TIBCO® Data Virtualization

MongoDB Adapter Guide

Version 8.8.0 | October 2023

Contents

MongoDB Adapter

MongoDB Version Support

The adapter models MongoDB instances as relational databases and supports MongoDB versions 2.6 through 5.0. The adapter leverages the MongoDB API, including the MongoDB aggregation framework, to enable bidirectional SQL access to MongoDB data. See the NoSQL Database chapter for SQL-to-MongoDB query mappings and more information about accessing unstructured data in MongoDB through SQL. See the DBaaS Connections page to connect to popular services such as Atlas and ObjectRocket.

SQL Compliance

The SQL Compliance section shows the SQL syntax supported by the adapter and points out any limitations.

Getting Started

Connecting to MongoDB

Basic Tab shows how to authenticate to MongoDB and configure any necessary connection properties. Additional adapter capabilities can be configured using the available Connection properties on the Advanced tab. The Advanced Settings section shows how to set up more advanced configurations and troubleshoot connection errors.

Deploying the MongoDB Adapter

To deploy the adapter, you can execute the server_util utility via the command line by

1. Unzip the tdv.mongodb.zip file to the location of your choice.
2. Open a command prompt window.
3. Navigate to the <TDV_install_dir>/bin
4. Enter the server_util command with the -deploy option:

```
server_util -server <hostname> [-port <port>] -user <user> -
password <password> -deploy -package <TDV_install_
dir>/adapters/tdv.mongodb/tdv.mongodb.jar
```

Note: When deploying a build of an existing adapter, you will need to undeploy the existing adapter using the server_util command with the -undeploy option.

```
server_util -server <hostname> [-port <port>] -user <user> -password
<password> -undeploy -version 1 -name MongoDB
```

Basic Tab

Connecting to MongoDB

Set the following connection properties to connect to a single MongoDB instance:

- Server: Set this to the name or address of the server your MongoDB instance is running on. You can specify the port here or in Port.
- Database: Set this to the database you want to read from and write to.

Connecting to MongoDB Using DNS Seed Lists

To connect using DNS seed lists

- Server: Set this to "mongodb+srv://"" + the name of the server your MongoDB instance is running on. You can specify the port here or in Port.
- Database: Set this to the database you want to read from and write to.
- DNSServer: Set this to the hostname of a DNSServer that can resolve the necessary DNS entries.

Using DNS seed list connections allows for auto-detection of cluster topologies and more flexibility in deployment. See https://docs.mongodb.com/manual/reference/connection-string/#dns-seed-list-connection-format for more information.

Connecting to Replica Sets

To connect to a replica set, set the following in addition to the preceding connection properties:

- ReplicaSet: Set this to a comma-separated list of secondary servers in the replica set, specified by address and port.
- SlaveOK: Set this to true if you want to read from secondary (slave) servers.
- ReadPreference: Set this to fine-tune how the adapter reads from secondary servers.

Securing MongoDB Connections

You can set UseSSL to negotiate SSL/TLS encryption when you connect.

Authenticating MongoDB Connections

Supported AuthScheme types (MONGODB-CR,SCRAM-SHA-1,SCRAM-SHA-256,PLAIN,GSSAPI) are challenge-response authentication and LDAP.

Challenge-Response

In challenge-response authentication, the User and Password properties correspond to a username and password stored in a MongoDB database. If you want to connect to data from one database and authenticate to another database, set both Database and AuthDatabase.

LDAP

To use LDAP authentication, set AuthDatabase to "$external" and set AuthScheme to PLAIN. This value specifies the SASL PLAIN mechanism; note that this mechanism transmits credentials over plaintext, so it is not suitable for use without TLS/SSL on untrusted networks.

X.509 Certificates

Set AuthScheme to X509 to use X.509 certificate authentication.

Connecting to an Amazon DocumentDB Cluster

Before you can connect to Amazon DocumentDB, you will first need to, ensure your Amazon DocumentDB cluster and the EC2 instance containing the mongo shell are

currently running.

Next, configure an SSH tunnel to the EC2 instance as follows.

1. From the AWS management console, select **Services -> Database -> Amazon DocumentDB**. From the DocumentDB management page, select **Clusters**, then click your cluster.
2. Under the Connect section, **note the --host value** and its port found in the sample connection string.
3. Navigate to **Services -> Compute -> EC2**. Select **Running instances**.
4. Select your instance, then click the **Connect** button.
5. Under the Example section, note the value identifying the instance and user, shown in the form **<ami_username>@<Public DNS>**
6. In your preferred SSH client, establish a connection to your EC2 instance using the Host Name from the EC2 instance's Connect page (username@publicDNS) and Port 22.
7. Provide your EC2 instance's private key file (in Putty, you will need to convert the keys from .pem to .ppk) for authentication.
8. Configure an SSH tunnel using the port and host name from the DocumentDB cluster page.
9. Establish the connection to the EC2 virtual machine.

Specify the following to connect to the DocumentDB cluster.

- Server: Set this to the machine name which is hosting the SSH tunnel.
- Port: Set this to the port the SSH tunnel is hosted on.
- User: Set this to the master username used to provision the DocumentDB cluster.
- Password: Set this to the master password set when provisioning the DocumentDB cluster.
- UseSSL: Set this to true.
- UseFindAPI Set this to true.

Connecting to CosmosDB with the MongoDB API

To obtain the connection string needed to connect to a Cosmos DB account using the MongoDB API, log in to the Azure Portal, select Azure Cosmos DB, and select your account. In the Settings section, click Connection String and set the following values.

- Server: Set this to the Host value, the FQDN of the server provisioned for your account. You can also specify the port here or in Port.
- Port: Set this to the port.
- Database: Set this to the database you want to read from and write to.
- User: Set this to the database user.
- Password: Set this to the user's password.

Logging

The adapter uses TDV Server's logging (log4j) to generate log files. The settings within the TDV Server's logging (log4j) configuration file are used by the adapter to determine the type of messages to log. The following categories can be specified:

- Error: Only error messages are logged.
- Info: Both Error and Info messages are logged.
- Debug: Error, Info, and Debug messages are logged.

The Other property of the adapter can be used to set Verbosity to specify the amount of detail to be included in the log file, that is:

```
Verbosity=4;
```

You can use Verbosity to specify the amount of detail to include in the log within a category. The following verbosity levels are mapped to the log4j categories:

- 0 = Error
- 1-2 = Info
- 3-5 = Debug

For example, if the log4j category is set to DEBUG, the Verbosity option can be set to 3 for the minimum amount of debug information or 5 for the maximum amount of debug information.

Note that the log4j settings override the Verbosity level specified. The adapter never logs at a Verbosity level greater than what is configured in the log4j properties. In addition, if Verbosity is set to a level less than the log4j category configured, Verbosity defaults to the minimum value for that particular category. For example, if Verbosity is set to a value less than 3 and the Debug category is specified, the Verbosity defaults to 3.

The following list is an explanation of the Verbosity levels and the information that they log.

- 1 - Will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
- 2 - Will log everything included in Verbosity 1 and HTTP headers.
- 3 - Will additionally log the body of the HTTP requests.
- 4 - Will additionally log transport-level communication with the data source. This includes SSL negotiation.
- 5 - Will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

Configure Logging for the MongoDB Adapter

By default, logging is turned on without debugging. If debugging information is desired, uncomment the following line in the TDV Server's log4j.properties file (default location of this file is: C:\Program Files\TIBCO\TDV Server <version>\conf\server):

```
log4j.logger.com.cdata=DEBUG
```

The TDV Server must be restarted after changing the log4j.properties file, which can be accomplished by running the composite.bat script located at: C:\Program Files\TIBCO\TDV Server <version>\bin. Note that reauthenticating to the TDV Studio is required after restarting the server.

Here is an example of the calls:

```
.\composite.bat monitor restart
```

All logs for the adapter are written to the "cs_server_dsrc.log" file as specified in the log4j properties.

Note: The "log4j.logger.com.cdata=DEBUG" option is not required if the **Debug Output Enabled** option is set to true within the TDV Studio. To set this option, navigate to **Administrator > Configuration**. Select **Server > Configuration > Debugging** and set the Debug Output Enabled option to **True**.

DBaaS Connections

When you connect to Atlas, ObjectRocket, or another database-as-a-service provider, there typically are a few variations on the procedure outlined in Establishing a Connection. The following sections show how to obtain the necessary connection properties for several popular services.

Atlas

You can authenticate to MongoDB Atlas with a MongoDB user or an LDAP user. The following sections show how to map Atlas connection strings to adapter connection properties. To obtain the Atlas connection string, follow the steps below:

1. In the Clusters view, click Connect for the cluster you want to connect to.
2. Click Connect Your Application.
3. Select either driver option to display a connection string.

Prerequisites

In addition to creating a MongoDB user and/or setting up LDAP, your Atlas project's white-list must include the IP address of the machine the adapter is connecting from. To add an IP address to the white-list, select the Security tab in the Clusters view and then click IP Whitelist -> Add IP Address.

MonogDB User Credentials

Below is an example connection string providing a MongoDB user's credentials.

```
mongodb://USERNAME:PASSWORD@cluster0-shard-00-
00.mongodb.net:27017,cluster0-shard-00-01.mongodb.net:27017,cluster0-
shard-00-02.mongodb.net:27017/test?ssl=true&replicaSet=Cluster0-shard-
0&authSource=admin
```

Below are the corresponding adapter connection properties:

- Server: Set this to the first server in the replica set. Or, you can specify a primary or secondary server here (the adapter will query the servers in Server and ReplicaSet to find the primary).

  ```
  cluster0-shard-00-00.mongodb.net
  ```

- Port: Set this to the port the server is running on (27017 is the default).
- ReplicaSet: Set this to the other servers in the replica set. Server and ReplicaSet together specify all instances in the MongoDB replica set. Specify both the server name and port in ReplicaSet.

  ```
  mycluster0-shard-00-01.mongodb.net:27017,mycluster0-shard-00-
  02.mongodb.net:27017
  ```

- SlaveOK: Set this to true to allow reading from secondary (slave) servers in the replica set.
- AuthDatabase: Set this to "admin" to connect to MongoDB Atlas. All MongoDB users for Atlas are associated with the admin database, their authentication database.
- Database: Set this to the database you want to read from and write to.
- User: Set this to the username of a MongoDB user you added to your MongoDB project.
- Password: Set this to the password of the MongoDB user.
- UseSSL: Set this to true. Atlas requires TLS/SSL.

LDAP

The following list shows the MongoDB Atlas requirements for authenticating with an LDAP user. Below is an example command to connect with the mongo client:

```
mongo "mongodb://cluster0-shard-00-00.mongodb.net:27017,cluster0-shard-
00-01.mongodb.net:27017,cluster0-shard-00-
02.mongodb.net:27017/test?ssl=true&replicaSet=Cluster0-shard-
0&authSource=$external" --authenticationMechanism PLAIN --username
cn=rob,cn=Users,dc=atlas-ldaps-01,dc=myteam,dc=com
```

- Server: Set this to the first server in the replica set. Or, you can specify another

primary or secondary server here (the adapter will query the servers in Server and ReplicaSet to find the primary). For example:

```
cluster0-shard-00-00.mongodb.net
```

- Port: Set this to the port the server is running on (27017 is the default).
- ReplicaSet: Set this to the other servers in the replica set. Server and ReplicaSet together specify all instances in the MongoDB replica set. Below is an example value:

```
mycluster0-shard-00-01.mongodb.net:27017,mycluster0-shard-00-
02.mongodb.net:27017
```

- SlaveOK: Set this to true to allow reading from secondary (slave) servers in the replica set.
- AuthScheme: Set AuthScheme to PLAIN in LDAP authentication.
- Database: Set this to the database you want to read from and write to.
- AuthDatabase: Set this to "$external" to authenticate with an LDAP user.
- User: Set this to the full Distinguished Name (DN) of a user in your LDAP server as the Atlas username. For example:

```
cn=rob,cn=Users,dc=atlas-ldaps-01,dc=myteam,dc=com
```

- Password: Set this to the password of the LDAP user.
- UseSSL: Set this to true. Atlas requires TLS/SSL.

