statements and exception handling. The `getcustomer` function returns a hash corresponding to the table row for the specified customer, or an error message.

```
function getcustomer( conn, customer_id )
{
    try
    {
        var query = 'SELECT * FROM Customers WHERE ID=?';

        var pstmt = conn.prepareStatement(query);
        pstmt.setInteger( 1, customer_id );
        var rs = pstmt.executeQuery();
        if ( rs.next() )
        {
            return(
            {
                id          : rs.getInteger(1),
                surname     : rs.getString(2),
                givenname   : rs.getString(3),
                street      : rs.getString(4),
                city        : rs.getString(5),
                state       : rs.getString(6),
                country     : rs.getString(7),
                postalcode  : rs.getString(8),
                phone       : rs.getString(9),
                companyname : rs.getString(10)
            } );
        }
        else
        {
            return 'No customer with ID=' + customer_id;
        }
    }
    catch (ex)
    {
        return ex.message;
    }
    finally
    {
        if( pstmt )
        {
            pstmt.close();
        }
    }
}
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server     : 'demo',
             UserID     : 'DBA',
             Password   : 'sql'
            };
var conn = sqla.createConnection( cstr );
conn.connect();
for ( var i = 200; i < 225; i++ )
{
    console.log( getcustomer( conn, i ) );
}
conn.close();
conn.disconnect();
```

In this section:

CallableStatement Class [page 9]

A callable statement object.

Connection Class [page 40]

Represents the connection to the database.

ParameterMetaData Class [page 48]

ParameterMetaData represents the metadata of a prepared statement.

PreparedStatement Class [page 56]

PreparedStatement represents a prepared SQL statement.

ResultSet Class [page 79]

A callable statement object.

ResultSetMetaData Class [page 92]

ResultSetMetaData represents the metadata of a result set.

1.1 CallableStatement Class

A callable statement object.

Syntax

```
class CallableStatement
```

Members

All members of CallableStatement, including inherited members.

Methods

Type	Method	Description
	close() [page 13]	Closes the prepared callable statement.
	execute() [page 14]	Executes the prepared callable state-ment.
Number	getBigInt(Integer) [page 14]	Gets the Number value of the specified parameter.
Buffer	getBlob(Integer) [page 15]	Gets the Buffer value of the specified parameter.
String	getClob(Integer) [page 15]	Gets the string value of the specified parameter.
Date	getDate(Integer) [page 16]	Gets the Date value of the specified pa-rameter.
Number	getDecimal(Integer) [page 17]	Gets the Number value of the specified parameter.
Number	getDouble(Integer) [page 17]	Gets the Number value of the specified parameter.
Integer	getInteger(Integer) [page 18]	Gets the Number value of the specified parameter.
ResultSetMetaData	getMetaData() [page 19]	Creates and Returns the ResultSetMe-taData object.
Boolean	getMoreResults() [page 19]	Gets the next result set.
ParameterMetaData	getParameterMetaData() [page 20]	Returns a ParameterMetaData object.
Number	getReal(Integer) [page 20]	Gets the Number value of the specified parameter.
ResultSet	getResultSet() [page 21]	Creates and returns the ResultSet ob-ject.

Type	Method	Description
Integer	getSmallInt(Integer) [page 21]	Gets the Number value of the specified parameter.
String	getString(Integer) [page 22]	Gets the string value of the specified parameter.
String	getText(Integer) [page 22]	Gets the string value of the specified parameter.
Date	getTime(Integer) [page 23]	Gets the Date value of the specified parameter.
Date	getTimestamp(Integer) [page 24]	Gets the Date value of the specified parameter.
Integer	getTinyInt(Integer) [page 24]	Gets the Number value of the specified parameter.
Integer	getUnsigned(Integer) [page 25]	Gets the Number value of the specified parameter.
Boolean	isClosed() [page 26]	Checks to see if the statement is closed.
	setBigInt(Integer, Number) [page 26]	Sets a parameter to the specified Number value.
	setBlob(Integer, Buffer) [page 27]	Sets a binary parameter to the specified JavaScript Buffer value.
	setBString(Integer, String) [page 28]	Sets a parameter to the specified String value.
	setClob(Integer, String) [page 28]	Sets a parameter to the specified String value.
	setDate [page 29]	Sets a DATE parameter to the specified JavaScript Date value.
	setDecimal(Integer, Number) [page 30]	Sets a parameter to the specified Number value.
	setDouble(Integer, Number) [page 31]	Sets a parameter to the specified Number value.
	setInteger(Integer, Integer) [page 32]	Sets a parameter to the specified Number value.
	setNull(Integer) [page 32]	Sets the specified bound parameter to NULL.
	setReal(Integer, Number) [page 33]	Sets a parameter to the specified Number value.
	setSmallInt(Integer, Integer) [page 33]	Sets a parameter to the specified Number value.
	setString(Integer, String) [page 34]	Sets a parameter to the specified String value.
	setText(Integer, String) [page 35]	Sets a parameter to the specified String value.

Type	Method	Description
	setTime [page 35]	Sets a TIME parameter to the time portion of the specified JavaScript Date value.
	setTimestamp [page 37]	Sets a TIMESTAMP parameter to the specified JavaScript Date value.
	setTinyInt(Integer, Integer) [page 39]	Sets a TINYINT parameter to the specified Number value.
	setUnsigned(Integer, Integer) [page 40]	Sets a parameter to the specified 32-bit Unsigned Integer value.

Remarks

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=dba;PWD=sql" );
var callStmt = conn.prepareCall( "CALL ShowCustomers()" );
callStmt.execute();
var result = callStmt.getResultSet();
while ( result.next() )
{
   console.log( result.getInteger(1) +
           " " + result.getString(2) );
}
callStmt.close();
conn.close();
```

In this section:

close() Method [page 13]
Closes the prepared callable statement.

execute() Method [page 14]
Executes the prepared callable statement.

getBigInt(Integer) Method [page 14]
Gets the Number value of the specified parameter.

getBlob(Integer) Method [page 15]
Gets the Buffer value of the specified parameter.

getClob(Integer) Method [page 15]
Gets the string value of the specified parameter.

getDate(Integer) Method [page 16]
Gets the Date value of the specified parameter.

getDecimal(Integer) Method [page 17]
Gets the Number value of the specified parameter.

getDouble(Integer) Method [page 17]
Gets the Number value of the specified parameter.

getInteger(Integer) Method [page 18]

Gets the Number value of the specified parameter.

getMetaData() Method [page 19]
Creates and Returns the ResultSetMetaData object.

getMoreResults() Method [page 19]
Gets the next result set.

getParameterMetaData() Method [page 20]
Returns a ParameterMetaData object.

getReal(Integer) Method [page 20]
Gets the Number value of the specified parameter.

getResultSet() Method [page 21]
Creates and returns the ResultSet object.

getSmallInt(Integer) Method [page 21]
Gets the Number value of the specified parameter.

getString(Integer) Method [page 22]
Gets the string value of the specified parameter.

getText(Integer) Method [page 22]
Gets the string value of the specified parameter.

getTime(Integer) Method [page 23]
Gets the Date value of the specified parameter.

getTimestamp(Integer) Method [page 24]
Gets the Date value of the specified parameter.

getTinyInt(Integer) Method [page 24]
Gets the Number value of the specified parameter.

getUnsigned(Integer) Method [page 25]
Gets the Number value of the specified parameter.

isClosed() Method [page 26]
Checks to see if the statement is closed.

setBigInt(Integer, Number) Method [page 26]
Sets a parameter to the specified Number value.

setBlob(Integer, Buffer) Method [page 27]
Sets a binary parameter to the specified JavaScript Buffer value.

setBString(Integer, String) Method [page 28]
Sets a parameter to the specified String value.

setClob(Integer, String) Method [page 28]
Sets a parameter to the specified String value.

setDate Method [page 29]
Sets a DATE parameter to the specified JavaScript Date value.

setDecimal(Integer, Number) Method [page 30]
Sets a parameter to the specified Number value.

setDouble(Integer, Number) Method [page 31]
Sets a parameter to the specified Number value.

setInteger(Integer, Integer) Method [page 32]
: Sets a parameter to the specified Number value.

setNull(Integer) Method [page 32]
: Sets the specified bound parameter to NULL.

setReal(Integer, Number) Method [page 33]
: Sets a parameter to the specified Number value.

setSmallInt(Integer, Integer) Method [page 33]
: Sets a parameter to the specified Number value.

setString(Integer, String) Method [page 34]
: Sets a parameter to the specified String value.

setText(Integer, String) Method [page 35]
: Sets a parameter to the specified String value.

setTime Method [page 35]
: Sets a TIME parameter to the time portion of the specified JavaScript Date value.

setTimestamp Method [page 37]
: Sets a TIMESTAMP parameter to the specified JavaScript Date value.

setTinyInt(Integer, Integer) Method [page 39]
: Sets a TINYINT parameter to the specified Number value.

setUnsigned(Integer, Integer) Method [page 40]
: Sets a parameter to the specified 32-bit Unsigned Integer value.

1.1.1 close() Method

Closes the prepared callable statement.

Syntax

