JSColumnObject

Constants Summary

Number	DATABASE_IDENTITY Constant used when setting or getting the sequence type of columns.
Number	DATABASE_SEQUENCE Constant used when setting or getting the sequence type of columns.
Number	DATETIME Constant used when setting or getting the type of columns.
Number	EXCLUDED_COLUMN Constant used when setting or getting the flags of columns.
Number	INTEGER Constant used when setting or getting the type of columns.
Number	MEDIA Constant used when setting or getting the type of columns.
Number	NONE Constant for column information indicating unset values.
Number	NUMBER Constant used when setting or getting the type of columns.
Number	PK_COLUMN Constant used when setting or getting the row identifier type of columns.
Number	ROWID_COLUMN Constant used when setting or getting the row identifier type of columns.
Number	SERVOY_SEQUENCE Constant used when setting or getting the sequence type of columns.
Number	TEXT Constant used when setting or getting the type of columns.
Number	UUID_COLUMN Constant used when setting or getting the flags of columns.
Number	UUID_GENERATOR Constant used when setting or getting the sequence type of columns.

Property Summary

Boolean	allowNull Get or set the allow-null flag of a new column.
Number	rowIdentifierType Get or set the row identifier type of the column.
Number	sequenceType Get or set the sequence type of the column.

Method Summary

String	getDataProviderID() Get the data provider id for this column (which is the same as name if not explicitly defined otherwise).
String	getDescription() Get the description property of the column.
String	getForeignType() Get the foreign type of the column.
Number	getLength() Get the length of the column as reported by the JDBC driver.
String	getQualifiedName() Get the qualified name (including table name) of the column as known by the database.
String	getQuotedSQLName() Returns a quoted version of the column name, if necessary, as defined by the actual database used.
String	getSQLName() Get the name of the column as known by the database.
Number	getScale() Get the scale of the column as reported by the JDBC driver.
String	getTitle() Get the title property of the column.
Number	getType() Get the JDBC type of the column.
String	getTypeAsString() Get the name JDBC type of the column.

Boolean	hasFlag(flag) Check a flag of the column.
void	setDatabaseSequenceName() Set the database sequence name of the column, used for columns with sequence type JSColumn.
void	setFlag(flag, set) Set or clear a flag of a new column.

Constants Details

DATABASE_IDENTITY

Constant used when setting or getting the sequence type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getSequenceType())
{
case JSColumn.NONE:
        // handle column with no sequence
break;

case JSColumn.UUID_GENERATOR:
        // handle uuid generated column
break;
}
```

DATABASE_SEQUENCE

Constant used when setting or getting the sequence type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getSequenceType())
{
case JSColumn.NONE:
        // handle column with no sequence
break;

case JSColumn.UUID_GENERATOR:
        // handle uuid generated column
break;
}
```

DATETIME

Constant used when setting or getting the type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

EXCLUDED_COLUMN

Constant used when setting or getting the flags of columns.
This flag identifies columns that are skipped in the sql.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
if (column.hasFlag(JSColumn.UUID_COLUMN))
{
        // handle uuid column
}
```

INTEGER

Constant used when setting or getting the type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

MEDIA

Constant used when setting or getting the type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

NONE

Constant for column information indicating unset values.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getSequenceType())
{
case JSColumn.NONE:
        // handle column with no sequence
break;

case JSColumn.UUID_GENERATOR:
        // handle uuid generated column
break;
}
```

NUMBER

Constant used when setting or getting the type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

PK_COLUMN

Constant used when setting or getting the row identifier type of columns.
This value identifies columns that are defined as primary key in the database.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getRowIdentifierType())
{
case JSColumn.NONE:
        // handle normal column
break;

case JSColumn.PK_COLUMN:
        // handle database pk column
break;

case JSColumn.ROWID_COLUMN:
        // handle developer defined pk column
break;
}
```

ROWID_COLUMN

Constant used when setting or getting the row identifier type of columns.
This value identifies columns that are defined as primary key by the developer (but not in the database).

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getRowIdentifierType())
{
case JSColumn.NONE:
        // handle normal column
break;

case JSColumn.PK_COLUMN:
        // handle database pk column
break;

case JSColumn.ROWID_COLUMN:
        // handle developer defined pk column
break;
}
```

SERVOY_SEQUENCE

Constant used when setting or getting the sequence type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getSequenceType())
{
case JSColumn.NONE:
        // handle column with no sequence
break;

case JSColumn.UUID_GENERATOR:
        // handle uuid generated column
break;
}
```

TEXT

Constant used when setting or getting the type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

UUID_COLUMN

Constant used when setting or getting the flags of columns.
This flag identifies columns whose values are treated as UUID.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
if (column.hasFlag(JSColumn.UUID_COLUMN))
{
        // handle uuid column
}
```

UUID_GENERATOR

Constant used when setting or getting the sequence type of columns.