ObjectRocket

To connect to ObjectRocket, you authenticate with the credentials for a database user. You can obtain the necessary connection properties from the control panel: On the Instances page, select your instance and then select the Connect menu to display a MongoDB connection string.

Prerequisites

In addition to adding a user for your database, you also need to allow access to the IP address for the machine the adapter is connecting from. You can configure this by selecting

your instance on the Instances page and then clicking Add ACL.

MongoDB User

Below is an example connection string providing the credentials for a MongoDB user:

```
mongodb://YOUR_USERNAME:YOUR_PASSWORD@abc123-d4-
0.mongo.objectrocket.com:52826,abc123-d4-
2.mongo.objectrocket.com:52826,abc123-d4-
1.mongo.objectrocket.com:52826/YOUR_DATABASE_
NAME?replicaSet=89c04c5db2cf403097d8f2e8ca871a1c
```

Below are the corresponding adapter connection properties:

- Server: Set this to the first server in the replica set. Click Replica Set to obtain the server names. Or, you can specify another primary or secondary server here (the adapter will query the servers in Server and ReplicaSet to find the primary).

  ```
  abc123-d4-0.mongo.objectrocket.com
  ```

- Port: Set this to the port the server is running on (27017 is the default).
- ReplicaSet: Set this to the other servers in the replica set. Server and ReplicaSet together specify all instances in the MongoDB replica set. Below is an example value:

  ```
  abc123-d4-2.mongo.objectrocket.com:52826,abc123-d4-
  1.mongo.objectrocket.com:52826
  ```

- Database: Set this to the database you want to read from and write to. Note that this is also the authentication database for the user you are connecting with; database users cannot interact with other databases outside their database in ObjectRocket.
- User: Set this to the username of a MongoDB user you defined for the Database.
- Password: Set this to the password for the database user.
- UseSSL: Set this to true to enable TLS/SSL.

Changelog

General Changes

Date	Build Number	Change Type	Description
04/25/2023	8515	General	**Removed** • Removed support for the SELECT INTO CSV statement. The core code doesn't support it anymore.
12/26/2022	8395	MongoDB	**Added** • support new SQL API for MongoDB Atlas dedicated cluster.
12/14/2022	8383	General	**Changed** • Added the Default column to the sys_procedureparameters table.
12/13/2022	8382	MongoDB	**Added** • support reading $jsonSchema specification via listCollections command and fetch necessary description into column metadata.
11/29/2022	8368	MongoDB	**Added** • Added support ReadPreferenceTags connection property.
10/31/2022	8339	MongoDB	**Changed** • Doesn't support "eval" stored procedure since it's been deprecated by MongoDB server.
10/13/2022	8321	MongoDB	**Added** • Add support for the UpdateScheme connection property to control execution of the UPDATE statement by using an update or a merge operation.

Date	Build Number	Change Type	Description
09/30/2022	8308	General	**Changed** • Added the IsPath column to the sys_procedureparameters table.
09/28/2022	8306	MongoDB	**Changed** • Derive proper KerberosSPN for MongoDB DNS Seed scenario.
09/13/2022	8291	MongoDB	**Changed** • Enable OP_MSG protocol against MongoDB server higher or equal than 5.1.0.
08/18/2022	8265	MongoDB	**Added** • Updated driver to support communication over OP_MSG protocol.
08/17/2022	8264	General	**Changed** • We now support handling the keyword "COLLATE" as standard function name as well.
03/15/2022	8109	MongoDB	**Added** • Added support for SQL such as "INSERT INTO foo_table(foo_c) VALUES(())" to insert empty BSON array values for the "foo_c" column. • Added support for when the connection property WriteSchema is set to RawValue, new columns may be specified and created in the course of the update statement.
02/07/2022	8073	MongoDB	**Changed**

Date	Build Number	Change Type	Description
			• Made our DateTime to String conversion uniform across our drivers. Previously we returned exactly the string value returned from MongoDB when requesting a DateTime column as a String. Instead, we will now convert the value to a DateTime object before returning a string to ensure pairity with our other tools. • In order to control the response format when retrieving a DateTime column as a String, set the DateTimeStringFormat connection property. For example: DateTimeStringFormat=yyyy-MM-ddTHH:mm:ss.fffZ.
12/22/2021	8026	MongoDB	**Added** • Added the WriteConcern connection property.
12/15/2021	8019	MongoDB	**Changed** • We will now validate if a table exists during the CREATE TABLE statement.
12/14/2021	8018	MongoDB	**Added** • Added support for nested array inserts in a vertically flattened array.
11/11/2021	7986	MongoDB	**Added** • Added support for Mongo X-509 authentication.
09/23/2021	7936	MongoDB	**Added** • Added support for TLS encrypted communication over SSH tunnel.

Date	Build Number	Change Type	Description
09/16/2021	7929	MongoDB	**Added** • Added the attribute "other:tabletype" when rsd schema files are generated from the CreateSchema stored procedure. This attribute indicates if the table schema should map to a collection or view on the MongoDB server.
09/02/2021	7915	General	**Added** • Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/25/2021	7907	MongoDB	**Added** • Added support for better path collision detection for MongoDB server version 4.4 and higher. The change is related to this issue: https://docs.mongodb.com/manual/release-notes/4.4-compatibility/#path-collision-restrictions
08/07/2021	7889	General	**Changed** • Added the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	**Changed** • Added the new sys_primarykeys system table.
07/08/2021	7859	General	**Added** • Added the TCP Logging Module for the logging information happening on the TCP wire protocol. The transport bytes that are incoming and ongoing will be logged at

Date	Build Number	Change Type	Description
			verbosity=5.
05/18/2021	7795	MongoDB	**Added** • Added support for nested array vertical flattening on DataModel=Relational.
04/23/2021	7785	General	**Added** • Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = CONCAT (Col1, " - ", Col2) WHERE Col2 LIKE 'A%'
04/23/2021	7783	General	**Changed** • Updated how display sizes are determined for varchar primary key and foreign key columns so they will match the reported length of the column.
04/16/2021	7776	General	**Added** • Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2 **Changed** • Reduced the length to 255 for varchar primary key and foreign key columns. **Changed** • Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you need to generate new metadata caches if you are currently using CacheMetadata.

Date	Build Number	Change Type	Description
			Changed • Updated index naming convention to avoid duplicates.
04/15/2021	7775	General	**Changed** • Kerberos authentication is updated to use TCP by default, but will fall back to UDP if a TCP connection cannot be established.
04/06/2021	7766	MongoDB	**Deprecated** • The AuthMechanism connection property id deprecated. Use the AuthScheme connection property instead.
11/18/2020	7627	MongoDB	**Added** • Added support for SSH.
08/28/2020	7545	MongoDB	**Changed** • Returned all available databases on listing schema.
06/05/2020	7481	MongoDB	**Added** • Added support for CREATE table with rsd file on GenerateSchemaFiles=OnCreate.

NoSQL Database

MongoDB is a schemaless, document database that provides high performance, availability, and scalability. These features are not necessarily incompatible with a standards-compliant query language like SQL-92. In this section we will show various schemes that the adapter offers to bridge the gap with relational SQL and a document database.

Working with MongoDB Objects as Tables

The adapter models the schemaless MongoDB objects into relational tables and translates SQL queries into MongoDB queries to get the requested data. See Query Mapping for more details on how various MongoDB operations are represented as SQL.

Discovering Schemas Automatically

The Automatic Schema Discovery scheme automatically finds the data types in a MongoDB object by scanning a configured number of rows of the object. You can use RowScanDepth, FlattenArrays, and FlattenObjects to control the relational representation of the collections in MongoDB. You can also write Free-Form Queries not tied to the schema.

Customizing Schemas

Optionally, you can use Custom Schema Definitions to project your chosen relational structure on top of a MongoDB object. This allows you to define your chosen names of columns, their data types, and the location of their values in the collection.

Set GenerateSchemaFiles to save the detected schemas as simple configuration files that are easy to extend. You can persist schemas for all collections in the database or for the results of SELECT queries.

Automatic Schema Discovery

The adapter automatically infers a relational schema by inspecting a series of MongoDB documents in a collection. You can use the RowScanDepth property to define the number of documents the adapter will scan to do so. The columns identified during the discovery process depend on the FlattenArrays and FlattenObjects properties.

Flattening Objects

If FlattenObjects is set, all nested objects will be flattened into a series of columns. For example, consider the following document:

```
{
  id: 12,
  name: "Lohia Manufacturers Inc.",
  address: {street: "Main Street", city: "Chapel Hill", state: "NC"},
  offices: ["Chapel Hill", "London", "New York"],
```

```
  annual_revenue: 35,600,000
}
```

This document will be represented by the following columns:

Column Name	Data Type	Example Value
id	Integer	12
name	String	Lohia Manufacturers Inc.
address.street	String	Main Street
address.city	String	Chapel Hill
address.state	String	NC
offices	String	["Chapel Hill", "London", "New York"]
annual_revenue	Double	35,600,000

If FlattenObjects is not set, then the address.street, address.city, and address.state columns will not be broken apart. The address column of type string will instead represent the entire object. Its value would be *{street: "Main Street", city: "Chapel Hill", state: "NC"}*. See JSON Functions for more details on working with JSON aggregates.

Flattening Arrays

The FlattenArrays property can be used to flatten array values into columns of their own. This is only recommended for arrays that are expected to be short, for example the coordinates below:

```
"coord": [ -73.856077, 40.848447 ]
```

The FlattenArrays property can be set to 2 to represent the array above as follows:

Column Name	Data Type	Example Value
coord.0	Float	-73.856077
coord.1	Float	40.848447

It is best to leave other unbounded arrays as they are and piece out the data for them as needed using JSON Functions.

Free-Form Queries

As discussed in Automatic Schema Discovery, intuited table schemas enable SQL access to unstructured MongoDB data. JSON Functions enable you to use standard JSON functions to summarize MongoDB data and extract values from any nested structures. Custom Schema Definitions enable you to define static tables and give you more granular control over the relational view of your data; for example, you can write schemas defining parent/child tables or fact/dimension tables. However, you are not limited to these schemes.

After connecting you can query any nested structure without flattening the data. Any relations that you can access with FlattenArrays and FlattenObjects can also be accessed with an ad hoc SQL query.

Let's consider an example document from the following Restaurant data set:

```
{
  "address": {
    "building": "1007",
    "coord": [
      -73.856077,
      40.848447
    ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    {
      "grade": "A",
      "score": 2,
      "date": {
```

```
        "$date": "1393804800000"
      }
    },
    {
      "date": {
        "$date": "1378857600000"
      },
      "grade": "B",
      "score": 6
    },
    {
      "score": 10,
      "date": {
        "$date": "1358985600000"
      },
      "grade": "C"
    }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

You can access any nested structure in this document as a column. Use the dot notation to drill down to the values you want to access as shown in the query below. Note that arrays have a zero-based index. For example, the following query retrieves the second grade for the restaurant in the example:

```
SELECT "address.building", "grades.1.grade" FROM restaurants WHERE
restaurant_id = '30075445'
```

The preceding query returns the following results:

Column Name	Data Type	Example Value
address.building	String	1007
grades.1.grade	String	A

Vertical Flattening

It is possible to retrieve an array of documents as if it were a separate table. Take the following JSON structure from the restaurants collection for example:

```
{
  "_id" : ObjectId("568c37b748ddf53c5ed98932"),
  "address" : {
    "building" : "1007",
    "coord" : [-73.856077, 40.848447],
    "street" : "Morris Park Ave",
    "zipcode" : "10462"
  },
  "borough" : "Bronx",
  "cuisine" : "Bakery",
  "grades" : [{
      "date" : ISODate("2014-03-03T00:00:00Z"),
      "grade" : "A",
      "score" : 2
    }, {
      "date" : ISODate("2013-09-11T00:00:00Z"),
      "grade" : "A",
      "score" : 6
    }, {
      "date" : ISODate("2013-01-24T00:00:00Z"),
      "grade" : "A",
      "score" : 10
    }, {
      "date" : ISODate("2011-11-23T00:00:00Z"),
      "grade" : "A",
      "score" : 9
    }, {
      "date" : ISODate("2011-03-10T00:00:00Z"),
      "grade" : "B",
      "score" : 14
    }],
  "name" : "Morris Park Bake Shop",
  "restaurant_id" : "30075445"
}
```

Vertical flattening will allow you to retrieve the grades array as a separate table:

```
SELECT * FROM "restaurants.grades"
```

This query returns the following data set:

date	grade	score	P_id	_index
2014-03-03T00:00:00.000Z	A	2	568c37b748ddf53c5ed98932	1

date	grade	score	P_id	_index
2013-09-11T00:00:00.000Z	A	6	568c37b748ddf53c5ed98932	2
2013-01-24T00:00:00.000Z	A	10	568c37b748ddf53c5ed98932	3

You may also want to include information from the base restaurants table. You can do this with a join. Flattened arrays can only be joined with the root document. The adapter expects the left part of the join is the array document you want to flatten vertically. Disable SupportEnhancedSQL to join nested MongoDB documents -- this type of query is supported through the MongoDB API.

