



TIBCO® Data Virtualization

CosmosDB Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
Cosmos DB Adapter	4
Getting Started	4
Basic Tab	5
Logging	12
Creating a Custom AzureAD App	13
Using Azure Service Principal Authentication	16
Fine-Tuning Data Access	17
Setting a RU Budget for Batch Writes	18
Changelog	19
NoSQL Database	23
Automatic Schema Discovery	24
Free-Form Queries	26
Vertical Flattening	28
JSON Functions	30
SQL API Built-In Functions	32
SQL API GROUP BY	38
Query Mapping (Sql API)	38
Custom Schema Definitions	42
Custom Schema Example	43
Stored Procedures	44
AddDocument	45
CreateSchema	46
GetOAuthAccessToken	47
GetOAuthAuthorizationURL	48
RefreshOAuthAccessToken	49
SQL Compliance	50

SELECT Statements	51
SELECT INTO Statements	77
INSERT Statements	78
UPDATE Statements	79
DELETE Statements	79
EXECUTE Statements	80
PIVOT and UNPIVOT	81
Connection String Options	82
Authentication	87
OAuth	92
SSL	100
Firewall	106
Proxy	110
Logging	116
Schema	117
Miscellaneous	119
TIBCO Product Documentation and Support Services	134
How to Access TIBCO Documentation	134
How to Contact TIBCO Support	135
Release Version Support	135
How to Join TIBCO Community	136
Legal and Third-Party Notices	137

Cosmos DB Adapter

Cosmos DB Version Support

The adapter enables standards-based access to Azure Cosmos DB.

SQL Compliance

The [SQL Compliance](#) section shows the SQL syntax supported by the adapter and points out any limitations.

Getting Started

Connecting to Cosmos DB

[Basic Tab](#) shows how to authenticate to Cosmos DB 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 Cosmos DB Adapter