```
callableStatement.close ()
```

Remarks

This method closes the prepared statement and frees up allocated resources.

This method supports asynchronous callbacks.

1.1.2 execute() Method

Executes the prepared callable statement.

Syntax

```
callableStatement.execute ()
```

Remarks

This method executes the prepared SQL statement. All parameters must be bound before execution.

This method supports asynchronous callbacks.

1.1.3 getBigInt(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getBigInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

This method returns a SQL BIGINT parameter value as a Number.

Note that JavaScript Numbers are implemented using IEEE 754 double-precision floating-point. The largest accurate integer value that can be represented is +/- 2^53 (or +/- 9007199254740992) since 64-bit floating-point employs 53 bits in the mantissa. Any BIGINT or UNSIGNED BIGINT values larger than this are approximations.

1.1.4 getBlob(Integer) Method

Gets the Buffer value of the specified parameter.

Syntax

```
callableStatement.getBlob (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a Node.js Buffer object.

Remarks

This method returns a binary parameter value as a Node.js Buffer.

1.1.5 getClob(Integer) Method

Gets the string value of the specified parameter.

Syntax

```
callableStatement.getClob (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the parameter value as a string after execution.

1.1.6 getDate(Integer) Method

Gets the Date value of the specified parameter.

Syntax

```
callableStatement.getDate (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns a parameter value as a JavaScript Date. The local time zone is always assumed.

This method can be used to retrieve TIMESTAMP and DATE SQL data types.

1.1.7 getDecimal(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getDecimal (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

This method returns a SQL numeric parameter value as a Number.

SQL data types such as NUMERIC and DECIMAL can be retrieved using this method.

1.1.8 getDouble(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getDouble (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

Any numeric SQL data types, including REAL, FLOAT, DOUBLE, and BIGINT can be retrieved using this method.

1.1.9 getInteger(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getInteger (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

This method returns a SQL integer parameter value as a Number.

Acceptable SQL data types include TINYINT, SMALLINT, INTEGER, BIGINT, and their UNSIGNED counterparts. However, the SQL value returned must fit into a 32-bit integer value.

1.1.10 getMetaData() Method

Creates and Returns the ResultSetMetaData object.

Syntax

```
callableStatement.getMetaData ()
```

Returns

Returns a ResultSetMetaData object.

1.1.11 getMoreResults() Method

Gets the next result set.

Syntax

```
callableStatement.getMoreResults ()
```

Returns

Returns true if there are more result sets and false otherwise. (type: Boolean)

Remarks

This method checks to see if there are more result sets and fetches the next available result set.

This method supports asynchronous callbacks.

1.1.12 getParameterMetaData() Method

Returns a ParameterMetaData object.

Syntax

```
callableStatement.getParameterMetaData ()
```

Returns

Returns a ParameterMetaData object.

1.1.13 getReal(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getReal (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

This method returns a SQL REAL parameter value as a Number.

1.1.14 getResultSet() Method

Creates and returns the ResultSet object.

Syntax

```
callableStatement.getResultSet ()
```

Returns

Returns a ResultSet object.

1.1.15 getSmallInt(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getSmallInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

This method returns a SQL SMALLINT or UNSIGNED SMALLINT parameter value as a Number.

1.1.16 getString(Integer) Method

Gets the string value of the specified parameter.

Syntax

```
callableStatement.getString (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the parameter value as a JavaScript string.

1.1.17 getText(Integer) Method

Gets the string value of the specified parameter.

Syntax

```
callableStatement.getText (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the parameter value as a JavaScript string.

1.1.18 getTime(Integer) Method

Gets the Date value of the specified parameter.

Syntax

```
callableStatement.getTime (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns a parameter value as a JavaScript Date. The local time zone is always assumed.

This method can be used to retrieve TIMESTAMP, DATE, and TIME SQL data types.

1.1.19 getTimestamp(Integer) Method

Gets the Date value of the specified parameter.

Syntax

```
callableStatement.getTimestamp (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns a parameter value as a JavaScript Date. The local time zone is always assumed.

This method can be used to retrieve TIMESTAMP, DATE, and TIME SQL data types.

1.1.20 getTinyInt(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getTinyInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

This method returns a SQL TINYINT parameter value as a Number.

1.1.21 getUnsigned(Integer) Method

Gets the Number value of the specified parameter.

Syntax

```
callableStatement.getUnsigned (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

The SQL value returned must fit into a 32-bit unsigned integer value. Acceptable SQL data types include TINYINT, UNSIGNED SMALLINT, UNSIGNED INTEGER, and UNSIGNED BIGINT.

An exception will be thrown if the value is bigger than 9007199254740992 (2^53) or smaller than -9007199254740992 (-2^53).

1.1.22 isClosed() Method

Checks to see if the statement is closed.

Syntax

```
callableStatement.isClosed ()
```

Returns

True on if closed, false otherwise. (type: Boolean)

1.1.23 setBigInt(Integer, Number) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setBigInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Number	value	Numeric value to be set. (type: Num-ber)

Remarks

This method is used to set large integer columns like BIGINT and UNSIGNED BIGINT.

Note that JavaScript Numbers are implemented using IEEE 754 double-precision floating-point. The largest accurate integer value that can be represented is +/- 2^53 (or +/- 9007199254740992) since 64-bit floating-point employs 53 bits in the mantissa. Any BIGINT or UNSIGNED BIGINT values larger than this are approximations.

This method sets a bind parameter to the specified value. The value being sent can be any JavaScript Number value and this value is transmitted to the server as a SQL DOUBLE. The database server converts this result to the appropriate SQL data type.

1.1.24 setBlob(Integer, Buffer) Method

Sets a binary parameter to the specified JavaScript Buffer value.

Syntax

```
callableStatement.setBlob (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Buffer	value	Binary value to be set. (type: Buffer)

Remarks

This method sets a BLOB bind parameter in the specified column. The input parameter must be a Node.js Buffer object.

1.1.25 setBString(Integer, String) Method

Sets a parameter to the specified String value.

Syntax

```
callableStatement.setBString (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1.
String	value	String value to be set.

1.1.26 setClob(Integer, String) Method

Sets a parameter to the specified String value.

Syntax

```
callableStatement.setClob (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1.
String	value	String value to be set.

1.1.27 setDate Method

Sets a DATE parameter to the specified JavaScript Date value.

Overload list

Type	Overload Name	Description
	setDate(Integer, Date) [page 29]	Sets a DATE parameter to the specified JavaScript Date value.
	setDate(Integer, String) [page 30]	Sets a DATE parameter to the specified JavaScript string value.

In this section:

setDate(Integer, Date) Method [page 29]
    Sets a DATE parameter to the specified JavaScript Date value.

setDate(Integer, String) Method [page 30]
    Sets a DATE parameter to the specified JavaScript string value.

1.1.27.1 setDate(Integer, Date) Method

Sets a DATE parameter to the specified JavaScript Date value.

Syntax

```
callableStatement.setDate (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Date	value_obj	Value used to set the SQL value. (type: Date object)

Remarks

This method sets a bind parameter to the specified value. The value being sent must be a JavaScript Date object and is be converted into a SQL DATE. In all cases, the local time zone is assumed.

This method can be used to set a SQL DATE value.

1.1.27.2 setDate(Integer, String) Method

Sets a DATE parameter to the specified JavaScript string value.

Syntax

```
callableStatement.setDate (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a bind parameter to the specified value. The string value being sent must be in a format suitable for conversion to a DATE value by the database server. In all cases, the local time zone is assumed.

