

# JSTableObject

## Method Summary

JSColumnObject	<a href="#">createNewColumn</a> (columnName, type, length)	Creates a new column in this table.
JSColumnObject	<a href="#">createNewColumn</a> (columnName, type, length, allowNull)	Creates a new column in this table.
JSColumnObject	<a href="#">createNewColumn</a> (columnName, type, length, allowNull, pkColumn)	Creates a new column in this table.
void	<a href="#">deleteColumn</a> (columnName)	Deletes the column with the specified name from this table.
JSColumn	<a href="#">getColumn</a> (name)	Returns a JSColumn for the named column (or column dataproviderID).
String[]	<a href="#">getColumnNames</a> ()	Returns an array containing the names of all table columns.
String	<a href="#">getDataSource</a> ()	Returns the table data source uri.
String	<a href="#">getQuotedSQLName</a> ()	Returns a quoted version of the table name, if necessary, as defined by the actual database used.
String[]	<a href="#">getRowIdentifierColumnNames</a> ()	Returns an array containing the names of the identifier (PK) column(s).
String	<a href="#">getSQLName</a> ()	Returns the table name.
String	<a href="#">getServerName</a> ()	Returns the Servoy server name.
Boolean	<a href="#">isMetadataTable</a> ()	Returns whether table was flagged as metadata table.

## Method Details

### createNewColumn

**JSColumnObject** [createNewColumn](#) (columnName, type, length)

Creates a new column in this table. The name, type and length of the new column must be specified. For specifying the type of the column, use the JSColumn constants. The column is not actually created in the database until this table is synchronized with the database using the JSServer.synchronizeWithDB method.

The method returns a JSColumn instance that corresponds to the newly created column. If any error occurs and the column cannot be created, then the method returns null.

#### Parameters

{String} columnName  
{Number} type  
{Number} length

#### Returns

JSColumnObject

**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.");
    }
}

```

**createNewColumn**

**JSColumnObject** **createNewColumn** (columnName, type, length, allowNull)

Creates a new column in this table. The name, type and length of the new column must be specified. For specifying the type of the column, use the JSColumn constants. The column is not actually created in the database until this table is synchronized with the database using the JSServer.synchronizeWithDB method.

The method returns a JSColumn instance that corresponds to the newly created column. If any error occurs and the column cannot be created, then the method returns null.

**Parameters**

{String} columnName  
 {Number} type  
 {Number} length  
 {Boolean} allowNull

**Returns**

**JSColumnObject**

**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.");
    }
}

```

**createNewColumn**

**JSColumnObject** **createNewColumn** (columnName, type, length, allowNull, pkColumn)

Creates a new column in this table. The name, type and length of the new column must be specified. For specifying the type of the column, use the JSColumn constants. The column is not actually created in the database until this table is synchronized with the database using the JSServer.synchronizeWithDB method.

The method returns a JSColumn instance that corresponds to the newly created column. If any error occurs and the column cannot be created, then the method returns null.

#### Parameters

{String} columnName  
{Number} type  
{Number} length  
{Boolean} allowNull  
{Boolean} pkColumn

#### Returns

JSColumnObject

#### 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.");
    }
}
```

### deleteColumn

void **deleteColumn** (columnName)

Deletes the column with the specified name from this table. The column is not actually deleted from the database until this table is synchronized with the database using the JSServer.synchronizeWithDB method.

#### Parameters

{String} columnName

#### Returns

void

#### Sample

```
var server = plugins.maintenance.getServer("example_data");
if (server) {
    var table = server.getTable("users");
    if (table) {
        table.deleteColumn("last_login");
        server.synchronizeWithDB(table);
    }
}
```

### getColumn

JSColumn **getColumn** (name)

Returns a JSColumn for the named column (or column dataproviderID).

#### Parameters

{String} name - The name of the column to return the value from.

#### Returns

JSColumn - JSColumn column.

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var jsColumn = jsTable.getColumn('campaign_name')
```

**getColumnNames**

**String[] getColumnNames ()**

Returns an array containing the names of all table columns.

**Returns**

**String[]** - String array of column names.

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var columnNames = jsTable.getColumnNames()
```

**getDataSource**

**String getDataSource ()**

Returns the table data source uri.

**Returns**

**String** - String datasource uri.

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var dataSource = jsTable.getDataSource()
```

**getQuotedSQLName**

**String getQuotedSQLName ()**

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

**Returns**

**String** - String table 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()
plugins.rawSQL.executeSQL('udm', quotedTableName, 'select * from ' + quotedTableName + ' where is_active = ?', [1])
```

**getRowIdentifierColumnNames**

**String[] getRowIdentifierColumnNames ()**

Returns an array containing the names of the identifier (PK) column(s).

**Returns**

**String[]** - String array of row identifier column names.

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var identifierColumnNames = jsTable.getRowIdentifierColumnNames()
```

**getSQLName**

**String getSQLName ()**

Returns the table name.

**Returns**

**String** - String table name.

---

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var tableNameForDisplay = jsTable.getSQLName()
```

**getServerName**

**String** **getServerName** ()

Returns the Servoy server name.

**Returns**

**String** - String server name.

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var serverName = jsTable.getServerName()
```

**isMetadataTable**

**Boolean** **isMetadataTable** ()

Returns whether table was flagged as metadata table.

**Returns**

**Boolean** - boolean is metadata

**Sample**

```
var jsTable = databaseManager.getTable('udm', 'campaigns')
var isMetaDataTable = jsTable.isMetadataTable()
```