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

1. Unzip the `tdv.cosmosdb.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.cosmosdb/tdv.cosmosdb.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 CosmosDB
```

Basic Tab

Before You Connect

Role Assignment

Ensure that the Azure identity has the correct role assignment. The identity is the account that you log in to the browser during AzureAD authentication or the Application itself for AzureServicePrincipal authentication. Please visit the link below for more details:

[Configure role-based access control for your Azure Cosmos DB account with Azure AD](#)

You can either assign one of the built-in role definitions:

- CosmosDB Built-in Data Reader
- CosmosDB Built-in Data Contributor

or create your own custom role definitions. You must also set the scope of the role assignment, where "/" means that the identity has access to all the databases.

Connecting to Cosmos DB

Account Key

Log in to the Azure Portal, select **Azure Cosmos DB**, and select your account.

Set the following to authenticate:

- AccountEndpoint: The Cosmos DB account URL. Set this to the **URI** value found in the **Settings > Keys** blade of the Cosmos DB account.
- AccountKey: A master key token or a resource token for connecting to Cosmos DB. Set this to the **PRIMARY KEY** value found in the **Settings > Keys** blade of the Cosmos

DB account.

- TokenType: (optional). Set this to "master" (the default value) if you are using a Master Token, which is a full permissions token generated during account creation. Otherwise, set this property to "resource" if you are using a Resource Token, which is a custom permissions token generated when a database user is set up.

Azure AD

Azure AD is a connection type that leverages OAuth to authenticate. OAuth requires the authenticating user to interact with Cosmos DB using an internet browser. The adapter facilitates this in several ways as described below. Set your AuthScheme to **AzureAD**. All AzureAD flows assume that you have done so.

Desktop Applications

CData provides an embedded OAuth application that simplifies OAuth desktop Authentication. Alternatively, you can create a custom OAuth application. See [Creating a Custom AzureAD App](#) for information about creating custom applications and reasons for doing so.

For authentication, the only difference between the two methods is that you must set two additional connection properties when using custom OAuth applications.

After setting the following connection properties, you are ready to connect:

- InitiateOAuth: Set this to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.
- OAuthClientId: (custom applications only) Set this to the client Id in your application settings.
- OAuthClientSecret: (custom applications only) Set this to the client secret in your application settings.
- CallbackURL: Set this to the Redirect URL in your application settings.

When you connect the adapter opens the OAuth endpoint in your default browser. Log in and grant permissions to the application. The adapter then completes the OAuth process:

1. Extracts the access token from the callback URL and authenticates requests.

2. Obtains a new access token when the old one expires.
3. Saves OAuth values in OAuthSettingsLocation that persist across connections.

Headless Machines

To configure the driver to use OAuth with a user account on a headless machine, you need to authenticate on another device that has an internet browser.

1. Choose one of these two options:
 - Option 1: Obtain the OAuthVerifier value as described in "Obtain and Exchange a Verifier Code" below.
 - Option 2: Install the adapter on another machine and transfer the OAuth authentication values after you authenticate through the usual browser-based flow, as described in "Transfer OAuth Settings" below.
2. Then configure the adapter to automatically refresh the access token from the headless machine.

Option 1: Obtain and Exchange a Verifier Code

To obtain a verifier code, you must authenticate at the OAuth authorization URL.

Follow the steps below to authenticate from the machine with an internet browser and obtain the OAuthVerifier connection property.

1. Choose one of these options:
 - If you are using the Embedded OAuth Application click [Cosmos DB OAuth endpoint](#) to open the endpoint in your browser.
 - If you are using a custom OAuth application, create the Authorization URL by setting the following properties:
 - InitiateOAuth: Set to **OFF**.
 - OAuthClientId: Set to the client Id assigned when you registered your application.
 - OAuthClientSecret: Set to the client secret assigned when you registered

your application.

Then call the [GetOAuthAuthorizationURL](#) stored procedure with the appropriate CallbackURL. Open the URL returned by the stored procedure in a browser.

2. Log in and grant permissions to the adapter. You are then redirected to the callback URL, which contains the verifier code.
3. Save the value of the verifier code. Later you will set this in the [OAuthVerifier](#) connection property.

Next, you need to exchange the OAuth verifier code for OAuth refresh and access tokens. Set the following properties:

On the headless machine, set the following connection properties to obtain the OAuth authentication values:

- [InitiateOAuth](#): Set this to **REFRESH**.
- [OAuthVerifier](#): Set this to the verifier code.
- [OAuthClientId](#): (custom applications only) Set this to the client Id in your custom OAuth application settings.
- [OAuthClientSecret](#): (custom applications only) Set this to the client secret in the custom OAuth application settings.
- [OAuthSettingsLocation](#): Set this to the path to the file where the driver saves the OAuth token values that persist across connections.

After the OAuth settings file is generated, you need to re-set the following properties to connect:

- [InitiateOAuth](#): Set this to **REFRESH**.
- [OAuthClientId](#): (custom applications only) Set this to the client Id assigned when you registered your application.
- [OAuthClientSecret](#): (custom applications only) Set this to the client secret assigned when you registered your application.
- [OAuthSettingsLocation](#): Set this to the file containing the encrypted OAuth authentication values. Make sure this file grants read and write permissions to the adapter to enable the automatic refreshing of the access token.

Option 2: Transfer OAuth Settings

Prior to connecting on a headless machine, you need to create and install a connection with the driver on a device that supports an internet browser. Set the connection properties as described in "Desktop Applications" above.

After completing the instructions in "Desktop Applications", the resulting authentication values are encrypted and written to the path specified by OAuthSettingsLocation. The default filename is *OAuthSettings.txt*.

Once you have successfully tested the connection, copy the OAuth settings file to your headless machine.

On the headless machine, set the following connection properties to connect to data:

- InitiateOAuth: Set this to **REFRESH**.
- OAuthClientId: (custom applications only) Set this to the client Id assigned when you registered your application.
- OAuthClientSecret: (custom applications only) Set this to the client secret assigned when you registered your application.
- OAuthSettingsLocation: Set this to the path to your OAuth settings file. Make sure this file gives read and write permissions to the adapter to enable the automatic refreshing of the access token.

Client Credentials

Client credentials refers to a flow in OAuth where there is no direct user authentication taking place. Instead, credentials are created for just the app itself. All tasks taken by the app are done without a default user context. This makes the authentication flow a bit different from standard.

Client OAuth Flow

All permissions related to the client oauth flow require admin consent. This means the app embedded with the Cosmos DB Adapter cannot be used in the client oauth flow. You must create your own OAuth app in order to use client credentials. See [Creating a Custom AzureAD App](#) for more details.

In your App Registration in *portal.azure.com*, navigate to API Permissions and select the **Microsoft Graph permissions**. There are two distinct sets of permissions - Delegated and Application permissions. The permissions used during client credential authentication are under Application Permissions. Select the permissions you require for your integration.

You are ready to connect after setting one of the connection properties groups depending on the authentication type.

1. Authenticating using a Client Secret

- InitiateOAuth: Set this to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.
- AzureTenant: Set this to the tenant you wish to connect to.
- OAuthGrantType: Set this to **CLIENT**.
- OAuthClientId: Set this to the client Id in your app settings.
- OAuthClientSecret: Set this to the client secret in your app settings.

2. Authenticating using a Certificate

- InitiateOAuth: Set this to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.
- AzureTenant: Set this to the tenant you wish to connect to.
- OAuthGrantType: Set this to **CLIENT**.
- OAuthClientId: Set this to the client Id in your app settings.
- OAuthJWTCert: Set this to the JWT Certificate store.
- OAuthJWTCertType: Set this to the type of the certificate store specified by OAuthJWTCert.

Authentication with client credentials takes place automatically like any other connection, except there is no window opened prompting the user. Because there is no user context, there is no need for a browser popup. Connections will take place and be handled internally.

Azure Service Principal

Azure Service Principal is a connection type that goes through OAuth. Set your AuthScheme to **AzureServicePrincipal**. The authentication as an Azure Service Principal is handled via the OAuth Client Credentials flow, and it does not involve direct user authentication. Instead, credentials are created for just the app itself. All tasks taken by the app are done without a default user context, but based on the assigned roles. The application access to the resources is controlled through the assigned roles' permissions.

Note: You must create a custom application prior to assigning a role. See [Creating a Custom AzureAD App](#) for more information.

When authenticating using an Azure Service Principal, you must register an application with an Azure AD tenant. Follow the steps below to create a new service principal that can be used with the role-based access control.

Assign a role to the application

To access resources in your subscription, you must assign a role to the application.

1. Open the **Subscriptions** page by searching and selecting the Subscriptions service from the search bar.
2. Select the particular subscription to assign the application to.
3. Open the **Access control (IAM)** and select **Add > Add role assignment** to open the **Add role assignment** page.
4. Select **Owner** as the role to assign to your created Azure AD app.

Complete the Authentication

You are ready to connect after setting one of the below connection properties groups, depending on the configured app authentication (client secret or certificate).

In both methods

Before choosing client secret or certificate authentication, follow these steps then continue to the relevant section below:

1. AuthScheme: Set this to the **AzureServicePrincipal** in your app settings.
2. InitiateOAuth: Set this to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.
3. AzureTenant: Set this to the tenant you wish to connect to.
4. OAuthClientId: Set this to the client Id in your app settings.

Authenticating using a Client Secret

Continue with the following:

1. OAuthClientId: Set this to the client Id in your app settings.
2. OAuthClientSecret: Set this to the client secret in your app settings.

Authenticating using a Certificate

Continue with the following:

1. OAuthJWTCert: Set this to the JWT Certificate store.
2. OAuthJWTCertType: Set this to the type of the certificate store specified by OAuthJWTCert.

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 Cosmos DB 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**.

Creating a Custom AzureAD App

There are two types of custom AzureAD applications: AzureAD and AzureAD with an Azure Service Principal. Both are OAuth-based.

When to Create a Custom Application

CData embeds OAuth Application Credentials with CData branding that can be used when connecting via either a Desktop Application or from a Headless Machine.

You may choose to use your own AzureAD Application Credentials when you want to

- control branding of the Authentication Dialog
- control the redirect URI that the application redirects the user to after the user authenticates
- customize the permissions that you are requesting from the user

Custom AzureAD Applications

You can use a custom AzureAD application to authenticate a service account or a user account. You can always create a custom AzureAD application, but note that desktop and headless connections support embedded OAuth, which simplifies the process of authentication. See "Establishing a Connection" for information about using the embedded OAuth application.

Create a Custom AzureAD App

Follow the steps below to obtain the AzureAD values for your application, the OAuthClientId and OAuthClientSecret.

1. Log in to <https://portal.azure.com>.
2. In the left-hand navigation pane, select **Azure Active Directory**, then **applicationRegistrations**, and click **New registration**.
3. Enter an application name and select the desired tenant setup. When creating a custom AzureAD application in Azure Active Directory, you can define whether the application is single- or multi-tenant. If you select the default option, "Accounts in this organizational directory only", you must set the AzureTenant connection property to the Id of the Azure AD Tenant when establishing a connection with the Cosmos DB Adapter. Otherwise, the authentication attempt fails with an error. If your application is for private use only, "Accounts in this organization directory only"

should be sufficient. Otherwise, if you want to distribute your application, choose one of the multi-tenant options.

4. Set the redirect url to *http://localhost:33333*, the adapter's default. Or, specify a different port and set CallbackURL to the exact reply URL you defined.
5. Click **Register** to register the new application. This opens an application management screen. Note the value in **Application (client) ID** as the OAuthClientId and the **Directory (tenant) ID** as the AzureTenant.
6. Navigate to the "Certificates & Secrets" and define the application authentication type. There are two types of authentication available: using a client secret or a certificate. The recommended authentication method is using a certificate.
 - Option 1: Upload a certificate: In "Certificates & Secrets", select **Upload certificate** and the certificate to upload from your local machine.
 - Option 2: Create a new application secret: In "Certificates & Secrets", select **New Client Secret** for the application and specify its duration. After saving the client secret, the key value is displayed. *Copy this value as it is displayed only once.* You will need it as the OAuthClientSecret.
7. Select **API Permissions > Add**. If you plan for your application to connect without a user context, select **Application Permissions** (OAuthGrantType = **CLIENT**). Otherwise, use the **Delegated permissions**.
8. Save your changes.
9. If you have selected to use permissions that require admin consent (such as the Application Permissions), you can grant them from the current tenant on the API Permissions page.

Custom AzureAD Service Principal Applications

When authenticating using an Azure Service Principal, you must create both a custom AzureAD application and a service principal that can access the necessary resources. Follow the steps below to create a custom AzureAD application and obtain the connection properties for Azure Service Principal authentication.

Create a Custom AzureAD App with an Azure Service Principal

Follow the steps below to obtain the AzureAD values for your application.

1. Log in to <https://portal.azure.com>.
2. In the left-hand navigation pane, select Azure Active Directory then App Registrations and click **New registration**.
3. Enter an app name and select **Any Azure AD Directory - Multi Tenant**. Then set the redirect url to <http://localhost:33333>, the adapter's default.
4. After creating the application, copy the Application (client) Id value displayed in the "Overview" section. This value is used as the OAuthClientId
5. Define the app authentication type by going to the "Certificates & Secrets" section. There are two types of authentication available: using a client secret and using a certificate. The recommended authentication method is via a certificate.
 - Option 1 - Upload a certificate: In "Certificates & Secrets", select **Upload certificate** and the certificate to upload from your local machine.
 - Option 2 - Create a new application secret: In "Certificates & Secrets", select **New Client Secret** for the application and specify its duration. After saving the client secret, the key value is displayed. *Copy this value as it is displayed only once.* You will use it as the OAuthClientSecret.
6. On the **Authentication** tab, make sure to select **Access tokens (used for implicit flows)**.

Using Azure Service Principal Authentication

The authentication as an Azure Service Principal is handled via the OAuth Client Credentials flow, and it does not involve direct user authentication. Instead, credentials are created for just the app itself. All tasks taken by the app are done without a default user context, but based on the assigned roles. The application access to the resources is controlled through the assigned roles' permissions.

Create an AzureAD App and an Azure Service Principal

When authenticating using an Azure Service Principal, you must register an application with an Azure AD tenant.

Assign a role to the application

To access resources in your subscription, you must assign a role to the application.

1. Open the **Subscriptions** page by searching and selecting the Subscriptions service from the search bar.
2. Select the particular subscription to assign the application to.
3. Open the **Access control (IAM)** and select **Add > Add role assignment** to open the **Add role assignment** page.
4. Select **Owner** as the role to assign to your created Azure AD app.

Complete the Authentication

You are ready to connect after setting one of the below connection properties groups, depending on the configured app authentication (client secret or certificate).

In both methods

Before choosing client secret or certificate authentication, follow these steps then continue to the relevant section below:

1. AuthScheme: Set this to the **AzureServicePrincipal** in your app settings.
2. InitiateOAuth: Set this to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.
3. AzureTenant: Set this to the tenant you wish to connect to.
4. OAuthClientId: Set this to the client Id in your app settings.

Authenticating using a Client Secret

Continue with the following:

1. OAuthClientId: Set this to the client Id in your app settings.
2. OAuthClientSecret: Set this to the client secret in your app settings.

Authenticating using a Certificate

Continue with the following:

1. OAuthJWTCert: Set this to the JWT Certificate store.
2. OAuthJWTCertType: Set this to the type of the certificate store specified by OAuthJWTCert.

Fine-Tuning Data Access

Fine Tuning Data Access

You can use the following properties to gain greater control over Cosmos DB API features and the strategies the adapter uses to surface them:

- RowScanDepth: This property determines the number of rows that will be scanned to detect column data types when generating table metadata.
- TypeDetectionScheme: This property allows more control over the strategy implemented by the RowScanDepth property.
- GenerateSchemaFiles: This property enables you to persist table metadata in static schema files that are easy to customize, to persist your changes to column data types, for example.

You can set this property to "OnStart" to generate schema files for all tables in your database at connection. Or, you can generate schemas as you execute SELECT queries to tables.

The resulting schemas are based on the connection properties you use to configure [Automatic Schema Discovery](#)

To use the resulting schema files, set the Location property to the folder containing the schemas.

Setting a RU Budget for Batch Writes

Just as described in the [SQL Compliance](#) the adapter supports batch CUD (Create, Update, Delete) operations. Batch processing is achieved by issuing multiple requests simultaneously. Even though this method greatly improves the performance for write operations, the cost of these operations is relatively high, thus the Request Units (RU) budget per second for a certain container or database may be exceeded. Depending on your Cosmos DB Service Quotas, exceeding the RU budgets may incur in extra costs, or it may even temporary throttle or interrupt the Cosmos DB usage for other workloads.

In order to avoid exceeding the RU budget per second, the adapter dynamically adjusts the number of concurrent requests per second depending on the set WriteThroughputBudget and the constantly adjusted average RU cost per statement. The user can utilize the WriteThroughputBudget connection property to define the RU budget per second, that batch write operations should not exceed. Another important factor in batch write operations is the MaxThreads connection property, which specifies the maximum number of concurrent requests. If using a low MaxThreads value, the adapter might not be able to efficiently use the available budget.

Since the requests throttling logic is applied client-side, in a few cases the RU/s budget may be exceeded by a relatively small amount. These cases include Inserting, Updating and Deleting records with highly variable column count and input value length per column.

Note: By default, the WriteThroughputBudget property is set 1000 RU/s and the MaxThreads property is set to 200 threads.

Changelog

General Changes

Date	Build Number	Change Type	Description
12/14/2022	8383	General	Changed <ul style="list-style-type: none"> Added the Default column to the sys_procedureparameters table.
09/30/2022	8308	General	Changed <ul style="list-style-type: none"> Added the IsPath column to the sys_procedureparameters table.
09/23/2022	8301	Cosmos DB	Added <ul style="list-style-type: none"> Added the FileStream input for the CreateSchema stored procedure. It streams the contents of the created schema if no FileName is set.
08/19/2022	8266	Cosmos DB	Added <ul style="list-style-type: none"> Added the FileData output for the CreateSchema stored procedure. It pushes the generated schema encoded in Base64 only if the FileName input is not set.
08/17/2022	8264	General	Changed

			<ul style="list-style-type: none"> We now support handling the keyword "COLLATE" as standard function name as well.
07/28/2022	8179	Cosmos DB	Changed <ul style="list-style-type: none"> Changed provider name to Azure Cosmos DB.
01/14/2022	8049	Cosmos DB	Added <ul style="list-style-type: none"> Added support for the AzureAD and AzureServicePrincipal authentication methods.
12/14/2021	8018	Cosmos DB	Added <ul style="list-style-type: none"> Added the new connection property UseRidAsPK. Since CosmosDB allows you to use both _rid and id fields as unique values for retrieving resource data, you can set this property to false to switch using the id column as primary key instead the default _rid.
12/07/2021	8011	Cosmos DB	Added <ul style="list-style-type: none"> Added support for the 'RawValue' TypeDetectionScheme. Setting the TypeDetectionScheme to 'RawValue' will push each document as a single aggregate on a column named JsonData, along with its resource identifier on the separate Primary Key column.
10/10/2021	7953	Cosmos DB	Added <ul style="list-style-type: none"> Added support for Batch Insert, Update and Delete operations. Added support to dynamically control the number of parallel batch CUD requests to prevent exceeding the limit Max Requests

			Units Per Second, which can be configured via the MaxWriteThroughput connection property.
09/02/2021	7915	General	Added <ul style="list-style-type: none"> Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/07/2021	7889	General	Changed <ul style="list-style-type: none"> Added the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	Changed <ul style="list-style-type: none"> Added the new sys_primarykeys system table.
07/23/2021	7874	General	Changed <ul style="list-style-type: none"> Updated the Literal Function Names for relative date/datetime functions. Previously relative date/datetime functions resolved to a different value when used in the projection vs the predicate. I.e: SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col > LAST_MONTH(). Formerly the two LAST_MONTH() methods would resolve to different datetimes. Now they will match. As a replacement for the previous behavior, the relative date/datetime functions in the criteria may have an 'L' appended to them. I.e: WHERE col > L_LAST_MONTH(). This will continue to resolve to the same values that previously were calculated in the criteria. Note that the "L_" prefix will only work in the predicate - it not available for the projection.

07/08/2021	7859	General	Added <ul style="list-style-type: none"> 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 verbosity=5.
06/14/2021	7835	Cosmos DB	Added <ul style="list-style-type: none"> Added the SetPartitionKeyAsPK connection property, which controls whether or not to use the collection's Partition Key field as part of a composite Primary Key for the corresponding exposed table.
04/23/2021	7785	General	Added <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2 Changed <ul style="list-style-type: none"> Reduced the length to 255 for varchar primary key and foreign key columns. Updated implicit and metadata caching to improve performance and support for

			<p>multiple connections. Old metadata caches are not compatible - you would need to generate new metadata caches if you are currently using CacheMetadata.</p> <ul style="list-style-type: none"> • Updated index naming convention to avoid duplicates • Updated and standardized Getting Started connection help. • Added the Advanced Features section to the help of all drivers. • Categorized connection property listings in the help for all editions.
04/15 /2021	7775	General	<p>Changed</p> <ul style="list-style-type: none"> • Kerberos authentication is updated to use TCP by default, but will fall back to UDP if a TCP connection cannot be established

NoSQL Database

Cosmos DB 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 Cosmos DB Objects as Tables

The adapter models the schemaless Cosmos DB objects into relational tables and translates SQL queries into Cosmos DB queries to get the requested data. See [Query Mapping \(Sql API\)](#) for more details on how various Cosmos DB operations are represented as SQL.

Discovering Schemas Automatically

The [Automatic Schema Discovery](#) scheme automatically finds the data types in a Cosmos DB 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 Cosmos DB. 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 Cosmos DB 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.

Limitations of the RawValue TypeDetectionScheme

If the [TypeDetectionScheme](#) is set to **RawValue**, the adapter will push each document as single aggregate value on a column named JsonData, along with its resource identifier on the separate Primary Key column. The JSON documents are not processed, and as a result, the below functionalities are NOT supported with this configuration.

- [Automatic Schema Discovery](#)
- [Free-Form Queries](#)
- [Vertical Flattening](#)
- [SQL API Built-In Functions](#)
- [SQL API GROUP BY](#)
- Almost all server side supported filters apart from WHERE clause conditions built with the resource identifier.

Automatic Schema Discovery

The adapter automatically infers a relational schema by inspecting a series of Cosmos DB 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.

You can change the separator character in the column name from a dot by setting [SeparatorCharacter](#).

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 Cosmos DB data. [JSON Functions](#) enable you to use standard JSON functions to summarize Cosmos DB 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
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 Cosmos DB documents -- this type of query is supported through the Cosmos DB 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

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 }
```