This method can be used to set a SQL DATE value.

1.1.28 setDecimal(Integer, Number) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setDecimal (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Number	value	Numeric value to be set. (type: Number)

Remarks

This method can be used to set numeric SQL data types like DECIMAL or NUMERIC.

Note that JavaScript Numbers are implemented using IEEE 754 double-precision floating-point. The largest accurate integral value that can be represented is +/- 2^53 (or +/- 9007199254740992) since 64-bit floating-point employs 53 bits in the mantissa. Any BIGINT or UNSIGNED BIGINT values larger than this are approximations.

This method sets a bind parameter to the specified value. The value being sent can be any JavaScript Number value and this value is transmitted to the server as a SQL DOUBLE. The database server converts this result to the appropriate SQL data type.

1.1.29 setDouble(Integer, Number) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setDouble (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Number	value	Numeric value to be set. (type: Number)

Remarks

This method can be used to set numeric SQL data types like FLOAT and DOUBLE.

This method sets a bind parameter to the specified value. The value being sent can be any JavaScript Number value and this value is transmitted to the server as a SQL DOUBLE. The database server converts this result to the appropriate SQL data type.

1.1.30 setInteger(Integer, Integer) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setInteger (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Integer	value	Integer value to be set. (type: Integer)

Remarks

This method can be used to set integer SQL data types like INTEGER.

This method sets a bind parameter to the specified value. The value must fit into a 32-bit signed integer value.

1.1.31 setNull(Integer) Method

Sets the specified bound parameter to NULL.

Syntax

```
callableStatement.setNull (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1.

1.1.32 setReal(Integer, Number) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setReal (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Number	value	Numeric value to be set. (type: Number)

Remarks

This method can be used to set any numeric SQL data types like REAL or FLOAT.

This method sets a bind parameter to the specified value. The value being sent can be any JavaScript Number value and this value is transmitted to the server as a SQL DOUBLE. The database server converts this result to the appropriate SQL data type.

1.1.33 setSmallInt(Integer, Integer) Method

Sets a parameter to the specified Number value.

Syntax

```
callableStatement.setSmallInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Integer	value	Integer value to be set. (type: Integer)

Remarks

This method can be used to set integer SQL data types like SMALLINT or UNSIGNED SMALLINT.

This method sets a bind parameter to the specified value. The value must fit into a 16-bit integer value.

1.1.34 setString(Integer, String) Method

Sets a parameter to the specified String value.

Syntax

```
callableStatement.setString (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
String	value	String value to be set. (type: String)

Remarks

This method sets a bound parameter to the specified value. This method sets the value using the database CHAR character set.

1.1.35 setText(Integer, String) Method

Sets a parameter to the specified String value.

Syntax

```
callableStatement.setText (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
String	value	String value to be set. (type: String)

1.1.36 setTime Method

Sets a TIME parameter to the time portion of the specified JavaScript Date value.

Overload list

Type	Overload Name	Description
	setTime(Integer, Date) [page 36]	Sets a TIME parameter to the time portion of the specified JavaScript Date value.
	setTime(Integer, String) [page 36]	Sets a TIME parameter to the time portion of the specified JavaScript string value.

In this section:

setTime(Integer, Date) Method [page 36]
Sets a TIME parameter to the time portion of the specified JavaScript Date value.

setTime(Integer, String) Method [page 36]
Sets a TIME parameter to the time portion of the specified JavaScript string value.

1.1.36.1 setTime(Integer, Date) Method

Sets a TIME parameter to the time portion of the specified JavaScript Date value.

Syntax

```
callableStatement.setTime (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Date	value_obj	Value used to set the SQL value. (type: Date object)

Remarks

This method sets a bind parameter to the specified value. The value being sent must be a JavaScript Date object and will be converted into a SQL TIME. In all cases, the local time zone is assumed.

This method can be used to set a SQL TIME value.

1.1.36.2 setTime(Integer, String) Method

Sets a TIME parameter to the time portion of the specified JavaScript string value.

Syntax

```
callableStatement.setTime (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a bind parameter to the specified value. The string value being sent must be in a format suitable for conversion to a TIME value by the database server. In all cases, the local time zone is assumed.

This method can be used to set a SQL TIME value.

1.1.37 setTimestamp Method

Sets a TIMESTAMP parameter to the specified JavaScript Date value.

Overload list

Type	Overload Name	Description
	setTimestamp(Integer, Date) [page 38]	Sets a TIMESTAMP parameter to the specified JavaScript Date value.
	setTimestamp(Integer, String) [page 38]	Sets a TIMESTAMP parameter to the specified JavaScript string value.

In this section:

setTimestamp(Integer, Date) Method [page 38]
  Sets a TIMESTAMP parameter to the specified JavaScript Date value.

setTimestamp(Integer, String) Method [page 38]
  Sets a TIMESTAMP parameter to the specified JavaScript string value.

1.1.37.1 setTimestamp(Integer, Date) Method

Sets a TIMESTAMP parameter to the specified JavaScript Date value.

Syntax

```
callableStatement.setTimestamp (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Date	value_obj	Value used to set the SQL value. (type: Date object)

Remarks

This method sets a bind parameter to the specified value. The value being sent must be a JavaScript Date object and is converted into a SQL TIMESTAMP. In all cases, the local time zone is assumed.

This method can be used to set a SQL TIMESTAMP value.

1.1.37.2 setTimestamp(Integer, String) Method

Sets a TIMESTAMP parameter to the specified JavaScript string value.

Syntax

```
callableStatement.setTimestamp (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a bind parameter to the specified value. The string value being sent must be in a format suitable for conversion to a TIMESTAMP value by the database server. In all cases, the local time zone is assumed.

This method can be used to set a SQL TIMESTAMP value.

1.1.38 setTinyInt(Integer, Integer) Method

Sets a TINYINT parameter to the specified Number value.

Syntax

```
callableStatement.setTinyInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Integer	value	Integer value to be set. (type: Integer)

Remarks

This method sets a bind parameter to the specified value. The value must fit into a TINYINT value (0 - 255).

1.1.39 setUnsigned(Integer, Integer) Method

Sets a parameter to the specified 32-bit Unsigned Integer value.

Syntax

```
callableStatement.setUnsigned (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
Integer	value	Integer value to be set. (type: Integer)

Remarks

This method can be used to set integer SQL data types like UNSIGNED INTEGER and UNSIGNED SMALLINT.

This method sets a bind parameter to the specified value. The value must fit into a 32-bit unsigned integer value. The database server converts this result to the appropriate SQL data type.

1.2 Connection Class

Represents the connection to the database.

Syntax

```
class Connection
```

Members

All members of Connection, including inherited members.

Methods

Type	Method	Description
	close() [page 42]	Closes the current connection.
	commit() [page 42]	Performs a commit on the database using the connection.
	connect [page 43]	Creates a connection to the database.
	disconnect() [page 45]	Closes the current connection.
Boolean	isClosed() [page 46]	Checks if the connection is closed.
CallableStatement	prepareCall(String) [page 46]	Prepares a callable statement.
PreparedStatement	prepareStatement(String) [page 47]	Prepares a SQL statement.
	rollback() [page 47]	Performs a rollback on the database using the connection.
	setAutoCommit(Boolean) [page 48]	Changes the autocommit setting on the connection.

Remarks

The following example uses synchronous calls to create a new connection to the database server, issue a SQL query against the server, display the result set, and then disconnect from the server.

```
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server   : 'demo',
             UserID   : 'DBA',
             Password : 'sql'
};
var conn = sqla.createConnection(cstr);
conn.connect();
console.log('Connected');
stmt = conn.prepareStatement("SELECT * FROM Customers");
stmt.execute();
var result = stmt.getResultSet();
while ( result.next() )
{
  console.log( result.getString(1) +
         " " + result.getString(3) +
         " " + result.getString(2) );
}
conn.disconnect();
console.log('Disconnected');
```

In this section:

close() Method [page 42]
Closes the current connection.

commit() Method [page 42]
Performs a commit on the database using the connection.

connect Method [page 43]
Creates a connection to the database.

disconnect() Method [page 45]

Closes the current connection.

isClosed() Method [page 46]
Checks if the connection is closed.

prepareCall(String) Method [page 46]
Prepares a callable statement.

prepareStatement(String) Method [page 47]
Prepares a SQL statement.

rollback() Method [page 47]
Performs a rollback on the database using the connection.

setAutoCommit(Boolean) Method [page 48]
Changes the autocommit setting on the connection.