```
SELECT "restaurants"."restaurant_id", "restaurants.grades".* FROM
"restaurants.grades" JOIN "restaurants" WHERE "restaurants".name =
'Morris Park Bake Shop'
```

This query returns the following data set:

restaurant_id	date	grade	score	P_id	_index
30075445	2014-03-03T00:00:00.000Z	A	2	568c37b748ddf53c5ed98932	1
30075445	2013-09-11T00:00:00.000Z	A	6	568c37b748ddf53c5ed98932	2
30075445	2013-01-24T00:00:00.000Z	A	10	568c37b748ddf53c5ed98932	3
30075445	2011-11-23T00:00:00.000Z	A	9	568c37b748ddf53c5ed98932	4
30075445	2011-03-10T00:00:00.000Z	B	14	568c37b748ddf53c5ed98932	5

It's also possible to build queries targeting arrays within other arrays.

Consider this sample Inventory collection:

```
{
	"_id": {
		"$oid": "xxxxxxxxxxxxxxxxxxxxx"
	},
	"Company Branch": "Main Branch",
	"ItemList": [
		{
			"item": "journal",
			"instock": [
				{
					"warehouse": "A",
					"qty": 15
				},
				{
					"warehouse": "B",
					"qty": 45
				}
			]
		},
		{
			"item": "paper",
			"instock": [
				{
					"warehouse": "A",
					"qty": 50
				},
				{
					"warehouse": "B",
					"qty": 5
				}
			]
		}
	]
}
```

Insert data into the nested arrays using the syntax of **<parent array>.<index>.<child array>**, as follows:

```
INSERT INTO [Inventory.ItemList] (p_id, item, [instock.0.warehouse],
[instock.0.qty], [instock.0.price]) VALUES ('xxxxxxxxxxxxxxxxxxxxx',
'NoteBook', 'B', 20, '5$')
```

The Inventory collection after executing the INSERT statement:

```
{
	"_id": {
		"$oid": "xxxxxxxxxxxxxxxxxxxxx"
	},
	"Company Branch": "Main Branch",
	"ItemList": [
		{
			"item": "journal",
			"instock": [
				{
					"warehouse": "A",
					"qty": 15
				},
				{
					"warehouse": "B",
					"qty": 45
				}
			]
		},
		{
			"item": "paper",
			"instock": [
				{
					"warehouse": "A",
					"qty": 50
				},
				{
					"warehouse": "B",
					"qty": 5
				}
			]
		},
		{
			"item": "NoteBook",
			"instock": [
				{
					"warehouse": "B",
					"qty": 20,
					"price": "5$"
				}
			]
		}
	]
}
```

JSON Functions

The adapter can return JSON structures as column values. The adapter enables you to use standard SQL functions to work with these JSON structures. The examples in this section use the following array:

```
[
     { "grade": "A", "score": 2 },
     { "grade": "A", "score": 6 },
     { "grade": "A", "score": 10 },
     { "grade": "A", "score": 9 },
     { "grade": "B", "score": 14 }
]
```

JSON_EXTRACT

The JSON_EXTRACT function can extract individual values from a JSON object. The following query returns the values shown below based on the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_EXTRACT(grades,'[0].grade') AS Grade, JSON_EXTRACT
(grades,'[0].score') AS Score FROM Students;
```

Column Name	Example Value
Grade	A
Score	2

JSON_COUNT

The JSON_COUNT function returns the number of elements in a JSON array within a JSON object. The following query returns the number of elements specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_COUNT(grades,'[x]') AS NumberOfGrades FROM Students;
```

Column Name	Example Value
NumberOfGrades	5

JSON_SUM

The JSON_SUM function returns the sum of the numeric values of a JSON array within a JSON object. The following query returns the total of the values specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_SUM(score,'[x].score') AS TotalScore FROM Students;
```

Column Name	Example Value
TotalScore	41

JSON_MIN

The JSON_MIN function returns the lowest numeric value of a JSON array within a JSON object. The following query returns the minimum value specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_MIN(score,'[x].score') AS LowestScore FROM Students;
```

Column Name	Example Value
LowestScore	2

JSON_MAX

The JSON_MAX function returns the highest numeric value of a JSON array within a JSON object. The following query returns the maximum value specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_MAX(score,'[x].score') AS HighestScore FROM Students;
```

Column Name	Example Value
HighestScore	14

DOCUMENT

The DOCUMENT function can be used to retrieve the entire document as a JSON string. See the following query and its result as an example:

```
SELECT DOCUMENT(*) FROM Customers;
```

The query above will return the entire document as shown.

```
{ "id": 12, "name": "Lohia Manufacturers Inc.", "address": { "street":
"Main Street", "city": "Chapel Hill", "state": "NC"}, "offices": [
"Chapel Hill", "London", "New York" ], "annual_revenue": 35,600,000 }
```

Query Mapping

The adapter maps SQL queries into the corresponding MongoDB queries. A detailed description of all the transformations is out of scope, but we will describe some of the common elements that are used. The adapter takes advantage of MongoDB features such as the aggregation framework to compute the desired results.

SELECT Queries

The SELECT statement is mapped to the find() function as shown below:

SQL Query	MongoDB Query
`SELECT * FROM Users`	`db.users.find()`

SQL Query	MongoDB Query
`SELECT user_id, status FROM Users`	`db.users.find( {}, { user_id: 1, status: 1, _id: 0 } )`
`SELECT * FROM Users WHERE status = 'A'`	`db.users.find( { status: "A" } )`
`SELECT * FROM Users WHERE status = 'A' OR age=50`	`db.users.find( { $or: [ { status: "A" }, { age: 50 } ] } )`
`SELECT * FROM Users WHERE name LIKE 'A%'`	`db.users.find( {name: /^a/} )`
`SELECT * FROM Users WHERE status = 'A' ORDER BY user_id ASC`	`db.users.find( { status: "A" }.sort( { user_id: 1 } )`
`SELECT * FROM Users WHERE status = 'A' ORDER BY user_id DESC`	`db.users.find( {status: "A" }.sort( {user_id: -1} )`

Aggregate Queries

The MongoDB aggregation framework was added in MongoDB version 2.2. The adapter makes extensive use of this for various aggregate queries. See some examples below:

SQL Query	MongoDB Query

`SELECT Count(*) As Count`
`FROM Orders`
	`db.orders.aggregate( [`
	`  {`
	`  $group: {`
	`  _id: null,`
	`  count: { $sum: 1 }`
	`  }`
	`  }`
	`] )`

`SELECT Sum(price) As Total`
`FROM Orders`
	`db.orders.aggregate( [`
	`  {`
	`  $group: {`
	`  _id: null,`
	`  total: { $sum:`
	`"$price" }`
	`  }`
	`  }`
	`] )`

`SELECT cust_id, Sum(price) As total`
`FROM Orders`
`GROUP BY cust_id`
`ORDER BY total`
	`db.orders.aggregate( [`
	`  {`
	`  $group: {`
	`  _id: "$cust_id",`
	`  total: { $sum:`
	`"$price" }`
	`  }`
	`  } ,`
	`  { $sort: {total: 1 } }`

SQL Query	MongoDB Query
	])

SELECT cust_id, ord_date, Sum(price) As total
FROM Orders
GROUP BY cust_id, ord_date
HAVING total > 250
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id: "$cust_id",
	ord_date: {
	month: { $month: "$ord_date" },
	day: { $dayOfMonth: "$ord_date" },
	year: { $year: "$ord_date"}
	}
	},
	total: { $sum: "$price" }
	}
	},
	{ $match: { total: { $gt: 250 } } }
	])

INSERT Statements

The INSERT statement is mapped to the INSERT function as shown below:

SQL Query	MongoDB Query
INSERT INTO users (user_id, age, status,	db.users.ins

SQL Query	MongoDB Query
[address.city], [address.postalcode]) VALUES ('bcd001', 45, 'A', 'Chapel Hill', 27517)	ert({ user_id: "bcd001", age: 45, status: "A", address:{ city:"Chapel Hill", postalCode:2 7514} })
INSERT INTO t1 ("c1") VALUES (('a1', 'a2', 'a3'))	db.users.ins ert({"c1": ['a1', 'a2', 'a3']})
INSERT INTO t1 ("c1") VALUES (())	db.users.ins ert({"c1": []})
INSERT INTO t1 ("a.b.c.c1") VALUES (('a1', 'a2', 'a3'))	db.users.ins ert("a": {"b":{"c": {"c1": ['a1','a2', 'a3']}}})

Update Statements

The UPDATE statement is mapped to the update function as shown below:

SQL Query	MongoDB Query
`UPDATE users SET status = 'C', [address.postalcode] = 90210 WHERE age > 25`	`db.users.update( { age: { $gt: 25 } }, { $set: { status: "C", address.postalCode: 90210 }, { multi: true } )`

Delete Statements

The DELETE statement is mapped to the delete function as shown below:

SQL Query	MongoDB Query
`DELETE FROM users WHERE status = 'D'`	`db.users.remove( { status: "D" } )`

Custom Schema Definitions

You can extend the table schemas created with Automatic Schema Discovery by saving them into schema files. The schema files have a simple format that makes the schemas to edit.

Generating Schema Files

Set GenerateSchemaFiles to "OnStart" to persist schemas for all tables when you connect. You can also generate table schemas as needed: Set GenerateSchemaFiles to "OnUse" and execute a SELECT query to the table.

For example, consider a schema for the restaurants data set. This is a sample data set provided by MongoDB. To download the data set, follow the Getting Started with MongoDB guide.

Below is an example document from the collection:

```
{
  "address":{
    "building":"461",
      "coord":[
        -74.138492,
        40.631136
      ],
      "street":"Port Richmond Ave",
      "zipcode":"10302"
   },
   "borough":"Staten Island",
   "cuisine":"Other",
   "name":"Indian Oven",
   "restaurant_id":"50018994"
}
```

Importing the MongoDB Restaurant Data Set

You can use the mongoimport utility to import the data set:

```
mongoimport --db test --collection restaurants --drop --file
dataset.json
```

Customizing a Schema

When GenerateSchemaFiles is set, the adapter saves schemas into the folder specified by the Location property. You can then change column behavior in the resulting schema.

The following schema uses the *other:bsonpath* property to define where in the collection to retrieve the data for a particular column. Using this model you can flatten arbitrary levels of hierarchy.

The *collection* attribute specifies the collection to parse. The *collection* attribute gives you the flexibility to use multiple schemas for the same collection. If *collection* is not specified, the filename determines the collection that is parsed.