SQL API Built-In Functions

Cosmos DB also supports a number of built-in functions for common operations, that can be used inside queries. Here are some example of how can be used as part of select columns or the WHERE clause:

Use Built-in functions as part of SELECT columns

```
SELECT IS_NUMBER(user_id) AS ISN_ATTR, IS_NUMBER(id) AS ISN_ID FROM [users]
SELECT POWER(user_id, 2) AS POWERSSS, LENGTH(id) AS LENGTH_ID, PI() AS JustThePI FROM [users]
```

Use Built-in functions as part of WHERE clause

```
SELECT * FROM [users] WHERE STARTSWITH(middle_name, 'G')
SELECT * FROM [users] WHERE REPLACE(middle_name, 'Chr', '___') = '___istopher'
```


Function group	Operations
Mathematical functions	ABS, CEILING, EXP, FLOOR, LOG, LOG10, POWER, ROUND, SIGN, SQRT, SQUARE, TRUNC, ACOS, ASIN, ATAN, ATN2, COS, COT, DEGREES, PI, RADIANS, SIN, and TAN
Type checking functions	IS_ARRAY, IS_BOOL, IS_NULL, IS_NUMBER, IS_OBJECT, IS_STRING, IS_DEFINED, and IS_PRIMITIVE
String functions	CONCAT, CONTAINS, ENDSWITH, INDEX_OF, LEFT, LENGTH, LOWER, LTRIM, REPLACE, REPLICATE, REVERSE, RIGHT, RTRIM, STARTSWITH, SUBSTRING, and UPPER
Array functions	ARRAY_CONCAT, ARRAY_CONTAINS, ARRAY_LENGTH, and ARRAY_SLICE

Mathematical functions

The mathematical functions each perform a calculation, based on input values that are provided as arguments, and return a numeric value. Here's a table of supported built-in mathematical functions.

Usage	Description
ABS (num_expr)	Returns the absolute (positive) value of the specified numeric expression.
CEILING (num_expr)	Returns the smallest integer value greater than, or equal to, the specified numeric expression.
FLOOR (num_expr)	Returns the largest integer less than or equal to the specified numeric expression.
EXP (num_expr)	Returns the exponent of the specified numeric expression.
LOG (num_expr [,base])	Returns the natural logarithm of the specified numeric expression, or the logarithm using the specified base

LOG10 (num_expr)	Returns the base-10 logarithmic value of the specified numeric expression.
ROUND (num_expr)	Returns a numeric value, rounded to the closest integer value.
TRUNC (num_expr)	Returns a numeric value, truncated to the closest integer value.
SQRT (num_expr)	Returns the square root of the specified numeric expression.
SQUARE (num_expr)	Returns the square of the specified numeric expression.
POWER (num_expr, num_expr)	Returns the power of the specified numeric expression to the value specified.
SIGN (num_expr)	Returns the sign value (-1, 0, 1) of the specified numeric expression.
ACOS (num_expr)	Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.
ASIN (num_expr)	Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.
ATAN (num_expr)	Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.
ATN2 (num_expr)	Returns the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x), where x and y are the values of the two specified float expressions.
COS (num_expr)	Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.
COT (num_expr)	Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.

DEGREES (num_expr)	Returns the corresponding angle in degrees for an angle specified in radians.
PI ()	Returns the constant value of PI.
RADIANS (num_expr)	Returns radians when a numeric expression, in degrees, is entered.
SIN (num_expr)	Returns the trigonometric sine of the specified angle, in radians, in the specified expression.
TAN (num_expr)	Returns the tangent of the input expression, in the specified expression.

Type checking functions

The type checking functions allow you to check the type of an expression within SQL queries. Type checking functions can be used to determine the type of properties within documents dynamically when it is variable or unknown. Here's a table of supported built-in type checking functions.

Usage	Description
IS_ARRAY (expr)	Returns a Boolean indicating if the type of the value is an array.
IS_BOOL (expr)	Returns a Boolean indicating if the type of the value is a Boolean.
IS_NULL (expr)	Returns a Boolean indicating if the type of the value is null.
IS_NUMBER (expr)	Returns a Boolean indicating if the type of the value is a number.
IS_OBJECT (expr)	Returns a Boolean indicating if the type of the value is a JSON object.
IS_STRING (expr)	Returns a Boolean indicating if the type of the value is a string.
IS_DEFINED (expr)	Returns a Boolean indicating if the property has been assigned a value.