1.2.1 close() Method

Closes the current connection.

Syntax

```
connection.close ()
```

Remarks

This method closes the current connection and should be called before the program ends to free up resources. This method is synonymous with Connection::disconnect().

This method supports asynchronous callbacks.

1.2.2 commit() Method

Performs a commit on the database using the connection.

Syntax

```
connection.commit ()
```

Remarks

This method supports asynchronous callbacks.

1.2.3 connect Method

Creates a connection to the database.

Overload list

Type	Overload Name	Description
	connect(Hash) [page 43]	Creates a connection to the database.
	connect(Number) [page 44]	Createa a connection to the database using an existing connection.
	connect(String) [page 45]	Creates a connection to the database.

In this section:

connect(Hash) Method [page 43]
Creates a connection to the database.

connect(Number) Method [page 44]
Createa a connection to the database using an existing connection.

connect(String) Method [page 45]
Creates a connection to the database.

1.2.3.1 connect(Hash) Method

Creates a connection to the database.

Syntax

```
connection.connect (conn_hash)
```

Parameters

Type	Name	Description
Hash	conn_hash	A hash of valid connection parameters. (type: Object)

Remarks

This method creates a new connection using the specified hash of connection parameters. Before the end of the program, the connection should be disconnected using the disconnect() or close() method to free up resources. The string ";CHARSET='UTF-8'" is automatically appended to the end of the resulting connection string as all strings are sent to the database server using that encoding.

Any connection parameters specified in the createConnection call are overridden.

This method supports asynchronous callbacks.

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( { uid: 'dba',
               pwd: 'sql',
               server: 'myserver' } );
```

1.2.3.2 connect(Number) Method

Createa a connection to the database using an existing connection.

Syntax

```
connection.connect (dbcapi_handle)
```

Parameters

Type	Name	Description
Number	dbcapi_handle	The connection handle. (type: Number)

Remarks

This method connects to the database using an existing connection. The connection handle obtained from the JavaScript External Environment is passed as a parameter. The close() or disconnect() method should be called before the end of the program to free up resources.

This method supports asynchronous callbacks.

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( sa_dbcapi_handle );
```

1.2.3.3 connect(String) Method

Creates a connection to the database.

Syntax

```
connection.connect (conn_string)
```

Parameters

Type	Name	Description
String	conn_string	A valid connection string. (type: String)

Remarks

This method creates a new connection using the specified connection string. Before the end of the program, the connection should be disconnected using the disconnect() or close() method to free up resources. The string ";CHARSET='UTF-8'" is automatically appended to the end of the connection string as all strings are sent to the database server using that encoding.

Any connection parameters specified in the createConnection call are overridden.

This method supports asynchronous callbacks.

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=dba;PWD=sql;SERVER=myserver" );
```

1.2.4 disconnect() Method

Closes the current connection.

Syntax

```
connection.disconnect ()
```

Remarks

This method closes the current connection and should be called before the program ends to free up resources. This method is synonymous with Connection::close().

This method supports asynchronous callbacks.

1.2.5 isClosed() Method

Checks if the connection is closed.

Syntax

```
connection.isClosed ()
```

Returns

Returns true if closed, and false otherwise. (type: Boolean)

1.2.6 prepareCall(String) Method

Prepares a callable statement.

Syntax

```
connection.prepareCall (sql_stmt)
```

Parameters

Type	Name	Description
String	sql_stmt	A SQL statement to prepare. (type: String)

Returns

Returns CallableStatement object.

Remarks

This method prepares a SQL statement and returns a callable statement object.

This method supports asynchronous callbacks.

1.2.7 prepareStatement(String) Method

Prepares a SQL statement.

Syntax

```
connection.prepareStatement (sql_stmt)
```

Parameters

Type	Name	Description
String	sql_stmt	A SQL statement to prepare. (type: String)

Returns

Returns a PreparedStatement object.

Remarks

This method prepares a SQL statement and returns a prepared statement object.

This method supports asynchronous callbacks.

1.2.8 rollback() Method

Performs a rollback on the database using the connection.

Syntax

```
connection.rollback ()
```

Remarks

This method supports asynchronous callbacks.

1.2.9 setAutoCommit(Boolean) Method

Changes the autocommit setting on the connection.

Syntax

```
connection.setAutoCommit (flag)
```

Parameters

Type	Name	Description
Boolean	flag	Value to enable (1) or disable (0) auto-commit. (type: Boolean)

Remarks

If autocommit is enabled, a commit is executed following every execute() call.

1.3 ParameterMetaData Class

ParameterMetaData represents the metadata of a prepared statement.

Syntax

```
class ParameterMetaData
```

Members

All members of ParameterMetaData, including inherited members.

Methods

Type	Method	Description
Integer	getParameterCount() [page 50]	Returns the number of parameters in the prepared statement.
Integer	getParameterMode(Integer) [page 51]	Returns the mode of the specified parameter.
String	getParameterName(Integer) [page 51]	Returns the name of the specified parameter.
Integer	getParameterType(Integer) [page 52]	Returns returns a number representing the type of the specified parameter.
String	getParameterTypeName(Integer) [page 53]	Returns a string containing the name of the type for the specified parameter.
Boolean	getPrecision(Integer) [page 53]	Returns the precision of the specified parameter.
Boolean	getScale(Integer) [page 54]	Returns the scale of the specified parameter.
Boolean	hasDefault(Integer) [page 54]	Checks if the specified parameter has a default value.
Boolean	isNullable(Integer) [page 55]	Checks if the specified parameter is nullable.
Boolean	isSigned(Integer) [page 56]	Checks if the specified parameter is signed.

Remarks

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=DBA;PWD=sql" );
var stmt = conn.prepareStatement( "INSERT INTO Departments
VALUES( :id,:name,:head )" );
stmt.setInteger( 1, 201 );
stmt.setString( 2, "Returns" );
stmt.setInteger( 3, 1250 );
try
{
    stmt.execute();
}
catch (ex)
{
}
var metaData = stmt.getParameterMetaData()
var parms = metaData.getParameterCount();
for ( var i = 1; i <= parms; i++ )
{
    console.log( metaData.getParameterName(i) );
}
stmt.close();
conn.commit();
conn.close();
```

In this section:

getParameterCount() Method [page 50]
Returns the number of parameters in the prepared statement.

getParameterMode(Integer) Method [page 51]
Returns the mode of the specified parameter.

getParameterName(Integer) Method [page 51]
Returns the name of the specified parameter.

getParameterType(Integer) Method [page 52]
Returns returns a number representing the type of the specified parameter.

getParameterTypeName(Integer) Method [page 53]
Returns a string containing the name of the type for the specified parameter.

getPrecision(Integer) Method [page 53]
Returns the precision of the specified parameter.

getScale(Integer) Method [page 54]
Returns the scale of the specified parameter.

hasDefault(Integer) Method [page 54]
Checks if the specified parameter has a default value.

isNullable(Integer) Method [page 55]
Checks if the specified parameter is nullable.

isSigned(Integer) Method [page 56]
Checks if the specified parameter is signed.

1.3.1 getParameterCount() Method

Returns the number of parameters in the prepared statement.

Syntax

```
parameterMetaData.getParameterCount ()
```

Returns

Returns the number of bind parameters. (type: Integer)

1.3.2 getParameterMode(Integer) Method

Returns the mode of the specified parameter.

Syntax

```
parameterMetaData.getParameterMode (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns the mode of the bind parameter. (type: Integer)

Remarks

0 is INVALID, 1 is IN, 2 is OUT, and 3 is INOUT.

1.3.3 getParameterName(Integer) Method

Returns the name of the specified parameter.

Syntax

```
parameterMetaData.getParameterName (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns the name of bind parameter. (type: String)

1.3.4 getParameterType(Integer) Method

Returns returns a number representing the type of the specified parameter.