Returns

Number

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getSequenceType())
{
case JSColumn.NONE:
        // handle column with no sequence
break;

case JSColumn.UUID_GENERATOR:
        // handle uuid generated column
break;
}
```

Property Details

allowNull

Get or set the allow-null flag of a new column.
Note that when a column is added to an existing table, allowNull will always be set.
For a primary key column, the allowNull flag will be always off, for other columns the flag is set by default.

Returns

Boolean

Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server)
{
        var table = server.createNewTable("users");
        if (table)
        {
                var pk = table.createNewColumn("id", JSColumn.MEDIA, 16); // can also use (JSColumn.TEXT,
36) for UUIDs
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.setFlag(JSColumn.UUID_COLUMN, true)
                pk.sequenceType = JSColumn.UUID_GENERATOR
                var c = table.createNewColumn("name", JSColumn.TEXT, 100);
                c.allowNull = false
                table.createNewColumn("age", JSColumn.INTEGER, 0);
                table.createNewColumn("last_login", JSColumn.DATETIME, 0);
                var result = server.synchronizeWithDB(table);
                if (result) application.output("Table successfully created.");
                else application.output("Table not created.");
        }
}
```

rowIdentifierType

Get or set the row identifier type of the column.
The sequence type is one of:
- JSColumn.PK_COLUMN
- JSColumn.ROWID_COLUMN
- JSColumn.NONE

Returns

Number

Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server)
{
        // users has uuid pk
        var table = server.createNewTable("users");
        if (table)
        {
                var pk = table.createNewColumn("id", JSColumn.MEDIA, 16); // can also use <JSColumn.TEXT,
36> for UUIDs)
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.setFlag(JSColumn.UUID_COLUMN, true)
                pk.sequenceType = JSColumn.UUID_GENERATOR
                table.createNewColumn("name", JSColumn.TEXT, 100);
                var result = server.synchronizeWithDB(table);
                if (result) application.output("Table users successfully created.");
                else application.output("Table users not created.");
        }

        // groups has database sequence pk
        table = server.createNewTable("groups");
        if (table)
        {
                pk = table.createNewColumn("id", JSColumn.INTEGER, 0);
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.sequenceType = JSColumn.DATABASE_SEQUENCE
                pk.setDatabaseSequenceName('mygroupsequence')
                table.createNewColumn("name", JSColumn.TEXT, 100);
                result = server.synchronizeWithDB(table);
                if (result) application.output("Table groups successfully created.");
                else application.output("Table groups not created.");
        }
}
```

sequenceType

Get or set the sequence type of the column.
The sequence type is one of:
- JSColumn.NONE
- JSColumn.SERVOY_SEQUENCE
- JSColumn.DATABASE_SEQUENCE
- JSColumn.DATABASE_IDENTITY
- JSColumn.UUID_GENERATOR;

Returns

Number

Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server)
{
        // users has uuid pk
        var table = server.createNewTable("users");
        if (table)
        {
                var pk = table.createNewColumn("id", JSColumn.MEDIA, 16); // can also use <JSColumn.TEXT,
36> for UUIDs)
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.setFlag(JSColumn.UUID_COLUMN, true)
                pk.sequenceType = JSColumn.UUID_GENERATOR
                table.createNewColumn("name", JSColumn.TEXT, 100);
                var result = server.synchronizeWithDB(table);
                if (result) application.output("Table users successfully created.");
                else application.output("Table users not created.");
        }

        // groups has database sequence pk
        table = server.createNewTable("groups");
        if (table)
        {
                pk = table.createNewColumn("id", JSColumn.INTEGER, 0);
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.sequenceType = JSColumn.DATABASE_SEQUENCE
                pk.setDatabaseSequenceName('mygroupsequence')
                table.createNewColumn("name", JSColumn.TEXT, 100);
                result = server.synchronizeWithDB(table);
                if (result) application.output("Table groups successfully created.");
                else application.output("Table groups not created.");
        }
}
```

Method Details

getDataProviderID

String **getDataProviderID** ()

Get the data provider id for this column (which is the same as name if not explicitly defined otherwise).

Returns

String - String dataprovider id.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
var dataProviderId = column.getDataProviderID()
```

getDescription

String **getDescription** ()

Get the description property of the column.

Returns

String - String column description.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customername')
var desc = column.getDescription()
```

getForeignType

String **getForeignType** ()

Get the foreign type of the column.
The foreign type can be defined design time as a foreign key reference to another table.

Returns

String - String foreign type.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
var foreignType = column.getForeignType()
if (foreignType != null)
{
        var fkTable = databaseManager.getTable('example_data', foreignType)
}
```

getLength

Number **getLength** ()

Get the length of the column as reported by the JDBC driver.

Returns

Number - int column length.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customername')
if (column.getLength() < 10)
{
         // handle short column
}
```

getQualifiedName

String **getQualifiedName** ()

Get the qualified name (including table name) of the column as known by the database.
The name is quoted, if necessary, as defined by the actual database used.