IS_PRIMITIVE (expr)	Returns a Boolean indicating if the type of the value is a string, number, Boolean or null.
---------------------	---

String functions

The following scalar functions perform an operation on a string input value and return a string, numeric or Boolean value. Here's a table of built-in string functions:

Usage	Description
LENGTH (str_expr)	Returns the number of characters of the specified string expression
CONCAT (str_expr, str_expr [, str_expr])	Returns a string that is the result of concatenating two or more string values.
SUBSTRING (str_expr, num_expr, num_expr)	Returns part of a string expression.
STARTSWITH (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression starts with the second
ENDSWITH (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression ends with the second
CONTAINS (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression contains the second.
INDEX_OF (str_expr, str_expr)	Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.
LEFT (str_expr, num_expr)	Returns the left part of a string with the specified number of characters.
RIGHT (str_expr, num_expr)	Returns the right part of a string with the specified number of characters.

LTRIM (str_expr)	Returns a string expression after it removes leading blanks.
RTRIM (str_expr)	Returns a string expression after truncating all trailing blanks.
LOWER (str_expr)	Returns a string expression after converting uppercase character data to lowercase.
UPPER (str_expr)	Returns a string expression after converting lowercase character data to uppercase.
REPLACE (str_expr, str_expr, str_expr)	Replaces all occurrences of a specified string value with another string value.
REPLICATE (str_expr, num_expr)	Repeats a string value a specified number of times.
REVERSE (str_expr)	Returns the reverse order of a string value.

Array functions

The following scalar functions perform an operation on an array input value and return numeric, Boolean or array value. Here's a table of built-in array functions:

Usage	Description
ARRAY_LENGTH (arr_expr)	Returns the number of elements of the specified array expression.
ARRAY_CONCAT (arr_expr, arr_expr [, arr_expr])	Returns an array that is the result of concatenating two or more array values.
ARRAY_CONTAINS (arr_expr, expr [, bool_expr])	Returns a Boolean indicating whether the array contains the specified value. Can specify if the match is full or partial.
ARRAY_SLICE (arr_expr, num_expr [, num_expr])	Returns part of an array expression.

Nested functions

You can also perform nested built-in functions, which will be processed server side as well:

```
i.e. SELECT TOP 10 CONCAT(SUBSTRING(UPPER(cuisine), 0, 3), '-cuisine')
FROM [restaurants]
```

SQL API GROUP BY

The GROUP BY clause divides the query's results according to the values of one or more specified properties. This operation is partially done server-side because of some API limitations. We still need to operate a client-side grouping.

GROUP BY Examples

```
SELECT COUNT(*) AS CNT, gender FROM [users] GROUP BY gender
SELECT COUNT(*) AS CNT, gender, doc_type FROM [users] GROUP BY gender,
doc_type
```

Query Mapping (Sql API)

The adapter maps SQL queries into the corresponding Cosmos DB SQL API 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 SQL API features such as the aggregation framework to compute the desired results.

SELECT Queries

Since all requests can be submitted to a specific collection, we can send any constant string as table name to the API. Following the Azure Portal standard we are using the "C" character as table name.

SQL Query	Sql API Query

```
SELECT id, name FROM Users
```

```
SELECT  
C.id,  
C.name  
FROM C
```

```
SELECT * FROM Users WHERE name = 'A'
```

```
SELECT *  
FROM C  
WHERE  
C.name =  
'A'
```

```
SELECT * FROM Users WHERE name = 'A' OR email =  
'zoe55@gmail.com'
```

```
SELECT *  
FROM C  
WHERE  
C.name =  
'A' OR  
C.email =  
'zoe55@gma  
il.com'
```

```
SELECT id, grantamt FROM WorldBank WHERE grantamt IN  
(4500000, 85400000) OR grantamt = 16200000
```

```
SELECT  
C.id,  
C.grantamt  
FROM C  
WHERE  
C.grantamt  
IN  
(4500000,  
85400000)  
OR  
C.grantamt  
= 16200000
```

```
SELECT * FROM WorldBank WHERE CountryCode = 'A' ORDER BY
TotalCommAmt ASC
```

```
SELECT *
FROM C
WHERE
C.countryc
ode = 'AL'
ORDER BY
C.totalcom
mamt ASC
```

```
SELECT * FROM WorldBank WHERE CountryCode = 'A' ORDER BY
TotalCommAmt DESC
```

```
SELECT *
FROM C
WHERE
C.countryc
ode = 'AL'
ORDER BY
C.totalcom
mamt DESC
```

Aggregate Queries

The adapter makes extensive use of this for various aggregate queries. See some examples below:

SQL Query

Sql API Query

```
SELECT COUNT(grantamt) AS COUNT_GRAMT FROM
WorldBank
```

```
SELECT COUNT
(C.grantamt) AS
COUNT_GRAMT FROM C
```

```
SELECT SUM(grantamt) AS SUM_GRAMT FROM WorldBank
```

```
SELECT SUM
```

```
(C.grantamt) AS
SUM_GRAMT FROM C
```

Built-In functions

SQL Query

```
SELECT IS_NUMBER(grantamt) AS ISN_ATTR, IS_NUMBER(id) AS
ISN_ID FROM WorldBank
```

Sql API Query

```
SELECT
IS_
NUMBER
(C.gran
tamt) AS
ISN_
ATTR,
IS_
NUMBER
(C.id)
AS ISN_
ID FROM
C
```

```
SELECT POWER(totalamt, 2) AS POWERS_A, LENGTH(id) AS LENGTH_
ID, PI() AS ThePI FROM WorldBank
```

```
SELECT
POWER
(C.tota
lamt, 2)
AS
POWERS_
A,
LENGTH
(C.id)
AS
LENGTH_
ID, PI()
AS ThePI
FROM C
```

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 Cosmos DB.

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"
}
```

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.

Below are the corresponding column definitions for the restaurants data set. 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
restaurants data set.">
    <!-- Column definitions -->
    <attr name="_rid"                xs:type="string"    key="true"
other:collrid="hWdRAKRi3Pg=" other:dbrid="hWdRAA=="
other:partitionpath="/name" />
    <attr name="borough"            xs:type="string"    />
    <attr name="cuisine"             xs:type="string"    />
    <attr name="address.building"    xs:type="string"    />
    <attr name="address.street"      xs:type="string"    />
    <attr name="address.coord.0"     xs:type="double"    />
    <attr name="address.coord.1"     xs:type="double"    />
    <input name="rows@next" desc="Internal attribute used for paging
through data." />
  </rsb:info>
  <rsb:set attr="collection" value="restaurants"/>
</rsb:script>
```

Custom Schema Example

This section contains a complete schema. The *info* section enables a relational view of a Cosmos DB object. For more details, see [Custom Schema Definitions](#). The table below allows the SELECT, INSERT, UPDATE, and DELETE commands as implemented in the GET, POST, MERGE, and DELETE sections of the schema below.

Copy the *rows@next* input as-is into your schema. The operations, such as *cosmosdbadoSysData*, are internal implementations and can also be copied as is.

Set the Location property to the file directory that will contain the schema file.

When, creating custom schemas, the attr for **_rid**, shown below, is required.

Also required are three properties for the **_rid** column definition:

- **other:dbrid** is found in the **_self** property of an item in the collection, after "dbs/".
- **other:collrid** is found in the **_self** property of an item in the collection, after "/colls/".
- **other:partitionpath** refers to the name of the partition specified when the collection was created.

```

<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">
  <rsb:info title="StaticRestaurants" description="Custom Schema for the
restaurants data set.">
    <!-- Column definitions -->
    <attr name="_rid"                xs:type="string"    key="true"
other:collrid="hWdRAKRi3Pg=" other:dbrid="hWdRAA=="
other:partitionpath="/name" />
    <attr name="borough"            xs:type="string"    />
    <attr name="cuisine"             xs:type="string"    />
    <attr name="address.building"    xs:type="string"    />
    <attr name="address.street"      xs:type="string"    />
    <attr name="address.coord.0"     xs:type="double"   />
    <attr name="address.coord.1"     xs:type="double"   />
    <input name="rows@next" desc="Internal attribute used for paging
through data." />
  </rsb:info>
  <rsb:script method="GET">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>
  <rsb:script method="POST">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>
  <rsb:script method="MERGE">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>
  <rsb:script method="DELETE">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>
</rsb:script>

```

Stored Procedures

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

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

Cosmos DB Adapter Stored Procedures

Name	Description
AddDocument	Insert entire JSON string to CosmosDB.
CreateSchema	Creates a schema file for the collection.
GetOAuthAccessToken	Gets the OAuth access token from CosmosDB.
GetOAuthAuthorizationURL	Gets the CosmosDB authorization URL. Access the URL returned in the output in a Web browser. This requests the access token that can be used as part of the connection string to CosmosDB.
RefreshOAuthAccessToken	Refreshes the OAuth access token used for authentication with CosmosDB.

AddDocument

Insert entire JSON string to CosmosDB.

Input

Name	Type	Description
Database	<i>String</i>	Name of the database.
Table	<i>String</i>	Name of the table.
PartitionKey	<i>String</i>	Partition key value of the table.
Document	<i>String</i>	The JSON string to be inserted.

Result Set Columns

Name	Type	Description
Success	<i>String</i>	Returns true if the operation is successful.

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	Accepts Output Streams	Description
SchemaName	<i>String</i>	<i>False</i>	The schema of the collection.

TableName	<i>String</i>	<i>False</i>	The name of the collection.
FileName	<i>String</i>	<i>False</i>	The full file path and name of the schema to generate. If not set, the FileData output is used instead.
FileStream	<i>String</i>	<i>True</i>	An instance of an output stream where file data is written to. Only used if FileName is not set.

Result Set Columns

Name	Type	Description
Result	<i>String</i>	Returns Success or Failure.
FileData	<i>String</i>	The generated schema encoded in Base64. Only returned if FileName is not set.

GetOAuthAccessToken

Gets the OAuth access token from CosmosDB.