Syntax

```
parameterMetaData.getParameterType (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns the type of bind parameter. (type: Integer)

Remarks

Refer to the a_sqlany_data_type enumeration to find the corresponding type.

1.3.5 getParameterTypeName(Integer) Method

Returns a string containing the name of the type for the specified parameter.

Syntax

```
parameterMetaData.getParameterTypeName (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns the type of bind parameter. (type: String)

1.3.6 getPrecision(Integer) Method

Returns the precision of the specified parameter.

Syntax

```
parameterMetaData.getPrecision (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns true if it is signed, false otherwise. (type: Boolean)

Remarks

Note: This method is not supported.

1.3.7 getScale(Integer) Method

Returns the scale of the specified parameter.

Syntax

```
parameterMetaData.getScale (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns true if it is signed, false otherwise. (type: Boolean)

Remarks

Note: This method is not supported.

1.3.8 hasDefault(Integer) Method

Checks if the specified parameter has a default value.

Syntax

```
parameterMetaData.hasDefault (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns true if it is signed, false otherwise. (type: Boolean)

Remarks

Note: This method is not supported.

1.3.9 isNullable(Integer) Method

Checks if the specified parameter is nullable.

Syntax

```
parameterMetaData.isNullable (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns true if it is signed, false otherwise. (type: Boolean)

Remarks

Note: This method is not supported.

1.3.10 isSigned(Integer) Method

Checks if the specified parameter is signed.

Syntax

```
parameterMetaData.isSigned (index)
```

Parameters

Type	Name	Description
Integer	index	The index of the parameter starting from 1. (type: Integer)

Returns

Returns true if it is signed, false otherwise. (type: Boolean)

1.4 PreparedStatement Class

PreparedStatement represents a prepared SQL statement.

Syntax

```
class PreparedStatement
```

Members

All members of PreparedStatement, including inherited members.

Methods

Type	Method	Description
PreparedStatement	addBatch() [page 60]	Adds a set of parameters.
PreparedStatement	close() [page 60]	Closes the prepared statement.
Boolean PreparedStatement	execute() [page 61]	Executes a prepared SQL statement.
PreparedStatement	executeBatch() [page 61]	Executes a batch.
ResultSet PreparedStatement	executeQuery() [page 62]	Executes a prepared SQL statement.
Integer PreparedStatement	executeUpdate() [page 62]	Executes a prepared SQL UPDATE statement.
ResultSetMetaData PreparedStatement	getMetaData() [page 63]	Creates and returns a ResultSetMetaData object.
Boolean PreparedStatement	getMoreResults() [page 63]	Gets the next result set.
ParameterMetaData PreparedStatement	getParameterMetaData() [page 63]	Creates and returns a ParameterMetaData object.
ResultSet PreparedStatement	getResultSet() [page 64]	Creates and returns a ResultSet object.
Boolean PreparedStatement	isClosed() [page 64]	Checks to see if the prepared statement is closed.
PreparedStatement	setBatchSize(Integer) [page 64]	Reserves space for the indicated number of parameter sets.
PreparedStatement	setBigInt(Integer, Number) [page 65]	Sets a numeric value for the specified parameter.
PreparedStatement	setBlob(Integer, Buffer) [page 66]	Sets a binary value for the specified parameter.
PreparedStatement	setBString(Integer, String) [page 66]	Set a string parameter used for BINARY, VARBINARY column types, should be used for strings containing UNICODE characters.
PreparedStatement	setClob(Integer, String) [page 67]	setClob is used to specify the values for CLOB (LONG VARCHAR) column types.
PreparedStatement	setDate [page 67]	Sets a date value for the specified parameter.
PreparedStatement	setDecimal(Integer, Number) [page 69]	Sets a numeric value for the specified parameter.
PreparedStatement	setDouble(Integer, Number) [page 70]	Sets a numeric value for the specified parameter.
PreparedStatement	setInteger(Integer, Integer) [page 70]	Sets an integer parameter used for integer column types like TINYINT, SMALLINT, and INT.
PreparedStatement	setNull(Integer) [page 71]	Sets a NULL value for the specified parameter.
PreparedStatement	setReal(Integer, Number) [page 72]	Sets a floating-point value for the specified parameter.

Type	Method	Description
PreparedStatement	setSmallInt(Integer, Integer) [page 72]	Sets an integer parameter used for SMALLINT column types.
PreparedStatement	setString(Integer, String) [page 73]	Sets a string parameter used for CHAR, VARCHAR, LONG VARCHAR and other character column types.
PreparedStatement	setText(Integer, String) [page 73]	setText is used to specify the values for TEXT (LONG VARCHAR) column types.
PreparedStatement	setTime [page 74]	Sets a time value for the specified parameter.
PreparedStatement	setTimestamp [page 76]	Sets a timestamp value for the specified parameter.
PreparedStatement	setTinyInt(Integer, Integer) [page 77]	Sets a TINYINT value for the specified parameter.
PreparedStatement	setUnsigned(Integer, Integer) [page 78]	Sets an unsigned integer value for the specified parameter.

Remarks

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=DBA;PWD=sql" );
var stmt = conn.prepareStatement( "DROP TABLE mytable" );
stmt.execute();
stmt.close();
conn.close();
```

In this section:

addBatch() Method [page 60]
Adds a set of parameters.

close() Method [page 60]
Closes the prepared statement.

execute() Method [page 61]
Executes a prepared SQL statement.

executeBatch() Method [page 61]
Executes a batch.

executeQuery() Method [page 62]
Executes a prepared SQL statement.

executeUpdate() Method [page 62]
Executes a prepared SQL UPDATE statement.

getMetaData() Method [page 63]
Creates and returns a ResultSetMetaData object.

getMoreResults() Method [page 63]

Gets the next result set.

getParameterMetaData() Method [page 63]
: Creates and returns a ParameterMetaData object.

getResultSet() Method [page 64]
: Creates and returns a ResultSet object.

isClosed() Method [page 64]
: Checks to see if the prepared statement is closed.

setBatchSize(Integer) Method [page 64]
: Reserves space for the indicated number of parameter sets.

setBigInt(Integer, Number) Method [page 65]
: Sets a numeric value for the specified parameter.

setBlob(Integer, Buffer) Method [page 66]
: Sets a binary value for the specified parameter.

setBString(Integer, String) Method [page 66]
: Set a string parameter used for BINARY, VARBINARY column types, should be used for strings containing UNICODE characters.

setClob(Integer, String) Method [page 67]
: setClob is used to specify the values for CLOB (LONG VARCHAR) column types.

setDate Method [page 67]
: Sets a date value for the specified parameter.

setDecimal(Integer, Number) Method [page 69]
: Sets a numeric value for the specified parameter.

setDouble(Integer, Number) Method [page 70]
: Sets a numeric value for the specified parameter.

setInteger(Integer, Integer) Method [page 70]
: Sets an integer parameter used for integer column types like TINYINT, SMALLINT, and INT.

setNull(Integer) Method [page 71]
: Sets a NULL value for the specified parameter.

setReal(Integer, Number) Method [page 72]
: Sets a floating-point value for the specified parameter.

setSmallInt(Integer, Integer) Method [page 72]
: Sets an integer parameter used for SMALLINT column types.

setString(Integer, String) Method [page 73]
: Sets a string parameter used for CHAR, VARCHAR, LONG VARCHAR and other character column types.

setText(Integer, String) Method [page 73]
: setText is used to specify the values for TEXT (LONG VARCHAR) column types.

setTime Method [page 74]
: Sets a time value for the specified parameter.

setTimestamp Method [page 76]
: Sets a timestamp value for the specified parameter.

setTinyInt(Integer, Integer) Method [page 77]

Sets a TINYINT value for the specified parameter.

setUnsigned(Integer, Integer) Method [page 78]
Sets an unsigned integer value for the specified parameter.

1.4.1 addBatch() Method

Adds a set of parameters.

Syntax

```
preparedStatement.addBatch ()
```

Remarks

This method adds the last set of parameter values and iterates to the next batch slot. Enough slots must have been reserved using the setBatchSize() method.