Returns

String - String qualified column name.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
var qualifiedSqlName = column.getQualifiedName()
```

getQuotedSQLName

String **getQuotedSQLName** ()

Returns a quoted version of the column name, if necessary, as defined by the actual database used.

Returns

String - column name, quoted if needed.

Sample

```
//use with the raw SQL plugin:
//if the table name contains characters that are illegal in sql, the table name will be quoted
var jsTable = databaseManager.getTable('udm', 'campaigns')
var quotedTableName = jsTable.getQuotedSQLName()
var jsColumn = jsTable.getColumn('active')
var quotedColumnName = jsColumn.getQuotedSQLName()
plugins.rawSQL.executeSQL('udm',  quotedTableName,  'select * from ' + quotedTableName + ' where ' +
quotedColumnName + ' = ?', [1])
```

getSQLName

String **getSQLName** ()

Get the name of the column as known by the database.

Returns

String - String sql name

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
var sqlName = column.getSQLName()
```

getScale

Number **getScale** ()

Get the scale of the column as reported by the JDBC driver.

Returns

Number - int column scale.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customername')
var scale = column.getScale()
```

getTitle

String **getTitle** ()

Get the title property of the column.

Returns

String - String column title.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customername')
var title = column.getTitle()
```

getType

Number **getType** ()

Get the JDBC type of the column.
The type reported by the JDBC driver will be mapped to one of:
- JSColumn.DATETIME
- JSColumn.TEXT
- JSColumn.NUMBER
- JSColumn.INTEGER
- JSColumn.MEDIA

Returns

Number - int sql type.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
switch (column.getType())
{
case JSColumn.TEXT:
        // handle text column
break;

case JSColumn.NUMBER:
case JSColumn.INTEGER:
        // handle numerical column
break;
}
```

getTypeAsString

String **getTypeAsString** ()

Get the name JDBC type of the column.
The same mapping as defined in JSColumn.getType() is applied.

Returns

String - String sql name.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
var typeName = column.getTypeAsString()
```

hasFlag

Boolean **hasFlag** (flag)

Check a flag of the column.
The flags are a bit pattern consisting of 1 or more of the following bits:
- JSColumn.UUID_COLUMN
- JSColumn.EXCLUDED_COLUMN

Parameters

{Number} flag

Returns

Boolean - boolean whether flag is set.

Sample

```
var table = databaseManager.getTable('db:/example_data/orders')
var column = table.getColumn('customerid')
if (column.hasFlag(JSColumn.UUID_COLUMN))
{
        // handle uuid column
}
```

setDatabaseSequenceName

void **setDatabaseSequenceName** ()

Set the database sequence name of the column, used for columns with sequence type JSColumn.DATABASE_SEQUENCE.

Returns

void

Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server)
{
        // users has uuid pk
        var table = server.createNewTable("users");
        if (table)
        {
                var pk = table.createNewColumn("id", JSColumn.MEDIA, 16); // can also use <JSColumn.TEXT,
36> for UUIDs)
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.setFlag(JSColumn.UUID_COLUMN, true)
                pk.sequenceType = JSColumn.UUID_GENERATOR
                table.createNewColumn("name", JSColumn.TEXT, 100);
                var result = server.synchronizeWithDB(table);
                if (result) application.output("Table users successfully created.");
                else application.output("Table users not created.");
        }

        // groups has database sequence pk
        table = server.createNewTable("groups");
        if (table)
        {
                pk = table.createNewColumn("id", JSColumn.INTEGER, 0);
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.sequenceType = JSColumn.DATABASE_SEQUENCE
                pk.setDatabaseSequenceName('mygroupsequence')
                table.createNewColumn("name", JSColumn.TEXT, 100);
                result = server.synchronizeWithDB(table);
                if (result) application.output("Table groups successfully created.");
                else application.output("Table groups not created.");
        }
}
```

setFlag

void **setFlag** (flag, set)

Set or clear a flag of a new column.
The flags are a bit pattern consisting of 1 or more of the following bits:
- JSColumn.UUID_COLUMN;
- JSColumn.EXCLUDED_COLUMN;

Parameters

{Number} flag - the flag to set
{Boolean} set - true for set flag, false for clear flag

Returns

void

Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server)
{
        var table = server.createNewTable("users");
        if (table)
        {
                var pk = table.createNewColumn("id", JSColumn.MEDIA, 16); // can also use (JSColumn.TEXT,
36) for UUIDs
                pk.rowIdentifierType = JSColumn.PK_COLUMN;
                pk.setFlag(JSColumn.UUID_COLUMN, true)
                pk.sequenceType = JSColumn.UUID_GENERATOR
                var c = table.createNewColumn("name", JSColumn.TEXT, 100);
                c.allowNull = false
                table.createNewColumn("age", JSColumn.INTEGER, 0);
                table.createNewColumn("last_login", JSColumn.DATETIME, 0);
                var result = server.synchronizeWithDB(table);
                if (result) application.output("Table successfully created.");
                else application.output("Table not created.");
        }
}
```