Input

Name	Type	Description
AuthMode	<i>String</i>	The type of authentication you are attempting. Use App for a Windows application, or Web for Web-based applications. The default value is <i>APP</i> .

Verifier	<i>String</i>	A verifier returned by the service that must be input to return the access token. Needed only when using the Web auth mode. Obtained by navigating to the URL returned in GetOAuthAuthorizationUrl.
CallbackUrl	<i>String</i>	The URL the user will be redirected to after authorizing your application.
Scope	<i>String</i>	The scope or permissions you are requesting.
Prompt	<i>String</i>	Defaults to 'select_account' which prompts the user to select account while authenticating. Set to 'None', for no prompt, 'login' to force user to enter their credentials or 'consent' to trigger the OAuth consent dialog after the user signs in, asking the user to grant permissions to the app.

Result Set Columns

Name	Type	Description
OAuthAccessToken	<i>String</i>	The access token used for communication with CosmosDB.
OAuthRefreshToken	<i>String</i>	A token that may be used to obtain a new access token.
ExpiresIn	<i>String</i>	The remaining lifetime for the access token in seconds.

GetOAuthAuthorizationURL

Gets the CosmosDB authorization URL. Access the URL returned in the output in a Web browser. This requests the access token that can be used as part of the connection string to CosmosDB.

Input

Name	Type	Description
CallbackUrl	<i>String</i>	The URL that CosmosDB will return to after the user has authorized your app.
Scope	<i>String</i>	The scope or permissions you are requesting.
State	<i>String</i>	This field indicates any state that may be useful to your application upon receipt of the response. Your application receives the same value it sent, as this parameter makes a round-trip to CosmosDB authorization server and back. Uses include redirecting the user to the correct resource in your site, using nonces, and mitigating cross-site request forgery.
Prompt	<i>String</i>	Defaults to 'select_account' which prompts the user to select account while authenticating. Set to 'None', for no prompt, 'login' to force user to enter their credentials or 'consent' to trigger the OAuth consent dialog after the user signs in, asking the user to grant permissions to the app.

Result Set Columns

Name	Type	Description
URL	<i>String</i>	The URL to be entered into a Web browser to obtain the verifier token and authorize the data provider with.

RefreshOAuthAccessToken

Refreshes the OAuth access token used for authentication with CosmosDB.

Input

Name	Type	Description
OAuthRefreshToken	<i>String</i>	The refresh token returned from the original authorization code exchange.
Scope	<i>String</i>	The scope or permissions you are requesting.

Result Set Columns

Name	Type	Description
OAuthAccessToken	<i>String</i>	The new OAuthAccessToken returned from the service.
OAuthRefreshToken	<i>String</i>	A token that may be used to obtain a new access token.
ExpiresIn	<i>String</i>	The remaining lifetime on the access token.

SQL Compliance

The Cosmos DB 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 Cosmos DB 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.

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 Cosmos DB adapter:

```

SELECT {
  [ TOP <numeric_literal> ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
    }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  } [ , ... ]
  [
    JOIN <table_reference> [ ON <search_condition> ] [ [ AS ]
<identifier> ]
  ] [ ... ]
  [ WHERE <search_condition> ]
  [ GROUP BY <column_reference> [ , ... ]
  [
    LIMIT <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> { = | > | < | >= | <= | <> | != | IN | AND | OR } [
    <expression> ]
    } [ { AND | OR } ... ]

```

Examples

1. Return all columns:

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

2. Rename a column:

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

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

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

4. Search data:

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

5. The Cosmos DB APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, IN, AND, OR.

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

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

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

7. Summarize data:

```
SELECT CompanyName, MAX(Balance) FROM [CData].[Entities].Customers  
GROUP BY CompanyName
```

See [Aggregate Functions](#) for details.

8. 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.