```
var stmt = conn.prepareStatement( "INSERT INTO mytable VALUES(?)" );
var BATCHSIZE = 100;
stmt.setBatchSize(BATCHSIZE);
for ( var i = 1; i <= BATCHSIZE; i++ )
{
   stmt.setInteger( 1, i );
   stmt.addBatch();
}
stmt.executeBatch();
```

1.4.2 close() Method

Closes the prepared statement.

Syntax

```
preparedStatement.close ()
```

Remarks

This method closes the prepared statement and frees up allocated resources.

1.4.3 execute() Method

Executes a prepared SQL statement.

Syntax

```
preparedStatement.execute ()
```

Returns

Returns true if there is a result set and false otherwise.

Remarks

This method executes the prepared SQL statement, returning true if a result is available and false otherwise.

This method supports asynchronous callbacks.

1.4.4 executeBatch() Method

Executes a batch.

Syntax

```
preparedStatement.executeBatch ()
```

Remarks

This method executes a batch. Use setBatchSize() and addBatch() to prepare the batch.

This method supports asynchronous callbacks.

```
var stmt = conn.prepareStatement( "INSERT INTO mytable VALUES(?)" );
var BATCHSIZE = 100;
stmt.setBatchSize(BATCHSIZE);
for ( var i = 1; i <= BATCHSIZE; i++ )
{
   stmt.setInteger( 1, i );
   stmt.addBatch();
}
stmt.executeBatch();
```

1.4.5 executeQuery() Method

Executes a prepared SQL statement.

Syntax

```
preparedStatement.executeQuery ()
```

Returns

Returns a ResultSet object.

Remarks

This method executes the prepared SQL statement, and returns a ResultSet object.

This method supports asynchronous callbacks.

1.4.6 executeUpdate() Method

Executes a prepared SQL UPDATE statement.

Syntax

```
preparedStatement.executeUpdate ()
```

Returns

Returns the number of rows affected by the statement.

Remarks

This method executes a prepared SQL UPDATE statement, and returns the number of rows affected.

This method supports asynchronous callbacks.

1.4.7 getMetaData() Method

Creates and returns a ResultSetMetaData object.

Syntax

```
preparedStatement.getMetaData ()
```

Returns

Returns the ResultSetMetaData object.

1.4.8 getMoreResults() Method

Gets the next result set.

Syntax

```
preparedStatement.getMoreResults ()
```

Returns

Returns true if there are more result sets and false otherwise. (type: Boolean)

Remarks

This method checks to see if there are more result sets and iterates to the next one if available.

1.4.9 getParameterMetaData() Method

Creates and returns a ParameterMetaData object.

Syntax

```
preparedStatement.getParameterMetaData ()
```

Returns

Returns the ParameterMetaData object.

1.4.10 getResultSet() Method

Creates and returns a ResultSet object.

Syntax

```
preparedStatement.getResultSet ()
```

Returns

Returns the ResultSet object.

1.4.11 isClosed() Method

Checks to see if the prepared statement is closed.

Syntax

```
preparedStatement.isClosed ()
```

ReturnsTrue on if closed, false otherwise.

1.4.12 setBatchSize(Integer) Method

Reserves space for the indicated number of parameter sets.

Syntax

```
preparedStatement.setBatchSize (size)
```

Parameters

Type	Name	Description
Integer	size	Number of slots to allocate (type: Integer)

Remarks

```
var stmt = conn.prepareStatement( "INSERT INTO mytable VALUES(?)" );
var BATCHSIZE = 100;
stmt.setBatchSize(BATCHSIZE);
for ( var i = 1; i <= BATCHSIZE; i++ )
{
   stmt.setInteger( 1, i );
   stmt.addBatch();
}
stmt.executeBatch();
```

1.4.13 setBigInt(Integer, Number) Method

Sets a numeric value for the specified parameter.

Syntax

```
preparedStatement.setBigInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Number	value	Numeric value to be set (type: Number)

Remarks

This method is used to set large integer columns like BIGINT and UNSIGNED BIGINT.

Note that JavaScript Numbers are implemented using IEEE 754 double-precision floating-point. The largest accurate integer value that can be represented is +/- 2^53 (or +/- 9007199254740992) since 64-bit floating-point employs 53 bits in the mantissa. Any BIGINT or UNSIGNED BIGINT values larger than this are approximations.

1.4.14 setBlob(Integer, Buffer) Method

Sets a binary value for the specified parameter.

Syntax

```
preparedStatement.setBlob (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Buffer	value	Binary value to be set (type: Buffer)

Remarks

This method sets a BLOB (LONG BINARY) parameter in the specified column. The input parameter must be a Node.js Buffer object.

1.4.15 setBString(Integer, String) Method

Set a string parameter used for BINARY, VARBINARY column types, should be used for strings containing UNICODE characters.

Syntax

```
preparedStatement.setBString (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
String	value	String value to be set (type: String)

1.4.16 setClob(Integer, String) Method

setClob is used to specify the values for CLOB (LONG VARCHAR) column types.

Syntax

```
preparedStatement.setClob (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
String	value	String value to be set (type: String)

1.4.17 setDate Method

Sets a date value for the specified parameter.

Overload list

Type	Overload Name	Description
PreparedStatement	setDate(Integer, Date) [page 68]	Sets a date value for the specified parameter.
PreparedStatement	setDate(Integer, String) [page 68]	Sets a date value for the specified parameter.

In this section:

setDate(Integer, Date) Method [page 68]
Sets a date value for the specified parameter.

setDate(Integer, String) Method [page 68]
Sets a date value for the specified parameter.

1.4.17.1 setDate(Integer, Date) Method

Sets a date value for the specified parameter.

Syntax

```
preparedStatement.setDate (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1 (type: Integer)
Date	value_obj	Value used to set the SQL value (type: Date object)

Remarks

This method sets a parameter to the specified value. The value must be a JavaScript Date object and is converted into a SQL DATE. In all cases, the local time zone is assumed.

This method is used to set a SQL DATE value.

1.4.17.2 setDate(Integer, String) Method

Sets a date value for the specified parameter.

Syntax

```
preparedStatement.setDate (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a parameter to the specified value. The value must be a timestamp string recognizable by the database server and is converted to the appropriate DATE value when sent to the server.

This method is used to set a SQL DATE value.

1.4.18 setDecimal(Integer, Number) Method

Sets a numeric value for the specified parameter.

Syntax

```
preparedStatement.setDecimal (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Number	value	Numeric value to be set (type: Number)

Remarks

This method is used to set numeric columns like NUMERIC and DECIMAL.

1.4.19 setDouble(Integer, Number) Method

Sets a numeric value for the specified parameter.

Syntax

```
preparedStatement.setDouble (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Number	value	Numeric value to be set (type: Number)

Remarks

This method sets a parameter to the specified value. The value is any JavaScript numeric value and is converted into a SQL DOUBLE. The database server then casts the result to the appropriate SQL value.

This method is used to set any numeric columns like REAL, FLOAT, DOUBLE, DECIMAL, or BIGINT.

1.4.20 setInteger(Integer, Integer) Method

Sets an integer parameter used for integer column types like TINYINT, SMALLINT, and INT.

Syntax

```
preparedStatement.setInteger (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Integer	value	Integer value to be set (type: Integer)

Remarks

This method sets a parameter to the specified value. The value must fit into a 32-bit integer value.

This method is used to set an integer value.

1.4.21 setNull(Integer) Method

Sets a NULL value for the specified parameter.

Syntax

```
preparedStatement.setNull (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)

Remarks

This method sets a specified parameter to NULL.

1.4.22 setReal(Integer, Number) Method

Sets a floating-point value for the specified parameter.

Syntax

```
preparedStatement.setReal (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Number	value	Numeric value to be set (type: Number)

Remarks

This method is used to set floating-point columns like REAL, DOUBLE, and FLOAT.

1.4.23 setSmallInt(Integer, Integer) Method

Sets an integer parameter used for SMALLINT column types.

Syntax

```
preparedStatement.setSmallInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1. (type: Integer)
Integer	value	Integer value to be set (type: Integer)

Remarks

This method sets a parameter to the specified value. The value must fit into a 16-bit integer value.

This method is used to set a small integer value.

1.4.24 setString(Integer, String) Method

Sets a string parameter used for CHAR, VARCHAR, LONG VARCHAR and other character column types.

Syntax

```
preparedStatement.setString (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
String	value	String value to be set (type: String)

Remarks

This method sets a string value for the specified character column. The setString() method sets the value in the database CHAR character set.