Below are the column definitions and the collection to extract the column values from. In Custom Schema Example, you will find the complete schema.

```
<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">
  <rsb:info title="StaticRestaurants" description="Custom Schema for the
MongoDB restaurants data set.">
    <!-- Column definitions -->
    <attr  name="borough"   xs:type="string" other:bsonpath="$.borough"
           />
    <attr  name="cuisine"   xs:type="string" other:bsonpath="$.cuisine"
           />
    <attr  name="building"  xs:type="string"
other:bsonpath="$.address.building"     />
    <attr  name="street"    xs:type="string"
other:bsonpath="$.address.street"       />
    <attr  name="latitude"  xs:type="double"
other:bsonpath="$.address.coord.0"      />
    <attr  name="longitude" xs:type="double"
other:bsonpath="$.address.coord.1"      />
  </rsb:info>
  <rsb:set attr="collection" value="restaurants"/>
</rsb:script>
```

Custom Schema Example

This section contains an example of a complete schema that has been automatically generated by GenerateSchemaFiles. Set the Location property to the file directory that will contain the schema file. The schema consists of the following parts:

- The *info* section enables a relational view of a MongoDB object. For more details, see Custom Schema Definitions.
- The *collection* attribute specifies the name of the collection to be parsed. The *collection* attribute can be used to define multiple schemas for the same collection. If *collection* is not specified, the filename determines the collection that is parsed.
- The GET, POST, MERGE, and DELETE methods allow SELECT, INSERT, UPDATE, and DELETE commands to this table. The *coreExecOperation* operation is an internal implementation and can be copied as-is to your own custom schema file.

```
<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">
  <rsb:info title="StaticRestaurants" description="Automatic
GenerateSchemaFile">
    <!-- Column definitions -->
    <attr  name="borough"           xs:type="string"
other:bsonpath="$.borough"          />
    <attr  name="cuisine"           xs:type="string"
other:bsonpath="$.cuisine"          />
    <attr  name="address_building"  xs:type="string"
other:bsonpath="$.address.building" />
    <attr  name="address_street"    xs:type="string"
other:bsonpath="$.address.street"   />
    <attr  name="address_coord_0"   xs:type="double"
other:bsonpath="$.address.coord.0"  />
    <attr  name="address_coord_1"   xs:type="double"
other:bsonpath="$.address.coord.1"  />
  </rsb:info>
  <rsb:set attr="collection" value="restaurants"/>
  <rsb:script method="GET">
    <rsb:call op="coreExecOperation" out="toout">
      <rsb:push item="toout"/>
    </rsb:call>
  </rsb:script>
  <rsb:script method="POST">
    <rsb:call op="coreExecOperation" out="toout">
      <rsb:push item="toout"/>
    </rsb:call>
  </rsb:script>
  <rsb:script method="DELETE">
    <rsb:call op="coreExecOperation" out="toout">
      <rsb:push item="toout"/>
    </rsb:call>
  </rsb:script>
  <rsb:script method="MERGE">
    <rsb:call op="coreExecOperation" out="toout">
      <rsb:push item="toout"/>
    </rsb:call>
  </rsb:script>
</rsb:script>
```

Data Type Mapping

Data Type Mappings

The adapter maps types from the data source to the corresponding data type available in the schema. The table below documents these mappings.

MongoDB	CData Schema
ObjectId	bson:ObjectId
Double	double
Decimal	decimal
String	string
Object	string
Array	bson:Array
Binary	string
Boolean	bool
Date	datetime
Null	bson:Null
Regex	bson:Regex
Integer	int
Long	long
MinKey	bson:MinKey
MaxKey	bson:MaxKey

Stored Procedures

Stored procedures are function-like interfaces that extend the functionality of the adapter beyond simple SELECT/INSERT/UPDATE/DELETE operations with MongoDB.

Stored procedures accept a list of parameters, perform their intended function, and then return, if applicable, any relevant response data from MongoDB, along with an indication of whether the procedure succeeded or failed.

MongoDB Adapter Stored Procedures

Name	Description
AddDocument	Insert entire JSON documents to MongoDB as-is.
CreateSchema	Creates a schema file for the collection.
CreateUserTable	Creates a schema file for the collection.
GetDocument	Take a pass-through query to retrieve documents.
SearchDocument	Get the entire document as a string.

AddDocument

Insert entire JSON documents to MongoDB as-is.

Input

Name	Type	Accepts Input Streams	Description
Collection	*String*	*False*	The collection name to be inserted.
Document	*String*	*True*	The JSON document to be inserted.

Result Set Columns

Name	Type	Description
Success	*String*	Returns true if the operation is successful, else an exception is returned.

CreateSchema

Creates a schema file for the collection.

CreateSchema

Creates a local schema file (.rsd) from an existing table or view in the data model.

The schema file is created in the directory set in the Location connection property when this procedure is executed. You can edit the file to include or exclude columns, rename columns, or adjust column datatypes.

The adapter checks the Location to determine if the names of any .rsd files match a table or view in the data model. If there is a duplicate, the schema file will take precedence over the default instance of this table in the data model. If a schema file is present in Location that does not match an existing table or view, a new table or view entry is added to the data model of the adapter.

Input

Name	Type	Description
SchemaName	*String*	The schema of the collection.
TableName	*String*	The name of the collection.

Name	Type	Description
FileName	*String*	The full file path and name of the schema to generate.
TableType	*String*	The table type of rsd to generate, 'TABLE', 'VIEW', 'UNKNOWN'

Result Set Columns

Name	Type	Description
Result	*String*	Returns Success or Failure.

CreateUserTable

Creates a schema file for the collection.

Input

Name	Type	Description
CatalogName	*String*	The catalog of the collection.
SchemaName	*String*	The schema of the collection.
TableName	*String*	The name of the collection.

Name	Type	Description
Location	*String*	The location where the file is saved.
ColumnNames#	*String*	The name of column.
ColumnDataTypes#	*String*	The datatype of column.
ColumnSizes#	*String*	The size of column.
ColumnScales#	*String*	The scale of column.
ColumnIsKeys#	*String*	The isKey of column.
ColumnIsNulls#	*String*	The isNull of column.
ColumnDefaults#	*String*	The default value of column.
ColumnAutoIncrements#	*String*	The AutoIncrement of column.

Result Set Columns

Name	Type	Description
AffectedTables	*String*	The number of tables created, either 0 or 1

GetDocument

Take a pass-through query to retrieve documents.

Input

Name	Type	Description
Collection	*String*	The collection name to be inserted.
Query	*String*	The Mongo pass-through JSON-style query.
Projection	*String*	The Mongo pass-through JSON-style projection.

Result Set Columns

Name	Type	Description
*	*String*	Output will vary for each collection.

SearchDocument

Get the entire document as a string.

Input

Name	Type	Description
Collection	*String*	The collection name to search.
_id	*String*	The primary key value of the collection.

Result Set Columns

Name	Type	Description
Document	*String*	Returns the entire document as a string.

Advanced Features

This section details a selection of advanced features of the MongoDB adapter.

User Defined Views

The adapter allows you to define virtual tables, called *user defined views*, whose contents are decided by a pre-configured query. These views are useful when you cannot directly control queries being issued to the drivers. See User Defined Views for an overview of creating and configuring custom views.

SSL Configuration

Use SSL Configuration to adjust how adapter handles TLS/SSL certificate negotiations. You can choose from various certificate formats; see the SSLServerCert property under "Connection String Options" for more information.

Firewall and Proxy

Configure the adapter for compliance with Firewall and Proxy, including Windows proxies. You can also set up tunnel connections.

Query Processing

The adapter offloads as much of the SELECT statement processing as possible to MongoDB and then processes the rest of the query in memory (client-side).

See Query Processing for more information.

Logging

See Logging for an overview of configuration settings that can be used to refine CData logging. For basic logging, you only need to set two connection properties, but there are numerous features that support more refined logging, where you can select subsets of information to be logged using the LogModules connection property.

User Defined Views

The MongoDB Adapter allows you to define a virtual table whose contents are decided by a pre-configured query. These are called *User Defined Views*, which are useful in situations where you cannot directly control the query being issued to the driver, e.g. when using the driver from a tool. The User Defined Views can be used to define predicates that are always applied. If you specify additional predicates in the query to the view, they are combined with the query already defined as part of the view.

There are two ways to create user defined views:

- Create a JSON-formatted configuration file defining the views you want.
- DDL statements.

Defining Views Using a Configuration File

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the UserDefinedViews connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom SQL query for the view.

For example:

```
{
	"MyView": {
		"query": "SELECT * FROM [CData].[Sample].Customers WHERE MyColumn =
'value'"
	},
	"MyView2": {
		"query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
	}
}
```

Use the UserDefinedViews connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

Defining Views Using DDL Statements

The adapter is also capable of creating and altering the schema via DDL Statements such as CREATE LOCAL VIEW, ALTER LOCAL VIEW, and DROP LOCAL VIEW.

Create a View

To create a new view using DDL statements, provide the view name and query as follows:

```
CREATE LOCAL VIEW [MyViewName] AS SELECT * FROM Customers LIMIT 20;
```

If no JSON file exists, the above code creates one. The view is then created in the JSON configuration file and is now discoverable. The JSON file location is specified by the UserDefinedViews connection property.

Alter a View

To alter an existing view, provide the name of an existing view alongside the new query you would like to use instead:

```
ALTER LOCAL VIEW [MyViewName] AS SELECT * FROM Customers WHERE
TimeModified > '3/1/2020';
```

The view is then updated in the JSON configuration file.

Drop a View

To drop an existing view, provide the name of an existing schema alongside the new query you would like to use instead.

```
DROP LOCAL VIEW [MyViewName]
```

This removes the view from the JSON configuration file. It can no longer be queried.

Schema for User Defined Views

User Defined Views are exposed in the **UserViews** schema by default. This is done to avoid the view's name clashing with an actual entity in the data model. You can change the name of the schema used for UserViews by setting the UserViewsSchemaName property.

Working with User Defined Views

For example, a SQL statement with a User Defined View called *UserViews.RCustomers* only lists customers in Raleigh:

```
SELECT * FROM Customers WHERE City = 'Raleigh';
```

An example of a query to the driver:

```
SELECT * FROM UserViews.RCustomers WHERE Status = 'Active';
```

Resulting in the effective query to the source:

```
SELECT * FROM Customers WHERE City = 'Raleigh' AND Status = 'Active';
```

That is a very simple example of a query to a User Defined View that is effectively a combination of the view query and the view definition. It is possible to compose these queries in much more complex patterns. All SQL operations are allowed in both queries and are combined when appropriate.

SSL Configuration

Customizing the SSL Configuration

By default, the adapter attempts to negotiate SSL/TLS by checking the server's certificate against the system's trusted certificate store.

To specify another certificate, see the SSLServerCert property for the available formats to do so.

Client SSL Certificates

The MongoDB adapter also supports setting client certificates. Set the following to connect using a client certificate.

- SSLClientCert: The name of the certificate store for the client certificate.
- SSLClientCertType: The type of key store containing the TLS/SSL client certificate.
- SSLClientCertPassword: The password for the TLS/SSL client certificate.
- SSLClientCertSubject: The subject of the TLS/SSL client certificate.

Firewall and Proxy

Connecting Through a Firewall or Proxy

Set the following properties:

- To use a proxy-based firewall, set FirewallType, FirewallServer, and FirewallPort.

- To tunnel the connection, set FirewallType to TUNNEL.
- To authenticate, specify FirewallUser and FirewallPassword.
- To authenticate to a SOCKS proxy, additionally set FirewallType to SOCKS5.

Query Processing

Query Processing

CData has a client-side SQL engine built into the adapter library. This enables support for the full capabilities that SQL-92 offers, including filters, aggregations, functions, etc.

For sources that do not support SQL-92, the adapter offloads as much of SQL statement processing as possible to MongoDB and then processes the rest of the query in memory (client-side). This results in optimal performance.

For data sources with limited query capabilities, the adapter handles transformations of the SQL query to make it simpler for the adapter. The goal is to make smart decisions based on the query capabilities of the data source to push down as much of the computation as possible. The MongoDB Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The MongoDB Query Slicer component is used in more specific cases to separate a single query into multiple independent queries. The client-side Query Engine makes decisions about simplifying queries, breaking queries into multiple queries, and pushing down or computing aggregations on the client-side while minimizing the size of the result set.