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].[Entities].Customers WHERE Country = 'US'
```

AVG

Returns the average of the column values.

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

MIN

Returns the minimum column value.

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

MAX

Returns the maximum column value.

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

SUM

Returns the total sum of the column values.

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

JOIN Queries

The Cosmos DB 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 Cosmos DB API.

For example, consider the following query from Cosmos DB'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.

Projection Functions

ABS(numeric_expr)

Returns the absolute (positive) value of the specified numeric expression.

- **numeric_expr**: A numeric expression.

ACOS(numeric_expr)

Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.

- **numeric_expr**: A numeric expression.

ASIN(numeric_expr)

Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.

- **numeric_expr**: A numeric expression.

ATAN(numeric_expr)

Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.

- **numeric_expr**: A numeric expression.

CEILING(numeric_expr)

Returns the smallest integer value greater than, or equal to, the specified numeric expression.

- **numeric_expr**: A numeric expression.

COS(numeric_expr)

Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

COT(numeric_expr)

Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.

- **numeric_expr**: A numeric expression.

DEGREES(numeric_expr)

Returns the corresponding angle in degrees for an angle specified in radians.

- **numeric_expr**: A numeric expression.

FLOOR(numeric_expr)

Returns the largest integer less than or equal to the specified numeric expression.

- **numeric_expr**: A numeric expression.

EXP(numeric_expr)

Returns the exponential value of the specified numeric expression.

- **numeric_expr**: A numeric expression.

LOG10(numeric_expr)

Returns the base-10 logarithm of the specified numeric expression.

- **numeric_expr**: A numeric expression.

RADIANS(numeric_expr)

Returns radians when a numeric expression, in degrees, is entered.

- **numeric_expr**: A numeric expression.

RAND()

Returns a randomly generated numeric value from [0,1).

ROUND(numeric_expr)

Returns a numeric value, rounded to the closest integer value.

- **numeric_expr**: A numeric expression.

SIGN(numeric_expr)

Returns the positive (+1), zero (0), or negative (-1) sign of the specified numeric expression.

- **numeric_expr**: A numeric expression.

SIN(numeric_expr)

Returns the trigonometric sine of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

SQRT(numeric_expr)

Returns the square root of the specified numeric value.

- **numeric_expr**: A numeric expression.

SQUARE(numeric_expr)

Returns the square of the specified numeric value.

- **numeric_expr**: A numeric expression.

TAN(numeric_expr)

Returns the tangent of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

TRUNC(numeric_expr)

Returns a numeric value, truncated to the closest integer value.

- **numeric_expr**: A numeric expression.

ATN2(y_expr, x_expr)

Returns the principal value of the arc tangent of y/x, expressed in radians.

- **y_expr**: The y numeric expression.
- **x_expr**: The x numeric expression.

LOG(numeric_expr [, base])

Returns the natural logarithm of the specified numeric expression.

- **numeric_expr**: A numeric expression.
- **base**: Optional numeric argument that sets the base for the logarithm.

PI()

Returns the constant value of PI.

POWER(numeric_expr, power_expr)

Returns the value of the specified expression to the specified power.

- **numeric_expr**: A numeric expression.
- **power_expr**: Is the power to which to raise numeric_expr.

IS_ARRAY(expr)

Returns a Boolean value indicating if the type of the specified expression is an array.

- **expr**: Any valid expression.

IS_BOOL(expr)

Returns a Boolean value indicating if the type of the specified expression is a Boolean.

- **expr**: Any valid expression.

IS_DEFINED(expr)

Returns a Boolean indicating if the property has been assigned a value.

- **expr**: Any valid expression.

IS_NULL(expr)

Returns a Boolean value indicating if the type of the specified expression is null.

- **expr**: Any valid expression.

IS_NUMBER(expr)

Returns a Boolean value indicating if the type of the specified expression is a number.

- **expr**: Any valid expression.

IS_OBJECT(expr)

Returns a Boolean value indicating if the type of the specified expression is a JSON object.

- **expr**: Any valid expression.

IS_PRIMITIVE(expr)

Returns a Boolean value indicating if the type of the specified expression is a primitive (string, Boolean, numeric, or null).

- **expr**: Any valid expression.

IS_STRING(expr)

Returns a Boolean value indicating if the type of the specified expression is a string.

- **expr**: Any valid expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns a string that is the result of concatenating two or more string values.

- **str1**: The first string to concatenate.
- **str2**: The second string to concatenate.
- **str3**: The third string to concatenate.

CONTAINS(str1, str2)

Returns a Boolean indicating whether the first string expression contains the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

ENDSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression ends with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

INDEX_OF(str1, str2)

Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, num_expr)

Returns the left part of a string with the specified number of characters.

- **str**: A valid string expression.
- **num_expr**: The number of characters to return.

LENGTH(str)

Returns the number of characters of the specified string expression.

- **str**: Any valid string expression.

LOWER(str)

Returns a string expression after converting uppercase character data to lowercase.

- **str**: Any valid string expression.

LTRIM(str)

Returns a string expression after it removes leading blanks.

- **str**: Any valid string expression.

REPLACE(original_value, from_value, to_value)

Replaces all occurrences of a specified string value with another string value.

- **original_value**: The string to search in.
- **from_value**: The string to search for.
- **to_value**: The string to replace instances of from_value.

REPLICATE(str, repeat_num)

Repeats a string value a specified number of times.

- **str**: The string expression to repeat.
- **repeat_num**: The number of times to repeat the str expression.

REVERSE(str)

Returns the reverse order of a string value.

- **str**: Any valid string expression.

RIGHT(str, num_expr)

Returns the right part of a string with the specified number of characters.

- **str**: Any valid string expression.
- **num_expr**: The starting index.

RTRIM(str)

Returns a string expression after it removes trailing blanks.

- **str**: Any valid string expression.

STARTSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression starts with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

SUBSTRING(str, start_index, length)

Returns part of a string expression starting at the specified character zero-based position and continues to the specified length, or to the end of the string.

- **str**: Any valid string expression.
- **start_index**: The starting index.
- **length**: The length of the string to return.

TOSTRING(expr)

Returns a string representation of scalar expression.

- **expr**: Any valid expression.

TRIM(str)

Returns a string expression after it removes leading and trailing blanks.

- **str**: Any valid string expression.

UPPER(str)

Returns a string expression after converting lowercase character data to uppercase.

- **str**: Any valid string expression.

ARRAY_CONCAT(array_exp1, array_exp2 [, array_exp3])

Returns an array that is the result of concatenating two or more array values.

- **array_exp1**: Any valid array expression.
- **array_exp2**: Any valid array expression.
- **array_exp3**: Any valid array expression.

ARRAY_CONTAINS(array_exp, expr [, bool_expr])

Returns a Boolean indicating whether the array contains the specified value. You can check for a partial or full match of an object by using a boolean expression within the command.

- **array_exp1**: Any array expression.
- **expr**: The expression to search for.
- **bool_expr**: If it's set to 'true' and if the specified search value is an object, the command checks for a partial match (the search object is a subset of one of the objects). If it's set to 'false', the command checks for a full match of all objects within the array. The default value if not specified is false.

ARRAY_LENGTH(array_exp)

Returns the number of elements of the specified array expression.

- **array_exp**: Any valid array expression.

ARRAY_SLICE(array_exp, start_index, max_size)

Returns part of an array expression.

- **array_exp**: Any valid array expression.
- **start_index**: Zero-based numeric index at which to begin the array. Negative values may be used to specify the starting index relative to the last element of the array i.e. -1 references the last element in the array.
- **max_size**: Maximum number of elements in the resulting array.

ST_DISTANCE(spatial_expr1, spatial_expr2)

Returns the distance between the two GeoJSON Point, Polygon, or LineString expressions.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_WITHIN(spatial_expr1, spatial_expr2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument is within the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1:** Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_INTERSECTS(1, 2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument intersects the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1:** Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALID(spatial_expr)

Returns a Boolean value indicating whether the specified GeoJSON Point, Polygon, or LineString expression is valid.

- **spatial_expr:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALIDDETAILED(spatial_expr)

Returns a JSON value containing a Boolean value if the specified GeoJSON Point, Polygon, or LineString expression is valid, and if invalid, additionally the reason as a string value.

- **spatial_expr:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

Predicate Functions

ABS(numeric_expr)

Returns the absolute (positive) value of the specified numeric expression.

- **numeric_expr**: A numeric expression.

ACOS(numeric_expr)

Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.

- **numeric_expr**: A numeric expression.

ASIN(numeric_expr)

Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.

- **numeric_expr**: A numeric expression.

ATAN(numeric_expr)

Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.

- **numeric_expr**: A numeric expression.

CEILING(numeric_expr)

Returns the smallest integer value greater than, or equal to, the specified numeric expression.

- **numeric_expr**: A numeric expression.

COS(numeric_expr)

Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

COT(numeric_expr)

Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.

- **numeric_expr**: A numeric expression.

DEGREES(numeric_expr)

Returns the corresponding angle in degrees for an angle specified in radians.

- **numeric_expr**: A numeric expression.

FLOOR(numeric_expr)

Returns the largest integer less than or equal to the specified numeric expression.

- **numeric_expr**: A numeric expression.

EXP(numeric_expr)

Returns the exponential value of the specified numeric expression.

- **numeric_expr**: A numeric expression.

LOG10(numeric_expr)

Returns the base-10 logarithm of the specified numeric expression.

- **numeric_expr**: A numeric expression.

RADIANS(numeric_expr)

Returns radians when a numeric expression, in degrees, is entered.

- **numeric_expr**: A numeric expression.

RAND()

Returns a randomly generated numeric value from [0,1).

ROUND(numeric_expr)

Returns a numeric value, rounded to the closest integer value.

- **numeric_expr**: A numeric expression.

SIGN(numeric_expr)

Returns the positive (+1), zero (0), or negative (-1) sign of the specified numeric expression.

- **numeric_expr**: A numeric expression.

SIN(numeric_expr)

Returns the trigonometric sine of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

SQRT(numeric_expr)

Returns the square root of the specified numeric value.

- **numeric_expr**: A numeric expression.

SQUARE(numeric_expr)

Returns the square of the specified numeric value.

- **numeric_expr**: A numeric expression.

TAN(numeric_expr)

Returns the tangent of the specified angle, in radians, in the specified expression.

- **numeric_expr**: A numeric expression.

TRUNC(numeric_expr)

Returns a numeric value, truncated to the closest integer value.

- **numeric_expr**: A numeric expression.

ATN2(y_expr, x_expr)

Returns the principal value of the arc tangent of y/x, expressed in radians.

- **y_expr**: The y numeric expression.
- **x_expr**: The x numeric expression.

LOG(numeric_expr [, base])

Returns the natural logarithm of the specified numeric expression.

- **numeric_expr**: A numeric expression.
- **base**: Optional numeric argument that sets the base for the logarithm.

PI()

Returns the constant value of PI.

POWER(numeric_expr, power_expr)

Returns the value of the specified expression to the specified power.

- **numeric_expr**: A numeric expression.
- **power_expr**: Is the power to which to raise numeric_expr.

IS_ARRAY(expr)

Returns a Boolean value indicating if the type of the specified expression is an array.

- **expr**: Any valid expression.

IS_BOOL(expr)

Returns a Boolean value indicating if the type of the specified expression is a Boolean.

- **expr**: Any valid expression.

IS_DEFINED(expr)

Returns a Boolean indicating if the property has been assigned a value.

- **expr**: Any valid expression.

IS_NULL(expr)

Returns a Boolean value indicating if the type of the specified expression is null.

- **expr**: Any valid expression.

IS_NUMBER(expr)

Returns a Boolean value indicating if the type of the specified expression is a number.

- **expr**: Any valid expression.

IS_OBJECT(expr)

Returns a Boolean value indicating if the type of the specified expression is a JSON object.

- **expr**: Any valid expression.

IS_PRIMITIVE(expr)

Returns a Boolean value indicating if the type of the specified expression is a primitive (string, Boolean, numeric, or null).

- **expr**: Any valid expression.

IS_STRING(expr)

Returns a Boolean value indicating if the type of the specified expression is a string.

- **expr**: Any valid expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns a string that is the result of concatenating two or more string values.

- **str1**: The first string to concatenate.
- **str2**: The second string to concatenate.
- **str3**: The third string to concatenate.

CONTAINS(str1, str2)

Returns a Boolean indicating whether the first string expression contains the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

ENDSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression ends with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

INDEX_OF(str1, str2)

Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, num_expr)

Returns the left part of a string with the specified number of characters.

- **str**: A valid string expression.
- **num_expr**: The number of characters to return.

LENGTH(str)

Returns the number of characters of the specified string expression.

- **str**: Any valid string expression.

LOWER(str)

Returns a string expression after converting uppercase character data to lowercase.

- **str**: Any valid string expression.

LTRIM(str)

Returns a string expression after it removes leading blanks.

- **str**: Any valid string expression.

REPLACE(original_value, from_value, to_value)

Replaces all occurrences of a specified string value with another string value.

- **original_value**: The string to search in.
- **from_value**: The string to search for.
- **to_value**: The string to replace instances of from_value.

REPLICATE(str, repeat_num)

Repeats a string value a specified number of times.

- **str**: The string expression to repeat.
- **repeat_num**: The number of times to repeat the str expression.

REVERSE(str)

Returns the reverse order of a string value.

- **str**: Any valid string expression.

RIGHT(str, num_expr)

Returns the right part of a string with the specified number of characters.

- **str**: Any valid string expression.
- **num_expr**: The starting index.

RTRIM(str)

Returns a string expression after it removes trailing blanks.

- **str**: Any valid string expression.

STARTSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression starts with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

SUBSTRING(str, start_index, length)

Returns part of a string expression starting at the specified character zero-based position and continues to the specified length, or to the end of the string.

- **str**: Any valid string expression.
- **start_index**: The starting index.

- **length:** The length of the string to return.

TOSTRING(expr)

Returns a string representation of scalar expression.

- **expr:** Any valid expression.

TRIM(str)

Returns a string expression after it removes leading and trailing blanks.

- **str:** Any valid string expression.

UPPER(str)

Returns a string expression after converting lowercase character data to uppercase.

- **str:** Any valid string expression.

ARRAY_CONCAT(array_exp1, array_exp2 [, array_exp3])

Returns an array that is the result of concatenating two or more array values.

- **array_exp1:** Any valid array expression.
- **array_exp2:** Any valid array expression.
- **array_exp3:** Any valid array expression.

ARRAY_CONTAINS(array_exp, expr [, bool_expr])

Returns a Boolean indicating whether the array contains the specified value. You can check for a partial or full match of an object by using a boolean expression within the command.

- **array_exp1:** Any array expression.
- **expr:** The expression to search for.
- **bool_expr:** If it's set to 'true' and if the specified search value is an object, the

command checks for a partial match (the search object is a subset of one of the objects). If it's set to 'false', the command checks for a full match of all objects within the array. The default value if not specified is false.

ARRAY_LENGTH(array_exp)

Returns the number of elements of the specified array expression.

- **array_exp**: Any valid array expression.

ARRAY_SLICE(array_exp, start_index, max_size)

Returns part of an array expression.

- **array_exp**: Any valid array expression.
- **start_index**: Zero-based numeric index at which to begin the array. Negative values may be used to specify the starting index relative to the last element of the array i.e. -1 references the last element in the array.
- **max_size**: Maximum number of elements in the resulting array.

ST_DISTANCE(spatial_expr1, spatial_expr2)

Returns the distance between the two GeoJSON Point, Polygon, or LineString expressions.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_WITHIN(spatial_expr1, spatial_expr2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument is within the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

- **spatial_expr2:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_INTERSECTS(1, 2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument intersects the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1:** Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALID(spatial_expr)

Returns a Boolean value indicating whether the specified GeoJSON Point, Polygon, or LineString expression is valid.

- **spatial_expr:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALIDDETAILED(spatial_expr)

Returns a JSON value containing a Boolean value if the specified GeoJSON Point, Polygon, or LineString expression is valid, and if invalid, additionally the reason as a string value.

- **spatial_expr:** Is any valid GeoJSON Point, Polygon, or LineString object expression.

SELECT INTO Statements

You can use the SELECT INTO statement to export formatted data to a file.

Data Export with an SQL Query

The following query exports data into a file formatted in comma-separated values (CSV):