1.4.25 setText(Integer, String) Method

setText is used to specify the values for TEXT (LONG VARCHAR) column types.

Syntax

```
preparedStatement.setText (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1. (type: Integer)
String	value	String value to be set (type: String)

1.4.26 setTime Method

Sets a time value for the specified parameter.

Overload list

Type	Overload Name	Description
PreparedStatement	setTime(Integer, Date) [page 74]	Sets a time value for the specified pa-rameter.
PreparedStatement	setTime(Integer, String) [page 75]	Sets a time value for the specified pa-rameter.

In this section:

setTime(Integer, Date) Method [page 74]
Sets a time value for the specified parameter.

setTime(Integer, String) Method [page 75]
Sets a time value for the specified parameter.

1.4.26.1 setTime(Integer, Date) Method

Sets a time value for the specified parameter.

Syntax

```
preparedStatement.setTime (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1 (type: Integer)
Date	value_obj	Value used to set the SQL value (type: Date object)

Remarks

This method sets a parameter to the specified value. The value must be a JavaScript Date object and is converted into a SQL TIME. In all cases, the local time zone is assumed.

This method is used to set a SQL TIME value.

1.4.26.2 setTime(Integer, String) Method

Sets a time value for the specified parameter.

Syntax

```
preparedStatement.setTime (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the pre-pared statement starting from 1 (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a parameter to the specified value. The value must be a timestamp string recognizable by the database server and is converted to the appropriate TIME value when sent to the server.

This method is used to set a SQL TIME value.

1.4.27 setTimestamp Method

Sets a timestamp value for the specified parameter.

Overload list

Type	Overload Name	Description
PreparedStatement	setTimestamp(Integer, Date) [page 76]	Sets a timestamp value for the specified parameter.
PreparedStatement	setTimestamp(Integer, String) [page 77]	Sets a timestamp value for the specified parameter.

In this section:

setTimestamp(Integer, Date) Method [page 76]
Sets a timestamp value for the specified parameter.

setTimestamp(Integer, String) Method [page 77]
Sets a timestamp value for the specified parameter.

1.4.27.1 setTimestamp(Integer, Date) Method

Sets a timestamp value for the specified parameter.

Syntax

```
preparedStatement.setTimestamp (colIndex, value_obj)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Date	value_obj	Value used to set the SQL value (type: Date object)

Remarks

This method sets a parameter to the specified value. The value must be a JavaScript Date object and is converted into a SQL TIMESTAMP datatype. In all cases, the local time zone is assumed.

This method is used to set a SQL TIMESTAMP value.

1.4.27.2 setTimestamp(Integer, String) Method

Sets a timestamp value for the specified parameter.

Syntax

```
preparedStatement.setTimestamp (colIndex, value_str)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
String	value_str	Value used to set the SQL value. (type: String)

Remarks

This method sets a parameter to the specified value. The value must be a timestamp string recognizable by the database server and is converted to the appropriate TIMESTAMP value when sent to the server.

This method is used to set a SQL TIMESTAMP value.

1.4.28 setTinyInt(Integer, Integer) Method

Sets a TINYINT value for the specified parameter.

Syntax

```
preparedStatement.setTinyInt (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Integer	value	Integer value to be set (type: Integer)

Remarks

This method sets a parameter to the specified value. The value must fit into a TINYINT column (0 - 255).

This method is used to set a SQL TINYINT column.

1.4.29 setUnsigned(Integer, Integer) Method

Sets an unsigned integer value for the specified parameter.

Syntax

```
preparedStatement.setUnsigned (colIndex, value)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the parameter in the prepared statement starting from 1 (type: Integer)
Integer	value	Integer value to be set (type: Integer)

Remarks

This method sets a parameter to the specified value. The value must fit into a 32-bit unsigned integer value. This method is used to set integer columns like UNSIGNED INTEGER, INTEGER, SMALLINT, or UNSIGNED SMALLINT.

1.5 ResultSet Class

A callable statement object.

Syntax

```
class ResultSet
```

Members

All members of ResultSet, including inherited members.

Methods

Type	Method	Description
	close() [page 81]	This method closes the ResultSet object and frees up resources.
Number	getBigInt(Integer) [page 81]	Gets the Numeric value of the specified column of the result set.
Buffer	getBlob(Integer) [page 82]	Gets the Binary value of the specified column of the result set.
String	getClob(Integer) [page 83]	Gets the String value of the specified column of the result set.
Date	getDate(Integer) [page 83]	Gets the Date value of the specified column of the result set.
Number	getDecimal(Integer) [page 84]	Gets the Numeric value of the specified column of the result set.
Number	getDouble(Integer) [page 85]	Gets the Numeric value of the specified column of the result set.
Integer	getInteger(Integer) [page 85]	Gets the Integer value of the specified column of the result set.
Number	getReal(Integer) [page 86]	Gets the Numeric value of the specified column of the result set.
Integer	getSmallInt(Integer) [page 87]	Gets the Integer value of the specified column of the result set.
String	getString(Integer) [page 87]	Gets the String value of the specified column of the result set.
String	getText(Integer) [page 88]	Gets the String value of the specified column of the result set.
Date	getTime(Integer) [page 89]	Gets the Time value of the specified column of the result set.

Type	Method	Description
Date	getTimestamp(Integer) [page 89]	Gets the Timestamp value of the specified column of the result set.
Integer	getTinyInt(Integer) [page 90]	Gets the Integer value of the specified column of the result set.
Integer	getUnsigned(Integer) [page 91]	Gets the Integer value of the specified column of the result set.
Boolean	isClosed() [page 91]	This method checks if the result set is closed.
Boolean	next() [page 92]	Fetches the next row of the result set.

Remarks

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=dba;PWD=sql" );
var stmt = conn.prepareStatement( "SELECT * FROM Customers" );
stmt.execute();
var result = stmt.getResultSet();
while ( result.next() )
{
   console.log( result.getInteger(1) +
           " " + result.getString(2) );
}
stmt.close();
conn.close();
```

In this section:

close() Method [page 81]
This method closes the ResultSet object and frees up resources.

getBigInt(Integer) Method [page 81]
Gets the Numeric value of the specified column of the result set.

getBlob(Integer) Method [page 82]
Gets the Binary value of the specified column of the result set.

getClob(Integer) Method [page 83]
Gets the String value of the specified column of the result set.

getDate(Integer) Method [page 83]
Gets the Date value of the specified column of the result set.

getDecimal(Integer) Method [page 84]
Gets the Numeric value of the specified column of the result set.

getDouble(Integer) Method [page 85]
Gets the Numeric value of the specified column of the result set.

getInteger(Integer) Method [page 85]
Gets the Integer value of the specified column of the result set.

getReal(Integer) Method [page 86]
: Gets the Numeric value of the specified column of the result set.

getSmallInt(Integer) Method [page 87]
: Gets the Integer value of the specified column of the result set.

getString(Integer) Method [page 87]
: Gets the String value of the specified column of the result set.

getText(Integer) Method [page 88]
: Gets the String value of the specified column of the result set.

getTime(Integer) Method [page 89]
: Gets the Time value of the specified column of the result set.

getTimestamp(Integer) Method [page 89]
: Gets the Timestamp value of the specified column of the result set.

getTinyInt(Integer) Method [page 90]
: Gets the Integer value of the specified column of the result set.

getUnsigned(Integer) Method [page 91]
: Gets the Integer value of the specified column of the result set.

isClosed() Method [page 91]
: This method checks if the result set is closed.

next() Method [page 92]
: Fetches the next row of the result set.

1.5.1 close() Method

This method closes the ResultSet object and frees up resources.

Syntax

```
resultSet.close ()
```

1.5.2 getBigInt(Integer) Method

Gets the Numeric value of the specified column of the result set.

Syntax

```
resultSet.getBigInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

Any numeric SQL types, including BIGINT, DOUBLE, REAL, FLOAT and DOUBLE can be retrieved using this method.

1.5.3 getBlob(Integer) Method

Gets the Binary value of the specified column of the result set.

Syntax

```
resultSet.getBlob (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a Node.js Buffer object.

Remarks

This method returns the value in the specified column as a Node.js Buffer.