There's a significant trade-off in evaluating queries, even partially, client-side. There are always queries that are impossible to execute efficiently in this model, and some can be particularly expensive to compute in this manner. CData always pushes down as much of the query as is feasible for the data source to generate the most efficient query possible and provide the most flexible query capabilities.

More Information

For a full discussion of how CData handles query processing, see CData Architecture: Query Execution.

Logging

Capturing adapter logging can be very helpful when diagnosing error messages or other unexpected behavior.

Basic Logging

You will simply need to set two connection properties to begin capturing adapter logging.

- Logfile: A filepath which designates the name and location of the log file.
- Verbosity: This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for an explanation of the five levels.
- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.
- MaxLogFileCount: A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Once this property is set, the adapter will populate the log file as it carries out various tasks, such as when authentication is performed or queries are executed. If the specified file doesn't already exist, it will be created.

Log Verbosity

The verbosity level determines the amount of detail that the adapter reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are described in the following list:

1	**Setting Verbosity to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.**
2	Setting Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request.
3	Setting Verbosity to 3 will additionally log the body of the request and the response.
4	Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.

1	**Setting Verbosity to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.**
5	Setting Verbosity to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosities, which can delay execution times.

To refine the logged content further by showing/hiding specific categories of information, see LogModules.

Sensitive Data

Verbosity levels 3 and higher may capture information that you do not want shared outside of your organization. The following lists information of concern for each level:

- Verbosity 3: The full body of the request and the response, which includes all the data returned by the adapter
- Verbosity 4: SSL certificates
- Verbosity 5: Any extra transfer data not included at Verbosity 3, such as non human-readable binary transfer data

Best Practices for Data Security

Although we mask sensitive values, such as passwords, in the connection string and any request in the log, it is always best practice to review the logs for any sensitive information before sharing outside your organization.

Java Logging

When Java logging is enabled in Logfile, the Verbosity will instead map to the following logging levels.

- 0: Level.WARNING
- 1: Level.INFO
- 2: Level.CONFIG

- 3: Level.FINE
- 4: Level.FINER
- 5: Level.FINEST

Advanced Logging

You may want to refine the exact information that is recorded to the log file. This can be accomplished using the LogModules property.

This property allows you to filter the logging using a semicolon-separated list of logging modules.

All modules are four characters long. **Please note that modules containing three letters have a required trailing blank space**. The available modules are:

- **EXEC**: Query Execution. Includes execution messages for original SQL queries, parsed SQL queries, and normalized SQL queries. Query and page success/failure messages appear here as well.
- **INFO**: General Information. Includes the connection string, driver version (build number), and initial connection messages.
- **HTTP**: HTTP Protocol messages. Includes HTTP requests/responses (including POST messages), as well as Kerberos related messages.
- **SSL** : SSL certificate messages.
- **OAUT**: OAuth related failure/success messages.
- **SQL** : Includes SQL transactions, SQL bulk transfer messages, and SQL result set messages.
- **META**: Metadata cache and schema messages.
- **TCP** : Incoming and Ongoing raw bytes on TCP transport layer messages.

An example value for this property would be.

```
LogModules=INFO;EXEC;SSL ;SQL ;META;
```

Note that these modules refine the information as it is pulled after taking the Verbosity into account.

SQL Compliance

The MongoDB Adapter supports several operations on data, including querying, deleting, modifying, and inserting.

SELECT Statements

See SELECT Statements for a syntax reference and examples.

See NoSQL Database for information on the capabilities of the MongoDB API.

INSERT Statements

See INSERT Statements for a syntax reference and examples.

UPDATE Statements

The primary key _id is required to update a record. See UPDATE Statements for a syntax reference and examples.

DELETE Statements

The primary key _id is required to delete a record. See DELETE Statements for a syntax reference and examples.

CREATE TABLE Statements

See CREATE TABLE Statements for a syntax reference and examples.

DROP TABLE Statements

See DROP TABLE Statements for a syntax reference and examples.

EXECUTE Statements

Use EXECUTE or EXEC statements to execute stored procedures. See EXECUTE Statements for a syntax reference and examples.

Names and Quoting

- Table and column names are considered identifier names; as such, they are restricted to the following characters: [A-Z, a-z, 0-9, _:@].
- To use a table or column name with characters not listed above, the name must be quoted using double quotes ("name") in any SQL statement.
- Strings must be quoted using single quotes (e.g., 'John Doe').

SELECT Statements

A SELECT statement can consist of the following basic clauses.

- SELECT
- INTO
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY
- LIMIT

SELECT Syntax

The following syntax diagram outlines the syntax supported by the MongoDB adapter:

```
SELECT {
  [ TOP <numeric_literal> | DISTINCT ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
  }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
```

```
    FROM <table_reference> [ [ AS ] <identifier> ]
  } [ , ... ]
  [ [
      INNER
    ] JOIN <table_reference> [ ON <search_condition> ] [ [ AS ]
<identifier> ]
  ] [ ... ]
  [ WHERE <search_condition> ]
  [ GROUP BY <column_reference> [ , ... ]
  [ HAVING <search_condition> ]
  [
    ORDER BY
    <column_reference> [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
  ]
  [
    LIMIT <expression>
    [
      { OFFSET | , }
      <expression>
    ]
  ]
} | SCOPE_IDENTITY()
  <expression> ::=
    | <column_reference>
    | @ <parameter>
    | ?
    | COUNT( * | { [ DISTINCT ] <expression> } )
    | { AVG | MAX | MIN | SUM | COUNT } ( <expression> )
    | NULLIF ( <expression> , <expression> )
    | COALESCE ( <expression> , ... )
    | CASE <expression>
        WHEN { <expression> | <search_condition> } THEN { <expression> |
NULL } [ ... ]
    [ ELSE { <expression> | NULL } ]
      END
    | <literal>
    | <sql_function>
  <search_condition> ::=
    {
      <expression> { = | > | < | >= | <= | <> | != | LIKE | IN | NOT IN
| AND | OR } [ <expression> ]
    } [ { AND | OR } ... ]
```

Examples

1. Return all columns:

```
SELECT * FROM [CData].[Sample].Customers
```

2. Rename a column:

```
SELECT "CompanyName" AS MY_CompanyName FROM [CData].
[Sample].Customers
```

3. Cast a column's data as a different data type:

```
SELECT CAST(Balance AS VARCHAR) AS Str_Balance FROM [CData].
[Sample].Customers
```

4. Search data:

```
SELECT * FROM [CData].[Sample].Customers WHERE Country = 'US'
```

5. The MongoDB APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, LIKE, IN, NOT IN, AND, OR.

```
SELECT * FROM [CData].[Sample].Customers WHERE Country = 'US';
```

6. Return the number of items matching the query criteria:

```
SELECT COUNT(*) AS MyCount FROM [CData].[Sample].Customers
```

7. Return the number of unique items matching the query criteria:

```
SELECT COUNT(DISTINCT CompanyName) FROM [CData].[Sample].Customers
```

8. Return the unique items matching the query criteria:

```
SELECT DISTINCT CompanyName FROM [CData].[Sample].Customers
```

9. Summarize data:

```
SELECT CompanyName, MAX(Balance) FROM [CData].[Sample].Customers
```

```
GROUP BY CompanyName
```

See Aggregate Functions for details.

10. Retrieve data from multiple tables.

```
SELECT "restaurants"."restaurant_id", "restaurants".name,
"restaurants.grades".* FROM "restaurants.grades" JOIN "restaurants"
WHERE "restaurants".name = 'Morris Park Bake Shop'
```

See JOIN Queries for details.

11. Sort a result set in ascending order:

```
SELECT City, CompanyName FROM [CData].[Sample].Customers  ORDER BY
CompanyName ASC
```

Aggregate Functions

Examples of Aggregate Functions

Below are several examples of SQL aggregate functions. You can use these with a GROUP BY clause to aggregate rows based on the specified GROUP BY criterion. This can be a reporting tool.

COUNT

Returns the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM [CData].[Sample].Customers WHERE Country = 'US'
```

COUNT(DISTINCT)

Returns the number of distinct, non-null field values matching the query criteria.

```
SELECT COUNT(DISTINCT City) AS DistinctValues FROM [CData].
[Sample].Customers WHERE Country = 'US'
```

AVG

Returns the average of the column values.

```
SELECT CompanyName, AVG(Balance) FROM [CData].[Sample].Customers WHERE
Country = 'US' GROUP BY CompanyName
```

MIN

Returns the minimum column value.

```
SELECT MIN(Balance), CompanyName FROM [CData].[Sample].Customers WHERE
Country = 'US' GROUP BY CompanyName
```

MAX

Returns the maximum column value.

```
SELECT CompanyName, MAX(Balance) FROM [CData].[Sample].Customers WHERE
Country = 'US' GROUP BY CompanyName
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(Balance) FROM [CData].[Sample].Customers WHERE Country = 'US'
```

JOIN Queries

The MongoDB Adapter supports joins of a nested array with its parent document.

Joining Nested Structures

The adapter expects the left part of the join is the array document you want to flatten vertically. This type of query is supported through the MongoDB API.

For example, consider the following query from MongoDB's restaurants collection:

```
SELECT "restaurants"."restaurant_id", "restaurants".name,
"restaurants.grades".*
FROM "restaurants.grades"
JOIN "restaurants"
WHERE "restaurants".name = 'Morris Park Bake Shop'
```

See Vertical Flattening for more details.

INSERT Statements

To create new records, use INSERT statements.

INSERT Syntax

The INSERT statement specifies the columns to be inserted and the new column values. You can specify the column values in a comma-separated list in the VALUES clause, as shown in the following example:

```
INSERT INTO <table_name>
( <column_reference> [ , ... ] )
VALUES
( { <expression> | NULL } [ , ... ] )

<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

You can use the executeUpdate method of the Statement and PreparedStatement classes to execute data manipulation commands and retrieve the rows affected.

```
String cmd = "INSERT INTO [CData].[Sample].Customers (CompanyName)
VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "Caterpillar");
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
connection.close();
```

UPDATE Statements

To modify existing records, use UPDATE statements.

Update Syntax

The UPDATE statement takes as input a comma-separated list of columns and new column values as name-value pairs in the SET clause, as shown in the following example:

```
UPDATE <table_name> SET { <column_reference> = <expression> } [ , ... ]
WHERE { _id = <expression>  } [ { AND | OR } ... ]
<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

You can use the executeUpdate method of the Statement or PreparedStatement classes to execute data manipulation commands and retrieve the rows affected, as shown in the following example:

```
String cmd = "UPDATE [CData].[Sample].Customers SET
CompanyName='Caterpillar' WHERE _id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
int count = pstmt.executeUpdate();
System.out.println(count + " rows were affected");
connection.close();
```

DELETE Statements

To delete information from a table, use DELETE statements.

DELETE Syntax

The DELETE statement requires the table name in the FROM clause and the row's primary key in the WHERE clause, as shown in the following example:

```
<delete_statement> ::= DELETE FROM <table_name> WHERE { _id =
<expression> } [ { AND | OR } ... ]
<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

You can use the executeUpdate method of the Statement or PreparedStatement classes to execute data manipulation commands and retrieve the number of affected rows, as shown in the following example:

```
Connection connection = DriverManager.getConnection
("jdbc:mongodb:Server=127.0.0.1;Port=27017;Database=test;User=test;Passw
ord=test;",);
String cmd = "DELETE FROM [CData].[Sample].Customers WHERE _id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
int count=pstmt.executeUpdate();
connection.close();
```

EXECUTE Statements

To execute stored procedures, you can use EXECUTE or EXEC statements.

EXEC and EXECUTE assign stored procedure inputs, referenced by name, to values or parameter names.