1.5.4 getClob(Integer) Method

Gets the String value of the specified column of the result set.

Syntax

```
resultSet.getClob (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the result set value as a string.

1.5.5 getDate(Integer) Method

Gets the Date value of the specified column of the result set.

Syntax

```
resultSet.getDate (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns the value in the specified column as a JavaScript Date object. The local time zone is always assumed.

This method is used to retrieve TIMESTAMP, TIME, and DATE SQL types.

1.5.6 getDecimal(Integer) Method

Gets the Numeric value of the specified column of the result set.

Syntax

```
resultSet.getDecimal (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

Any numeric SQL types, including BIGINT, DOUBLE, REAL, FLOAT and DOUBLE can be retrieved using this method.

1.5.7 getDouble(Integer) Method

Gets the Numeric value of the specified column of the result set.

Syntax

```
resultSet.getDouble (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

Any numeric SQL types, including BIGINT, DOUBLE, REAL, FLOAT and DOUBLE can be retrieved using this method.

1.5.8 getInteger(Integer) Method

Gets the Integer value of the specified column of the result set.

Syntax

```
resultSet.getInteger (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

This method returns the result set value as an Integer. The SQL value returned must fit into a 32-bit integer value. Acceptable SQL types include SMALLINT, INTEGER, and TINYINT.

1.5.9 getReal(Integer) Method

Gets the Numeric value of the specified column of the result set.

Syntax

```
resultSet.getReal  (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a numeric value.

Remarks

Any numeric SQL types, including BIGINT, DOUBLE, REAL, FLOAT and DOUBLE can be retrieved using this method.

1.5.10 getSmallInt(Integer) Method

Gets the Integer value of the specified column of the result set.

Syntax

```
resultSet.getSmallInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

1.5.11 getString(Integer) Method

Gets the String value of the specified column of the result set.

Syntax

```
resultSet.getString (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the result set value as a string.

1.5.12 getText(Integer) Method

Gets the String value of the specified column of the result set.

Syntax

```
resultSet.getText (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a String.

Remarks

This method returns the result set value as a string.

1.5.13 getTime(Integer) Method

Gets the Time value of the specified column of the result set.

Syntax

```
resultSet.getTime (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns the value in the specified column as a JavaScript Date object. The local time zone is always assumed.

This method is used to retrieve TIMESTAMP, TIME, and DATE SQL types.

1.5.14 getTimestamp(Integer) Method

Gets the Timestamp value of the specified column of the result set.

Syntax

```
resultSet.getTimestamp (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as a Date object.

Remarks

This method returns the value in the specified column as a JavaScript Date object. The local time zone is always assumed.

This method is used to retrieve TIMESTAMP, TIME, and DATE SQL types.

1.5.15 getTinyInt(Integer) Method

Gets the Integer value of the specified column of the result set.

Syntax

```
resultSet.getTinyInt (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

1.5.16 getUnsigned(Integer) Method

Gets the Integer value of the specified column of the result set.

Syntax

```
resultSet.getUnsigned (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column of the result set starting from 1. (type: Integer)

Returns

Returns the value as an Integer.

Remarks

The SQL value must fit into a 32-bit unsigned integer value. Acceptable SQL types include UNSIGNED SMALLINT, UNSIGNED INTEGER, and TINYINT.

1.5.17 isClosed() Method

This method checks if the result set is closed.

Syntax

```
resultSet.isClosed ()
```

Returns

Returns true if closed, false otherwise. (type: Boolean)

1.5.18 next() Method

Fetches the next row of the result set.

Syntax

```
resultSet.next ()
```

Returns

Returns true on success. (type: Boolean)

Remarks

This method attempts to fetch the next row of the result set, returning true when successful and false otherwise.

This method supports asynchronous callbacks.

1.6 ResultSetMetaData Class

ResultSetMetaData represents the metadata of a result set.

Syntax

```
class ResultSetMetaData
```

Members

All members of ResultSetMetaData, including inherited members.

Methods

Type	Method	Description
String	getCatalogName() [page 94]	Returns the database server name.
Integer	getColumnCount() [page 94]	Returns the number of columns in the result set.

Type	Method	Description
String	getColumnLabel(Integer) [page 95]	Returns the alias or name of the speci-fied column.
String	getColumnName(Integer) [page 95]	Returns the name of the specified col-umn.
Integer	getColumnType(Integer) [page 96]	Returns a number representing the type of the specified column.
String	getColumnTypeName(Integer) [page 96]	Returns a string containing the name of the type for the specified column.
Integer	getDisplaySize(Integer) [page 97]	Returns the maximum display size of the specified column.
Integer	getPrecision(Integer) [page 97]	Returns the precision of the specified column.
Integer	getScale(Integer) [page 98]	Returns the scale of the specified col-umn.
Integer	getTableName(Integer) [page 98]	Returns the table name for the speci-fied column.

Remarks

```
var sqla = require( 'sqlanywhere-xs' );
var conn = sqla.createConnection();
conn.connect( "UID=DBA;PWD=sql" );
var stmt = conn.prepareStatement( "SELECT * FROM Customers" );
stmt.execute();
var resultMeta = stmt.getMetaData()
var columns = resultMeta.getColumnCount();
for ( var i = 1; i <= columns; i++ )
{
   console.log( resultMeta.getColumnName(i) );
}
stmt.close();
conn.close();
```

In this section:

getCatalogName() Method [page 94]
Returns the database server name.

getColumnCount() Method [page 94]
Returns the number of columns in the result set.

getColumnLabel(Integer) Method [page 95]
Returns the alias or name of the specified column.

getColumnName(Integer) Method [page 95]
Returns the name of the specified column.

getColumnType(Integer) Method [page 96]
Returns a number representing the type of the specified column.

getColumnTypeName(Integer) Method [page 96]
: Returns a string containing the name of the type for the specified column.

getDisplaySize(Integer) Method [page 97]
: Returns the maximum display size of the specified column.

getPrecision(Integer) Method [page 97]
: Returns the precision of the specified column.

getScale(Integer) Method [page 98]
: Returns the scale of the specified column.

getTableName(Integer) Method [page 98]
: Returns the table name for the specified column.

1.6.1 getCatalogName() Method

Returns the database server name.

Syntax

```
resultSetMetaData.getCatalogName ()
```

Returns

Returns the database server name. (type: String)

1.6.2 getColumnCount() Method

Returns the number of columns in the result set.

Syntax

```
resultSetMetaData.getColumnCount ()
```

Returns

Returns the number of columns. (type: Integer)

1.6.3 getColumnLabel(Integer) Method

Returns the alias or name of the specified column.

Syntax

```
resultSetMetaData.getColumnLabel (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the column alias or name. (type: String)

1.6.4 getColumnName(Integer) Method

Returns the name of the specified column.

Syntax

```
resultSetMetaData.getColumnName (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the column name. (type: String)

1.6.5 getColumnType(Integer) Method

Returns a number representing the type of the specified column.

Syntax

```
resultSetMetaData.getColumnType (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the column type as an Integer. (type: Integer)

Remarks

Refer to the a_sqlany_data_type enumeration to find the corresponding type.

1.6.6 getColumnTypeName(Integer) Method

Returns a string containing the name of the type for the specified column.

Syntax

```
resultSetMetaData.getColumnTypeName (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the column type name. (type: String)

1.6.7 getDisplaySize(Integer) Method

Returns the maximum display size of the specified column.

Syntax

```
resultSetMetaData.getDisplaySize (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the maximum display size. (type: Integer)

1.6.8 getPrecision(Integer) Method

Returns the precision of the specified column.

Syntax

```
resultSetMetaData.getPrecision (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the precision. (type: Integer)

1.6.9 getScale(Integer) Method

Returns the scale of the specified column.

Syntax

```
resultSetMetaData.getScale (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the scale. (type: Integer)

1.6.10 getTableName(Integer) Method

Returns the table name for the specified column.

Syntax

```
resultSetMetaData.getTableName (colIndex)
```

Parameters

Type	Name	Description
Integer	colIndex	The index of the column starting from 1. (type: Integer)

Returns

Returns the table name. (type: String)

Remarks

Note: This method is not supported.

Important Disclaimers and Legal Information

www.sap.com/contactsap

THE BEST RUN SAP