Stored Procedure Syntax

To execute a stored procedure as an SQL statement, use the following syntax:

```
{ EXECUTE | EXEC } <stored_proc_name>
{
  [ @ ] <input_name> = <expression>
} [ , ... ]
<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

Example Statements

Reference stored procedure inputs by name:

```
EXECUTE my_proc @second = 2, @first = 1, @third = 3;
```

Execute a parameterized stored procedure statement:

```
EXECUTE my_proc second = @p1, first = @p2, third = @p3;
```

PIVOT and UNPIVOT

PIVOT and UNPIVOT can be used to change a table-valued expression into another table.

PIVOT

PIVOT rotates a table-value expression by turning unique values from one column into multiple columns in the output. PIVOT can run aggregations where required on any column value.

PIVOT Synax

```
"SELECT 'AverageCost' AS Cost_Sorted_By_Production_Days, [0], [1], [2],
[3], [4]
FROM
(
SELECT DaysToManufacture, StandardCost
FROM Production.Product
) AS SourceTable
PIVOT
(
AVG(StandardCost)
FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;"
```

UNPIVOT

UNPIVOT carries out nearly the opposite to PIVOT by rotating columns of a table-valued expressions into column values.

UNPIVOT Sytax

```
"SELECT VendorID, Employee, Orders
FROM
(SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
FROM pvt) p
```

```
UNPIVOT
(Orders FOR Employee IN
(Emp1, Emp2, Emp3, Emp4, Emp5)
)AS unpvt;"
```

For further information on PIVOT and UNPIVOT, see FROM clause plus JOIN, APPLY, PIVOT (Transact-SQL)

CREATE TABLE Statements

To create new MongoDB entities, use CREATE TABLE statements.

CREATE TABLE Syntax

The CREATE TABLE statement specifies the table name and a comma-separated list of column names and the primary keys of the table, as shown in the following example:

```
CREATE TABLE <table_name> [ IF [ NOT EXISTS ] ]
(
  {
     <column_name> <data_type>
     [ NOT NULL ]
     [ DEFAULT <literal> ]
     [ PRIMARY KEY ]
     [ UNIQUE ]
  } |  PRIMARY KEY ( <column_name> [ , ... ] )
  [ , ... ]
)
```

The following example statement creates a MyCustomers table on the MongoDB server with name, age, and address columns:

```
CREATE TABLE IF NOT EXISTS "MyCustomers" (name VARCHAR(20), age INT,
address VARCHAR(20))
```

DROP TABLE Statements

Use DROP TABLE statements to delete a table and all the data it contains from MongoDB.

DROP TABLE Syntax

The DROP TABLE statement accepts the name of the table to delete, as shown in the following example:

```
DROP TABLE [ IF EXISTS ] <table_name>
```

The following query deletes all MyCustomers data from the server:

```
DROP TABLE IF EXISTS MyCustomers
```

Connection String Options

The connection string properties are the various options that can be used to establish a connection. This section provides a complete list of the options you can configure in the connection string for this provider. Click the links for further details. For more information on establishing a connection, see Basic Tab.

Authentication

Property	Description
AuthScheme	The authentication mechanism that MongoDB will use to authenticate the connection.
Server	The host name or IP address of the server hosting the MongoDB database.
Port	The port for the MongoDB database.
User	The MongoDB user account used to authenticate.
Password	The password used to authenticate the user.
Database	The name of the MongoDB database.
UseSSL	This field sets whether SSL is enabled.
AuthDatabase	The name of the MongoDB database for authentication.

Property	Description
ReplicaSet	This property allows you to specify multiple servers in addition to the one configured in Server and Port . Specify both a server name and port; separate servers with a comma.
DNSServer	Specify the DNS server when resolving MongoDB seed list.

Kerberos

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

SSL

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Property	Description
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

SSH

Property	Description
SSHAuthMode	The authentication method to be used to log on to an SFTP server.
SSHClientCert	A certificate to be used for authenticating the SSHUser.
SSHClientCertPassword	The password of the SSHClientCert key if it has one.
SSHClientCertSubject	The subject of the SSH client certificate.
SSHClientCertType	The type of SSHClientCert private key.
SSHServer	The SSH server.
SSHPort	The SSH port.
SSHUser	The SSH user.
SSHPassword	The SSH password.
SSHServerFingerprint	The SSH server fingerprint.
UseSSH	Whether to tunnel the MongoDB connection over SSH. Use SSH.

Firewall

Property	Description
FirewallType	The protocol used by a proxy-based firewall.
FirewallServer	The name or IP address of a proxy-based firewall.
FirewallPort	The TCP port for a proxy-based firewall.
FirewallUser	The user name to use to authenticate with a proxy-based firewall.
FirewallPassword	A password used to authenticate to a proxy-based firewall.

Logging

Property	Description
LogModules	Core modules to be included in the log file.

Schema

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.

Miscellaneous

Property	Description
BuiltInColumnMapping	A list of column name mapping for MongoDB's built-in columns.
DataModel	By default, the provider will not automatically discover the metadata for a child table as its own distinct table. To enable this functionality, set DataModel to Relational .
FlattenArrays	By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.
FlattenObjects	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaxRows	Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.
NoCursorTimeout	The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from MongoDB.
QueryPassthrough	This option passes the query to MongoDB as-is.
Readonly	You can use this property to enforce read-only access to MongoDB from the provider.
ReadPreference	Set this to a strategy for reading from a replica set. Accepted values are primary, primaryPreferred, secondary,

Property	Description
	secondaryPreferred, and nearest.
ReadPreferenceTags	Use this property to target a replica set member or members that are associated with tags.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
SlaveOK	This property sets whether the provider is allowed to read from secondary (slave) servers.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
TypeDetectionScheme	Comma-separated options for how the provider will scan the data to determine the fields and datatypes in each document collection.
UpdateScheme	Sets replacing or merging target document with updating fields is performed by executing update statement.
UseFindAPI	Execute MongoDB queries using db.collection.find().
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
WriteConcern	Requests acknowledgment that the write operation has propagated to the specified number of mongod instances.
WriteConcernJournaled	Requires acknowledgment that the mongod instances, as specified in the WriteConcern property, have written to the on-disk journal.
WriteConcernTimeout	This option specifies a time limit, in milliseconds, for the write concern.
WriteScheme	Sets whether the object type for inserted or updated objects is determined from the existing column metadata or the input value type.

Authentication

This section provides a complete list of the Authentication properties you can configure in the connection string for this provider.

Property	Description
AuthScheme	The authentication mechanism that MongoDB will use to authenticate the connection.
Server	The host name or IP address of the server hosting the MongoDB database.
Port	The port for the MongoDB database.
User	The MongoDB user account used to authenticate.
Password	The password used to authenticate the user.
Database	The name of the MongoDB database.
UseSSL	This field sets whether SSL is enabled.
AuthDatabase	The name of the MongoDB database for authentication.
ReplicaSet	This property allows you to specify multiple servers in addition to the one configured in Server and Port . Specify both a server name and port; separate servers with a comma.
DNSServer	Specify the DNS server when resolving MongoDB seed list.

AuthScheme

The authentication mechanism that MongoDB will use to authenticate the connection.

Possible Values

GSSAPI, MONGODB-CR, X509, SCRAM-SHA-1, NONE, SCRAM-SHA-256, PLAIN

Data Type

string

Default Value

"SCRAM-SHA-1"

Remarks

Accepted values are MONGODB-CR, SCRAM-SHA-1, SCRAM-SHA-256, GSSAPI, PLAIN, and NONE. The following authentication types correspond to the authentication values.

Authenticating with Challenge-Response

Generally, this property does not need to be set for this authentication type, as the adapter uses different challenge-response mechanisms by default to authenticate a user to different versions of MongoDB.

- **MongoDB 2**: MongoDB 2 uses MONGODB-CR to authenticate.
- **MongoDB 3.x**: MongoDB 3 uses SCRAM-SHA-1 by default; new users you create in MongoDB 3 use this authentication method. However, MongoDB 3 servers will continue to use MONGODB-CR to authenticate users created in MongoDB 2.6.
- **MongoDB 4.x**: MongoDB 4 uses SCRAM-SHA-1 by default and does not support the deprecated MongoDB MONGODB-CR authentication mechanism.

Authenticating with LDAP

Set AuthScheme to PLAIN to use LDAP authentication. This value specifies the SASL PLAIN mechanism; note that this mechanism transmits credentials over plain-text, so it is not suitable for use without TLS/SSL on untrusted networks.

Authenticating with Kerberos

Set AuthScheme to GSSAPI to use Kerberos authentication. Additionally configure the following properties as configured for the MongoDB environment:

KerberosKDC	The FQDN of the domain controller.
KerberosRealm	The Kerberos Realm (for Windows this will be the AD domain).
KerberosSPN	The assigned service principle name for the user.
AuthDatabase	This value should be set to '$external'.
User	The user created in the $external database.
Password	The corresponding User's password.

Authenticating with X.509 Authentication

Set AuthScheme to X509 to use X.509 certificate authentication.

Server

The host name or IP address of the server hosting the MongoDB database.

Data Type

string

Default Value

""

Remarks

The host name or IP address of the server hosting the MongoDB database. If you choose to connect using DNS seed lists, set this option to "mongodb+srv://" + the name of the server your MongoDB instance is running on..

Port

The port for the MongoDB database.

Data Type

string

Default Value

"27017"

Remarks

The port for the MongoDB database.

User

The MongoDB user account used to authenticate.

Data Type

string

Default Value

""

Remarks

Together with Password, this field is used to authenticate against the MongoDB server.

Password

The password used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The User and Password are together used to authenticate with the server.

Database

The name of the MongoDB database.

Data Type

string

Default Value

""

Remarks

The name of the MongoDB database.

UseSSL

This field sets whether SSL is enabled.

Data Type

bool

Default Value

false

Remarks

This field sets whether the adapter will attempt to negotiate TLS/SSL connections to the server. By default, the adapter checks the server's certificate against the system's trusted certificate store. To specify another certificate, set SSLServerCert.

AuthDatabase

The name of the MongoDB database for authentication.

Data Type

string

Default Value

""

Remarks

The name of the MongoDB database for authentication. Only needed if the authentication database is different from the database to retrieve data from.

ReplicaSet

This property allows you to specify multiple servers in addition to the one configured in Server and Port . Specify both a server name and port; separate servers with a comma.

Data Type

string

Default Value

""

Remarks

This property allows you to specify the other servers in the replica set in addition to the one configured in Server and Port. You must specify all servers in the replica set using ReplicaSet, Server, and Port.

Specify both a server name and port in ReplicaSet; separate servers with a comma. For example:

```
Server=localhost;Port=27017;ReplicaSet=localhost:27018,localhost:27019;
```

To find the primary server, the adapter queries the servers in ReplicaSet and the server specified by Server and Port.

Note that only the primary server in a replica set is writable. Secondaries can be readable if the SlaveOK setting allows it. To configure a strategy executing SELECT queries to secondaries, see ReadPreference.

DNSServer

Specify the DNS server when resolving MongoDB seed list.

Data Type

string

Default Value

""

Remarks

Specify the DNS server when resolving MongoDB seed list.

Kerberos

This section provides a complete list of the Kerberos properties you can configure in the connection string for this provider.

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

KerberosKDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The adapter will request session tickets and temporary session keys from the Kerberos KDC service. The Kerberos KDC service is conventionally colocated with the domain controller.

If Kerberos KDC is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the KDC from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: *C:\ProgramData\MIT\Kerberos5\krb5.ini* (Windows) or */etc/krb5.conf* (Linux).
- **Java System Properties:** Using the system properties *java.security.krb5.realm* and *java.security.krb5.kdc*.
- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the configured domain name and host.

Note: Windows authentication is supported in JRE 1.6 and above only.

KerberosRealm

The Kerberos Realm used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name.

If Kerberos Realm is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the default realm from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: *C:\ProgramData\MIT\Kerberos5\krb5.ini* (Windows) or */etc/krb5.conf* (Linux)
- **Java System Properties:** Using the system properties *java.security.krb5.realm* and *java.security.krb5.kdc*.
- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the user-configured domain name and host. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

KerberosSPN

The service principal name (SPN) for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the SPN on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, use this property to set the SPN.

KerberosKeytabFile

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

Data Type

string

Default Value

""

Remarks

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

KerberosServiceRealm

The Kerberos realm of the service.

Data Type

string

Default Value

""

Remarks

The KerberosServiceRealm is the specify the service Kerberos realm when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosServiceKDC

The Kerberos KDC of the service.

Data Type

string

Default Value

""

Remarks

The KerberosServiceKDC is used to specify the service Kerberos KDC when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosTicketCache

The full file path to an MIT Kerberos credential cache file.

Data Type

string

Default Value

""

Remarks

This property can be set if you wish to use a credential cache file that was created using the MIT Kerberos Ticket Manager or kinit command.

SSL

This section provides a complete list of the SSL properties you can configure in the connection string for this provider.

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

SSLClientCert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The SSLClientCertType field specifies the type of the certificate store specified by SSLClientCert. If the store is password protected, specify the password in SSLClientCertPassword.

SSLClientCert is used in conjunction with the SSLClientCertSubject field in order to specify client certificates. If SSLClientCert has a value, and SSLClientCertSubject is set, a search for a certificate is initiated. See SSLClientCertSubject for more information.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (for example, PKCS12 certificate store).

SSLClientCertType

The type of key store containing the TLS/SSL client certificate.

Possible Values

USER, MACHINE, PFXFILE, PFXBLOB, JKSFILE, JKSBLOB, PEMKEY_FILE, PEMKEY_BLOB, PUBLIC_KEY_FILE, PUBLIC_KEY_BLOB, SSHPUBLIC_KEY_FILE, SSHPUBLIC_KEY_BLOB, P7BFILE, PPKFILE, XMLFILE, XMLBLOB

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

USER - default	**For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.**
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.

USER - default	**For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.**
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSLClientCertPassword

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password to open the certificate store.

SSLClientCertSubject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property. If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State

Field	Meaning
C	Country
E	Email Address

If a field value contains a comma, it must be quoted.

SSLServerCert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected.

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv.....Qw == -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY-----

Description	Example
	MIGfMA0GCSq.....AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

If not specified, any certificate trusted by the machine is accepted.

Certificates are validated as trusted by the machine based on the System's trust store. The trust store used is the 'javax.net.ssl.trustStore' value specified for the system. If no value is specified for this property, Java's default trust store is used (for example, JAVA_HOME\lib\security\cacerts).

Use '*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

SSH

This section provides a complete list of the SSH properties you can configure in the connection string for this provider.

Property	Description
SSHAuthMode	The authentication method to be used to log on to an SFTP server.
SSHClientCert	A certificate to be used for authenticating the SSHUser.
SSHClientCertPassword	The password of the SSHClientCert key if it has one.
SSHClientCertSubject	The subject of the SSH client certificate.
SSHClientCertType	The type of SSHClientCert private key.

Property	Description
SSHServer	The SSH server.
SSHPort	The SSH port.
SSHUser	The SSH user.
SSHPassword	The SSH password.
SSHServerFingerprint	The SSH server fingerprint.
UseSSH	Whether to tunnel the MongoDB connection over SSH. Use SSH.

SSHAuthMode

The authentication method to be used to log on to an SFTP server.

Possible Values

None, Password, Public_Key

Data Type

string

Default Value

"Password"

Remarks

- None: No authentication will be performed. The current User value is ignored, and the connection will be logged in as anonymous.
- Password: The adapter will use the values of User and Password to authenticate the user.

- Public_Key: The adapter will use the values of User and SSHClientCert to authenticate the user. SSHClientCert must have a private key available for this authentication method to succeed.

SSHClientCert

A certificate to be used for authenticating the SSHUser.

Data Type

string

Default Value

""

Remarks

SSHClientCert must contain a valid private key in order to use public key authentication. A public key is optional, if one is not included then the adapter generates it from the private key. The adapter sends the public key to the server and the connection is allowed if the user has authorized the public key.

The SSHClientCertType field specifies the type of the key store specified by SSHClientCert. If the store is password protected, specify the password in SSHClientCertPassword.

Some types of key stores are containers which may include multiple keys. By default the adapter will select the first key in the store, but you can specify a specific key using SSHClientCertSubject.

SSHClientCertPassword

The password of the SSHClientCert key if it has one.

Data Type

string

Default Value

""

Remarks

This property is only used when authenticating to SFTP servers with SSHAuthMode set to PublicKey and SSHClientCert set to a private key.

SSHClientCertSubject

The subject of the SSH client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property.

If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For instance "CN=www.server.com, OU=test, C=US, E=support@cdata.com". Common fields and their meanings are displayed below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State
C	Country
E	Email Address

If a field value contains a comma it must be quoted.

SSHClientCertType

The type of SSHClientCert private key.

Possible Values

USER, MACHINE, PFXFILE, PFXBLOB, JKSFILE, JKSBLOB, PEMKEY_FILE, PEMKEY_BLOB, PPKFILE, PPKBLOB, XMLFILE, XMLBLOB

Data Type

string

Default Value

"PEMKEY_FILE"

Remarks

This property can take one of the following values:

Types	Description	Allowed Blob Values
MACHINE/USER	Not available on this platform.	Blob values are not supported.
JKSFILE/JKSBLOB	A Java keystore file. Must contain both a certificate and a private key. Only available in Java.	base64-only
PFXFILE/PFXBLOB	A PKCS12-format (.pfx) file. Must contain both a certificate and a private key.	base64-only
PEMKEY_FILE/PEMKEY_BLOB	A PEM-format file. Must contain an RSA, DSA, or OPENSSH private key. Can optionally contain a certificate matching the private key.	base64 or plain text. Newlines may be replaced with spaces when providing the blob as text.
PPKFILE/PPKBLOB	A PuTTY-format private key created using the *puttygen* tool.	base64-only
XMLFILE/XMLBLOB	An XML key in the format generated by the .NET RSA class: *RSA.ToXmlString(true)*.	base64 or plain text.

SSHServer

The SSH server.

Data Type

string

Default Value

""

Remarks

The SSH server.

SSHPort

The SSH port.

Data Type

string

Default Value

"22"

Remarks

The SSH port.

SSHUser

The SSH user.

Data Type

string

Default Value

""

Remarks

The SSH user.

SSHPassword

The SSH password.

Data Type

string

Default Value

""

Remarks

The SSH password.

SSHServerFingerprint

The SSH server fingerprint.

Data Type

string

Default Value

""

Remarks

The SSH server fingerprint.

UseSSH

Whether to tunnel the MongoDB connection over SSH. Use SSH.

Data Type

bool

Default Value

false

Remarks

By default the adapter will attempt to connect directly to MongoDB. When this option is enabled, the adapter will instead establish an SSH connection with the SSHServer and tunnel the connection to MongoDB through it.

Firewall

This section provides a complete list of the Firewall properties you can configure in the connection string for this provider.

Property	Description
FirewallType	The protocol used by a proxy-based firewall.
FirewallServer	The name or IP address of a proxy-based firewall.
FirewallPort	The TCP port for a proxy-based firewall.
FirewallUser	The user name to use to authenticate with a proxy-based firewall.
FirewallPassword	A password used to authenticate to a proxy-based firewall.

FirewallType

The protocol used by a proxy-based firewall.

Possible Values

NONE, TUNNEL, SOCKS4, SOCKS5

Data Type

string

Default Value

"NONE"

Remarks

This property specifies the protocol that the adapter will use to tunnel traffic through the FirewallServer proxy.

Type	Default Port	Description
TUNNEL	80	When this is set, the adapter opens a connection to MongoDB and traffic flows back and forth through the proxy.
SOCKS4	1080	When this is set, the adapter sends data through the SOCKS 4 proxy specified by FirewallServer and FirewallPort and passes the FirewallUser value to the proxy, which determines if the connection request should be granted.
SOCKS5	1080	When this is set, the adapter sends data through the SOCKS 5 proxy specified by FirewallServer and FirewallPort. If your proxy requires authentication, set FirewallUser and FirewallPassword to credentials the proxy recognizes.

FirewallServer

The name or IP address of a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

This property specifies the IP address, DNS name, or host name of a proxy allowing traversal of a firewall. The protocol is specified by FirewallType: Use FirewallServer with this property to connect through SOCKS or do tunneling.

FirewallPort

The TCP port for a proxy-based firewall.

Data Type

int

Default Value

0

Remarks

This specifies the TCP port for a proxy allowing traversal of a firewall. Use FirewallServer to specify the name or IP address. Specify the protocol with FirewallType.

FirewallUser

The user name to use to authenticate with a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

The FirewallUser and FirewallPassword properties are used to authenticate against the proxy specified in FirewallServer and FirewallPort, following the authentication method specified in FirewallType.

FirewallPassword

A password used to authenticate to a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

This property is passed to the proxy specified by FirewallServer and FirewallPort, following the authentication method specified by FirewallType.

Logging

This section provides a complete list of the Logging properties you can configure in the connection string for this provider.

Property	Description
LogModules	Core modules to be included in the log file.

LogModules

Core modules to be included in the log file.

Data Type

string

Default Value

""

Remarks

Only the modules specified (separated by ';') will be included in the log file. By default all modules are included.

See the Logging page for an overview.

Schema

This section provides a complete list of the Schema properties you can configure in the connection string for this provider.

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.

Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

Data Type

string

Default Value

"%APPDATA%\\CData\\MongoDB Data Provider\\Schema"

Remarks

The path to a directory which contains the schema files for the adapter (.rsd files for tables and views, .rsb files for stored procedures). The folder location can be a relative path from the location of the executable. The Location property is only needed if you want to customize definitions (for example, change a column name, ignore a column, and so on) or extend the data model with new tables, views, or stored procedures.

If left unspecified, the default location is "%APPDATA%\\CData\\MongoDB Data Provider\\Schema" with **%APPDATA%** being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable
Mac	~/Library/Application Support
Linux	~/.config

Miscellaneous

This section provides a complete list of the Miscellaneous properties you can configure in the connection string for this provider.

Property	Description
BuiltInColumnMapping	A list of column name mapping for MongoDB's built-in columns.
DataModel	By default, the provider will not automatically discover the metadata for a child table as its own distinct table. To enable this functionality, set DataModel to Relational .
FlattenArrays	By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.
FlattenObjects	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaxRows	Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.
NoCursorTimeout	The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from MongoDB.
QueryPassthrough	This option passes the query to MongoDB as-is.
Readonly	You can use this property to enforce read-only access to MongoDB from the provider.
ReadPreference	Set this to a strategy for reading from a replica set. Accepted

Property	Description
	values are primary, primaryPreferred, secondary, secondaryPreferred, and nearest.
ReadPreferenceTags	Use this property to target a replica set member or members that are associated with tags.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
SlaveOK	This property sets whether the provider is allowed to read from secondary (slave) servers.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
TypeDetectionScheme	Comma-separated options for how the provider will scan the data to determine the fields and datatypes in each document collection.
UpdateScheme	Sets replacing or merging target document with updating fields is performed by executing update statement.
UseFindAPI	Execute MongoDB queries using db.collection.find().
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
WriteConcern	Requests acknowledgment that the write operation has propagated to the specified number of mongod instances.
WriteConcernJournaled	Requires acknowledgment that the mongod instances, as specified in the WriteConcern property, have written to the on-disk journal.
WriteConcernTimeout	This option specifies a time limit, in milliseconds, for the write concern.
WriteScheme	Sets whether the object type for inserted or updated objects is

Property	Description
	determined from the existing column metadata or the input value type.

BuiltInColumnMapping

A list of column name mapping for MongoDB's built-in columns.

Data Type

string

Default Value

""

Remarks

This property takes a comma-separated list of MongoDB's column name mapping for built-in columns. The built-in columns is "_index" and "P_id" for now.

Example: _index=BuiltInIndex,P_id=Parent_Id

DataModel

By default, the provider will not automatically discover the metadata for a child table as its own distinct table. To enable this functionality, set DataModel to Relational .

Possible Values

DOCUMENT, RELATIONAL

Data Type

string

Default Value

"DOCUMENT"

Remarks

When setting DataModel to Relational, the discovery of child tables extends to root level elements and those found within top-level array elements.

FlattenArrays

By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.

Data Type

string

Default Value

""

Remarks

By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. This is only recommended for arrays that are expected to be short.

Set FlattenArrays to the number of elements you want to return from nested arrays. The specified elements are returned as columns. The zero-based index is concatenated to the column name. Other elements are ignored.

For example, you can return an arbitrary number of elements from an array of strings:

```
["FLOW-MATIC","LISP","COBOL"]
```

When FlattenArrays is set to 1, the preceding array is flattened into the following table:

Column Name	Column Value
languages.0	FLOW-MATIC

Setting FlattenArrays to -1 will flatten all the elements of nested arrays.

FlattenObjects

Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.

Data Type

bool

Default Value

true

Remarks

Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON. To generate the column name, the adapter concatenates the property name onto the object name with a dot.

For example, you can flatten the nested objects below at connection time:

```
[
     { "grade": "A", "score": 2 },
     { "grade": "A", "score": 6 },
     { "grade": "A", "score": 10 },
     { "grade": "A", "score": 9 },
     { "grade": "B", "score": 14 }
]
```

When FlattenObjects is set to true and FlattenArrays is set to 1, the preceding array is flattened into the following table:

Column Name	Column Value
grades.0.grade	A
grades.0.score	2

GenerateSchemaFiles

Indicates the user preference as to when schemas should be generated and saved.

Possible Values

Never, OnUse, OnStart, OnCreate

Data Type

string

Default Value

"Never"

Remarks

GenerateSchemaFiles enables you to save the table definitions identified by Automatic Schema Discovery. This property outputs schemas to .rsd files in the path specified by Location.

Available settings are the following:

- Never: A schema file will never be generated.
- OnUse: A schema file will be generated the first time a table is referenced, provided the schema file for the table does not already exist.
- OnStart: A schema file will be generated at connection time for any tables that do not currently have a schema file.
- OnCreate: A schema file will be generated by when running a CREATE TABLE SQL query.

Note that if you want to regenerate a file, you will first need to delete it.

Generate Schemas with SQL

When you set GenerateSchemaFiles to **OnUse**, the adapter generates schemas as you execute SELECT queries. Schemas are generated for each table referenced in the query.

When you set GenerateSchemaFiles to **OnCreate**, schemas are only generated when a CREATE TABLE query is executed.

Generate Schemas on Connection

Another way to use this property is to obtain schemas for every table in your database when you connect. To do so, set GenerateSchemaFiles to **OnStart** and connect.

Alternatives to Static Schemas

If your data structures are volatile, consider setting GenerateSchemaFiles to Never and using dynamic schemas. See Automatic Schema Discovery for more information about dynamic schemas.

Editing Schemas

Schema files have a simple format that makes them easy to modify. See Custom Schema Definitions for more information.

MaxRows

Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.

Data Type

int

Default Value

-1

Remarks

Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.

NoCursorTimeout

The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that.

Data Type

bool

Default Value

false

Remarks

The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that.

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

DefaultColumnSize	**Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000.**
ConvertDateTimeToGMT	Determines whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

Pagesize

The maximum number of results to return per page from MongoDB.

Data Type

int

Default Value

4096

Remarks

The Pagesize property affects the maximum number of results to return per page from MongoDB. Setting a higher value may result in better performance at the cost of additional memory allocated per page consumed.

QueryPassthrough

This option passes the query to MongoDB as-is.

Data Type

bool

Default Value

false

Remarks

When set to 'True', the specified query will be passed to MongoDB as-is. Currently only these shell commands are supported:

- **db.myCollection.find()** returns all fields for all records in the collection.
- **db.myCollection.find({ query })** returns all fields for all records in the collection matching the query.
- **db.myCollection.find({ query }, { projection })** returns the fields in the projection, for all records matching the query.
- All of the above forms accept a **.json()** suffix. This returns a single column containing the matching documents as JSON instead of individual fields.

Note that you can use the EVAL stored procedure to execute other JavaScript functions.

Readonly

You can use this property to enforce read-only access to MongoDB from the provider.

Data Type

bool

Default Value

false

Remarks

If this property is set to true, the adapter will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

ReadPreference

Set this to a strategy for reading from a replica set. Accepted values are primary, primaryPreferred, secondary, secondaryPreferred, and nearest.

Data Type

string

Default Value

"primary"

Remarks

This property enables you to execute queries to a member in a replica set other other than the primary member. Accepted values are the following:

- **primary**: All SELECT queries are executed against the primary server.
- **primaryPreferred**: If the primary server is not available, SELECT queries are executed to a secondary server.
- **secondary**: All SELECT queries are executed to the secondary servers.
- **secondaryPreferred**: SELECT queries are executed to a secondary server if one is available. Otherwise, the queries are executed to the primary server.
- **nearest**: SELECT queries are executed to the server with the least latency.

When to Use ReadPreference

When this property is set, query results may not reflect the latest changes if a write operation has not yet been replicated to a secondary machine. You can use ReadPreference to accomplish the following, with some risk that the adapter will return stale data:

- **Configure failover queries**: If the primary server is unavailable, you can set this property to "primaryPreferred" to continue to execute queries online.
- **Execute faster queries to geographically distributed replica sets**: If your deployment uses multiple data centers, setting ReadPreference to "nearest" can result in faster queries, as the adapter executes SELECT queries to whichever replica set member has the lowest latency.

When directing the adapter to execute SELECT statements to a secondary server, SlaveOK must also be set. Otherwise, the adapter will return an error response.

ReadPreferenceTags

Use this property to target a replica set member or members that are associated with tags.

Data Type

string

Default Value

""

Remarks

To make use of ReadPreferenceTags you must configure ReadPreference to a value other than the primary value (the default value). The required format is a list of semicolon seperated tag sets where each tag set is a list of key value pairs separated by commas. For example:

- **tag1:val1,tag2:val2;**: Find members with both tag values. If none are found, find any eligible member.
- **tag1:val1;tag2:val2;**: Find members with the specified tag1, otherwise find members with the specified tag2. If none are found find any eligible member.
- **tag1:val1**: Find only members with the specified tag.
- **;**: (semicolon only) Find any eligible member. If left empty, any eligible member is targeted.

RowScanDepth

The maximum number of rows to scan to look for the columns available in a table.

Data Type

int

Default Value

100

Remarks

The columns in a table must be determined by scanning table rows. This value determines the maximum number of rows that will be scanned.

Setting a high value may decrease performance. Setting a low value may prevent the data type from being determined properly, especially when there is null data.

Setting to a value of -1 causes the adapter to scan an arbitrary number of rows until it reaches the final row.

SlaveOK

This property sets whether the provider is allowed to read from secondary (slave) servers.

Data Type

bool

Default Value

false

Remarks

This property sets whether the adapter is allowed to read from secondary (slave) servers in a replica set. You can fine-tune how the adapter queries secondary servers with

ReadPreference.

Timeout

The value in seconds until the timeout error is thrown, canceling the operation.

Data Type

int

Default Value

60

Remarks

If Timeout = 0, operations do not time out. The operations run until they complete successfully or until they encounter an error condition.

If Timeout expires and the operation is not yet complete, the adapter throws an exception.

TypeDetectionScheme

Comma-separated options for how the provider will scan the data to determine the fields and datatypes in each document collection.

Data Type

string

Default Value

"RowScan"

Remarks

None	**Setting TypeDetectionScheme to None will return all columns as a string type. Cannot be combined with other options.**
RowScan	Setting TypeDetectionScheme to RowScan will scan rows to heuristically determine the data type. The RowScanDepth determines the number of rows to be scanned. Can be used with Recent.
Recent	Setting TypeDetectionScheme to 'RowScan,Recent' will instead execute the rowscan on the most recent documents inserted into the collection. This is a more expensive operation that may be significantly slower on large datasets.

UpdateScheme

Sets replacing or merging target document with updating fields is performed by executing update statement.

Possible Values

Default, Merge

Data Type

string

Default Value

"Default"

Remarks

Sets replacing or merging target document with updating fields is performed by executing update statement. When the default value *Default* is used, the adapter updates the target document by replacing the whole original document with new one. When the value is set to *Merge*, only the specific field in the target document will be updated.

For example, if you have a collection 'classySample' as below.

```
{
  "_id": "1",
  "message": {
    "component_items": [{"locked": true}],
    "id":1
  }
}
```

```
UPDATE [classySample] SET [message.component_items.0.locked] = false
WHERE [message.id] = 1
```

In the query above, the 'message' document will be replaced with new document constructed with SET clause, the collection after updating looks like

```
{
  "_id": "1",
  "message": {
    "component_items": [
      {
        "locked": false
      }
    ]
  }
}
```

But when using **Merge**, only the 'locked' field in 'component_items' will be updated, the collection becomes

```
{
    "_id": "1",
    "message": {
        "component_items": [
            {
                "locked": false
            }
        ],
        "id": 1
    }
}
```

UseFindAPI

Execute MongoDB queries using db.collection.find().

Data Type

string

Default Value

"True"

Remarks

Amazon DocumentDB doesn't support the legacy OP_QUERY interface, so this must be set to True to query DocumentDB clusters with db.collection.find() instead.

UserDefinedViews

A filepath pointing to the JSON configuration file containing your custom views.

Data Type

string

Default Value

""

Remarks

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the UserDefinedViews connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom SQL query for the view.

For example:

```
{
	"MyView": {
		"query": "SELECT * FROM [CData].[Sample].Customers WHERE MyColumn =
'value'"
	},
	"MyView2": {
		"query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
	}
}
```

Use the UserDefinedViews connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

WriteConcern

Requests acknowledgment that the write operation has propagated to the specified number of mongod instances.

Data Type

string

Default Value

"0"

Remarks

Requests acknowledgment that the write operation has propagated to the specified number of mongod instances.

WriteConcernJournaled

Requires acknowledgment that the mongod instances, as specified in the WriteConcern property, have written to the on-disk journal.

Data Type

bool

Default Value

true

Remarks

It requests acknowledgment that the mongod instances, as specified in the WriteConcern property, have written to the on-disk journal.

WriteConcernTimeout

This option specifies a time limit, in milliseconds, for the write concern.

Data Type

string

Default Value

"0"

Remarks

This option specifies a time limit, in milliseconds, for the write concern.

WriteScheme

Sets whether the object type for inserted or updated objects is determined from the existing column metadata or the input value type.

Possible Values

RawValue, Metadata

Data Type

string

Default Value

"Metadata"

Remarks

Sets whether the object type for inserted or updated objects is determined from the existing column metadata or the input value type. When the default value *Metadata* is used, the adapter uses the data type as determined by the *TypeDetectionScheme* for objects pushed to MongoDB. When the value is set to *RawValue*, the type of the object in the INSERT determines what type is used for MongoDB.

For example, if you have a field 'c1' in MongoDB defined as **String** type, the metadata returns the column as **String** as well. In the following query, the resulting field in MongoDB is therefore defined as **String** when using *WriteScheme=Metadata*. But when using **RawValue**, the inserting field type is **Date** instead since the *FROM_UNIXTIME()* function returns an actual **Date** object:

```
INSERT INTO Table1 (c1) VALUES (FROM_UNIXTIME(1636910867039, 0))
```

Inserting an empty array

With *WriteScheme=RawValue*, use the following syntax to insert an empty BSON array:

```
INSERT INTO t1 ("c1") VALUES (())
```

This returns an empty array:

```
"c1":[]
```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO® Data Virtualization page.

Users

- TDV Getting Started Guide
- TDV User Guide
- TDV Web UI User Guide
- TDV Client Interfaces Guide
- TDV Tutorial Guide
- TDV Northbay Example

Administration

- TDV Installation and Upgrade Guide
- TDV Administration Guide
- TDV Active Cluster Guide
- TDV Security Features Guide

Data Sources

- TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

References

TDV Reference Guide

TDV Application Programming Interface Guide

Other

TDV Business Directory Guide

TDV Discovery Guide

TDV and Business Directory Release Notes - Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Release Version Support

TDV 8.5 and 8.8 are designated as Long Term Support (LTS) versions. Some release versions of TIBCO® Data Virtualization products are selected to be long-term support (LTS) versions. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also Long Term Support.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the our product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, TIBCO logo, TIBCO O logo, ActiveSpaces, Enterprise Messaging Service, Spotfire, TERR, S-PLUS, and S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file

for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.tibco.com/patents.