```
boolean ret = stat.execute("SELECT City, CompanyName INTO "csv://c:/[CData].[Entities].Customers.txt" FROM "[CData].[Entities].Customers" WHERE Country = 'US'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

You can specify other file formats in the URI. The following example exports tab-separated values:

```
Statement stat = conn.createStatement();
boolean ret = stat.execute("SELECT * INTO "[CData].[Entities].Customers" IN 'csv://filename=c:/[CData].[Entities].Customers.csv;delimiter=tab' FROM "[CData].[Entities].Customers" WHERE Country = 'US'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

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].[Entities].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].[Entities].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:cosmosdb:AccountEndpoint=myAccountEndpoint;AccountKey=myAccountKey;");
String cmd = "DELETE FROM [CData].[Entities].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 Syntax

```
"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 Syntax

```
"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 \(Transact-SQL\)](#)

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 type of authentication to use when connecting to Azure Cosmos DB.
AccountEndpoint	The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.
AccountKey	A master key token or a resource token for connecting to the Azure Cosmos DB REST API.
TokenType	Denotes the type of token: master or resource.

Azure Authentication

Property	Description
AzureTenant	The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.
AzureEnvironment	The Azure Environment to use when establishing a connection.

OAuth

Property	Description
InitiateOAuth	Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.
OAuthClientId	The client Id assigned when you register your application with an OAuth authorization server.
OAuthClientSecret	The client secret assigned when you register your application with an OAuth authorization server.
OAuthAccessToken	The access token for connecting using OAuth.
OAuthSettingsLocation	The location of the settings file where OAuth values are saved when InitiateOAuth is set to GETANDREFRESH or REFRESH. Alternatively, this can be held in memory by specifying a value starting with memory://.
CallbackURL	The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your app settings.
OAuthGrantType	The grant type for the OAuth flow.
OAuthVerifier	The verifier code returned from the OAuth authorization URL.
OAuthRefreshToken	The OAuth refresh token for the corresponding OAuth access token.

OAuthExpiresIn	The lifetime in seconds of the OAuth AccessToken.
OAuthTokenTimestamp	The Unix epoch timestamp in milliseconds when the current Access Token was created.

SSL

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.

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.

Proxy

Property	Description
ProxyAutoDetect	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

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.
-----------------	---

Schema	Specify the Azure Cosmos DB database you want to work with.
---------------	---

Miscellaneous

Property	Description
CalculateAggregates	Specifies whether will return the calculated value of the aggregates or grouped by partiton range.
ConsistencyLevel	Denotes the type of token: master or resource.
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 rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
MaxThreads	Specifies the maximum number of concurrent requests for Batch CUD (Create, Update, Delete) operations.
MultiThreadCount	Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.

Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Azure Cosmos DB.
Readonly	You can use this property to enforce read-only access to Azure Cosmos DB from the provider.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
SeparatorCharacter	The character or characters used to denote hierarchy.
SetPartitionKeyAsPK	Whether or not to use the collection's Partition Key field as part of composite Primary Key for the corresponding exposed table.
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.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
UseRidAsPk	Set this property to false to switch using the id column as primary key instead the default _rid.
WriteThroughputBudget	Defines the Requests Units (RU) budget per Second that the Batch CUD (Create, Update, Delete) operations should not exceed.

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 type of authentication to use when connecting to Azure Cosmos DB.
AccountEndpoint	The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.
AccountKey	A master key token or a resource token for connecting to the Azure Cosmos DB REST API.
TokenType	Denotes the type of token: master or resource.

AuthScheme

The type of authentication to use when connecting to Azure Cosmos DB.

Possible Values

AccountKey, AzureAD, AzureServicePrincipal

Data Type

string

Default Value

"AccountKey"

Remarks

- AccountKey: Set this to perform authentication with [AccountKey](#) and [AccountEndpoint](#).
- AzureAD: Set this to perform Azure Active Directory OAuth authentication.
- AzureServicePrincipal: Set this to authenticate as an Azure Service Principal.

AccountEndpoint

The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.

Data Type

string

Default Value

""

Remarks

The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.

AccountKey

A master key token or a resource token for connecting to the Azure Cosmos DB REST API.

Data Type

string

Default Value

""

Remarks

In the Azure portal, navigate to the Cosmos DB service and select your Azure Cosmos DB account. From the resource menu, go to the Keys page. Find the PRIMARY KEY value and set Token to this value.

TokenType

Denotes the type of token: master or resource.

Possible Values

master, resource

Data Type

string

Default Value

"master"

Remarks

The master key is created during the creation of an account. There are two sets of master keys, the primary key and the secondary key. The administrator of the account can then exercise key rotation using the secondary key. In addition, the account administrator can also regenerate the keys as needed.

Resource tokens are created when users in a database are set up with access permissions for precise access control on a resource, also known as a permission resource. A permission resource contains a hash resource token constructed with the information regarding the resource path and access type a user has access to. The permission resource token is time bound and the validity period can be overridden. When a permission resource is acted upon on (POST, GET, PUT), a new resource token is generated.

Azure Authentication

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

Property	Description
AzureTenant	The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.
AzureEnvironment	The Azure Environment to use when establishing a connection.

AzureTenant

The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.

Data Type

string

Default Value

""

Remarks

The Microsoft Online tenant being used to access data. For instance, contoso.onmicrosoft.com. Alternatively, specify the tenant Id. This value is the directory Id in the Azure Portal > Azure Active Directory > Properties.

Typically it is not necessary to specify the Tenant. This can be automatically determined by Microsoft when using the [OAuthGrantType](#) set to CODE (default). However, it may fail in the case that the user belongs to multiple tenants. For instance, if an Admin of domain A invites a user of domain B to be a guest user. The user will now belong to both tenants. It is a good practice to specify the Tenant, although in general things should normally work without having to specify it.

The AzureTenant is required when setting [OAuthGrantType](#) to CLIENT. When using client credentials, there is no user context. The credentials are taken from the context of the app itself. While Microsoft still allows client credentials to be obtained without specifying which Tenant, it has a much lower probability of picking the specific tenant you want to work with. For this reason, we require AzureTenant to be explicitly stated for all client credentials connections to ensure you get credentials that are applicable for the domain you intend to connect to.

The Microsoft Online tenant being used to access data. For instance, contoso.onmicrosoft.com. Alternatively, specify the tenant Id. This value is the directory Id in the Azure Portal > Azure Active Directory > Properties.

Typically it is not necessary to specify the Tenant. This can be automatically determined by Microsoft when using the [OAuthGrantType](#) set to CODE (default). However, it may fail in the case that the user belongs to multiple tenants. For instance, if an Admin of domain A

invites a user of domain B to be a guest user. The user will now belong to both tenants. It is a good practice to specify the Tenant, although in general things should normally work without having to specify it.

The AzureTenant is required when setting OAuthGrantType to CLIENT. When using client credentials, there is no user context. The credentials are taken from the context of the app itself. While Microsoft still allows client credentials to be obtained without specifying which Tenant, it has a much lower probability of picking the specific tenant you want to work with. For this reason, we require AzureTenant to be explicitly stated for all client credentials connections to ensure you get credentials that are applicable for the domain you intend to connect to.

AzureEnvironment

The Azure Environment to use when establishing a connection.

Possible Values

GLOBAL, CHINA, USGOVT, USGOVTDOD

Data Type

string

Default Value

"GLOBAL"

Remarks

In most cases, leaving the environment set to global will work. However, if your Azure Account has been added to a different environment, the AzureEnvironment may be used to specify which environment. The available values are GLOBAL, CHINA, USGOVT, USGOVTDOD.

OAuth

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

Property	Description
InitiateOAuth	Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.
OAuthClientId	The client Id assigned when you register your application with an OAuth authorization server.
OAuthClientSecret	The client secret assigned when you register your application with an OAuth authorization server.
OAuthAccessToken	The access token for connecting using OAuth.
OAuthSettingsLocation	The location of the settings file where OAuth values are saved when InitiateOAuth is set to GETANDREFRESH or REFRESH. Alternatively, this can be held in memory by specifying a value starting with memory://.
CallbackURL	The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your app settings.
OAuthGrantType	The grant type for the OAuth flow.
OAuthVerifier	The verifier code returned from the OAuth authorization URL.
OAuthRefreshToken	The OAuth refresh token for the corresponding OAuth access token.
OAuthExpiresIn	The lifetime in seconds of the OAuth AccessToken.
OAuthTokenTimestamp	The Unix epoch timestamp in milliseconds when the current Access Token was created.

InitiateOAuth

Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.

Possible Values

OFF, GETANDREFRESH, REFRESH

Data Type

string

Default Value

"OFF"

Remarks

The following options are available:

1. **OFF**: Indicates that the OAuth flow will be handled entirely by the user. An OAuthAccessToken will be required to authenticate.
2. **GETANDREFRESH**: Indicates that the entire OAuth Flow will be handled by the adapter. If no token currently exists, it will be obtained by prompting the user via the browser. If a token exists, it will be refreshed when applicable.
3. **REFRESH**: Indicates that the adapter will only handle refreshing the OAuthAccessToken. The user will never be prompted by the adapter to authenticate via the browser. The user must handle obtaining the OAuthAccessToken and OAuthRefreshToken initially.

OAuthClientId

The client Id assigned when you register your application with an OAuth authorization server.

Data Type

string

Default Value

""

Remarks

As part of registering an OAuth application, you will receive the OAuthClientId value, sometimes also called a consumer key, and a client secret, the [OAuthClientSecret](#).

OAuthClientSecret

The client secret assigned when you register your application with an OAuth authorization server.

Data Type

string

Default Value

""

Remarks

As part of registering an OAuth application, you will receive the [OAuthClientId](#), also called a consumer key. You will also receive a client secret, also called a consumer secret. Set the client secret in the OAuthClientSecret property.

OAuthAccessToken

The access token for connecting using OAuth.

Data Type

string

Default Value

""

Remarks

The `OAuthAccessToken` property is used to connect using OAuth. The `OAuthAccessToken` is retrieved from the OAuth server as part of the authentication process. It has a server-dependent timeout and can be reused between requests.

The access token is used in place of your user name and password. The access token protects your credentials by keeping them on the server.

OAuthSettingsLocation

The location of the settings file where OAuth values are saved when `InitiateOAuth` is set to `GETANDREFRESH` or `REFRESH`. Alternatively, this can be held in memory by specifying a value starting with `memory://`.

Data Type

string

Default Value

"%APPDATA%\CDData\CosmosDB Data Provider\OAuthSettings.txt"

Remarks

When `InitiateOAuth` is set to `GETANDREFRESH` or `REFRESH`, the adapter saves OAuth values to avoid requiring the user to manually enter OAuth connection properties and allowing the credentials to be shared across connections or processes.

Alternatively to specifying a file path, memory storage can be used instead. Memory locations are specified by using a value starting with `'memory://'` followed by a unique identifier for that set of credentials (ex: `memory://user1`). The identifier can be anything you choose but should be unique to the user. Unlike with the file based storage, you must manually store the credentials when closing the connection with memory storage to be able to set them in the connection when the process is started again. The OAuth property values can be retrieved with a query to the `sys_connection_props` system table. If there are multiple connections using the same credentials, the properties should be read from the last connection to be closed.

If left unspecified, the default location is "%APPDATA%\CDData\CosmosDB Data Provider\OAuthSettings.txt" 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

CallbackURL

The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your app settings.

Data Type

string

Default Value

""

Remarks

During the authentication process, the OAuth authorization server redirects the user to this URL. This value must match the callback URL you specify in your app settings.

OAuthGrantType

The grant type for the OAuth flow.

Possible Values

CODE, CLIENT, PASSWORD

Data Type

string

Default Value

"CODE"

Remarks

The following options are available: CODE,CLIENT,PASSWORD

OAuthVerifier

The verifier code returned from the OAuth authorization URL.

Data Type

string

Default Value

""

Remarks

The verifier code returned from the OAuth authorization URL. This can be used on systems where a browser cannot be launched such as headless systems.

Authentication on Headless Machines

See to obtain the OAuthVerifier value.

Set [OAuthSettingsLocation](#) along with OAuthVerifier. When you connect, the adapter exchanges the OAuthVerifier for the OAuth authentication tokens and saves them, encrypted, to the specified file. Set [InitiateOAuth](#) to GETANDREFRESH automate the exchange.

Once the OAuth settings file has been generated, you can remove OAuthVerifier from the connection properties and connect with [OAuthSettingsLocation](#) set.

To automatically refresh the OAuth token values, set [OAuthSettingsLocation](#) and additionally set [InitiateOAuth](#) to REFRESH.

OAuthRefreshToken

The OAuth refresh token for the corresponding OAuth access token.

Data Type

string

Default Value

""

Remarks

The OAuthRefreshToken property is used to refresh the [OAuthAccessToken](#) when using OAuth authentication.

OAuthExpiresIn

The lifetime in seconds of the OAuth AccessToken.

Data Type

string

Default Value

""

Remarks

Pair with OAuthTokenTimestamp to determine when the AccessToken will expire.

OAuthTokenTimestamp

The Unix epoch timestamp in milliseconds when the current Access Token was created.

Data Type

string

Default Value

""

Remarks

Pair with OAuthExpiresIn to determine when the AccessToken will expire.

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, JKSLOB, 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.

JKS BLOB	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.
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
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----- MIICHTCCAe4CAQAwDQYJKoZIhvd.....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----- MIGfMA0GCSq.....AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ca1b1bda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801c bb150d

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.

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. Note that by default, the adapter connects to the system proxy; to disable this behavior and connect to one of the following proxy types, set [ProxyAutoDetect](#) to false.

Type	Default Port	Description
TUNNEL	80	When this is set, the adapter opens a connection to Cosmos DB 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.

To connect to HTTP proxies, use [ProxyServer](#) and [ProxyPort](#). To authenticate to HTTP proxies, use [ProxyAuthScheme](#), [ProxyUser](#), and [ProxyPassword](#).

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. Use [ProxyServer](#) to connect to an HTTP proxy.

Note that the adapter uses the system proxy by default. To use a different proxy, set [ProxyAutoDetect](#) to false.

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](#).

Proxy

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

Property	Description
ProxyAutoDetect	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

ProxyAutoDetect

This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.

Data Type

bool

Default Value

true

Remarks

This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.

NOTE: When this property is set to True, the proxy used is determined as follows:

- A search from the JVM properties (**http.proxy**, **https.proxy**, **socksProxy**, etc.) is performed.
- In the case that the JVM properties don't exist, a search from **java.home/lib/net.properties** is performed.
- In the case that java.net.useSystemProxies is set to True, a search from **the SystemProxy** is performed.
- In Windows only, an attempt is made to retrieve these properties from the **Internet Options** in the **registry**.

To connect to an HTTP proxy, see [ProxyServer](#). For other proxies, such as SOCKS or tunneling, see [FirewallType](#).

ProxyServer

The hostname or IP address of a proxy to route HTTP traffic through.

Data Type

string

Default Value

""

Remarks

The hostname or IP address of a proxy to route HTTP traffic through. The adapter can use the HTTP, Windows (NTLM), or Kerberos authentication types to authenticate to an HTTP proxy.

If you need to connect through a SOCKS proxy or tunnel the connection, see [FirewallType](#).

By default, the adapter uses the system proxy. If you need to use another proxy, set [ProxyAutoDetect](#) to false.

ProxyPort

The TCP port the ProxyServer proxy is running on.

Data Type

int

Default Value

80

Remarks

The port the HTTP proxy is running on that you want to redirect HTTP traffic through. Specify the HTTP proxy in [ProxyServer](#). For other proxy types, see [FirewallType](#).

ProxyAuthScheme

The authentication type to use to authenticate to the ProxyServer proxy.

Possible Values

BASIC, DIGEST, NONE, NEGOTIATE, NTLM, PROPRIETARY

Data Type

string

Default Value

"BASIC"

Remarks

This value specifies the authentication type to use to authenticate to the HTTP proxy specified by [ProxyServer](#) and [ProxyPort](#).

Note that the adapter will use the system proxy settings by default, without further configuration needed; if you want to connect to another proxy, you will need to set [ProxyAutoDetect](#) to false, in addition to [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

The authentication type can be one of the following:

- **BASIC:** The adapter performs HTTP BASIC authentication.
- **DIGEST:** The adapter performs HTTP DIGEST authentication.
- **NEGOTIATE:** The adapter retrieves an NTLM or Kerberos token based on the applicable protocol for authentication.
- **PROPRIETARY:** The adapter does not generate an NTLM or Kerberos token. You must supply this token in the Authorization header of the HTTP request.

If you need to use another authentication type, such as SOCKS 5 authentication, see [FirewallType](#).

ProxyUser

A user name to be used to authenticate to the ProxyServer proxy.

Data Type

string

Default Value

""

Remarks

The [ProxyUser](#) and [ProxyPassword](#) options are used to connect and authenticate against the HTTP proxy specified in [ProxyServer](#).

You can select one of the available authentication types in [ProxyAuthScheme](#). If you are using HTTP authentication, set this to the user name of a user recognized by the HTTP proxy. If you are using Windows or Kerberos authentication, set this property to a user name in one of the following formats:

```
user@domain  
domain\user
```

ProxyPassword

A password to be used to authenticate to the ProxyServer proxy.

Data Type

string

Default Value

""

Remarks

This property is used to authenticate to an HTTP proxy server that supports NTLM (Windows), Kerberos, or HTTP authentication. To specify the HTTP proxy, you can set [ProxyServer](#) and [ProxyPort](#). To specify the authentication type, set [ProxyAuthScheme](#).

If you are using HTTP authentication, additionally set [ProxyUser](#) and [ProxyPassword](#) to HTTP proxy.

If you are using NTLM authentication, set [ProxyUser](#) and [ProxyPassword](#) to your Windows password. You may also need these to complete Kerberos authentication.

For SOCKS 5 authentication or tunneling, see [FirewallType](#).

By default, the adapter uses the system proxy. If you want to connect to another proxy, set [ProxyAutoDetect](#) to false.

ProxySSLType

The SSL type to use when connecting to the ProxyServer proxy.

Possible Values

AUTO, ALWAYS, NEVER, TUNNEL

Data Type

string

Default Value

"AUTO"

Remarks

This property determines when to use SSL for the connection to an HTTP proxy specified by [ProxyServer](#). This value can be AUTO, ALWAYS, NEVER, or TUNNEL. The applicable values are the following:

AUTO	Default setting. If the URL is an HTTPS URL, the adapter will use the TUNNEL option. If the URL is an HTTP URL, the component will use the NEVER option.
ALWAYS	The connection is always SSL enabled.
NEVER	The connection is not SSL enabled.
TUNNEL	The connection is through a tunneling proxy. The proxy server opens a connection to the remote host and traffic flows back and forth through the proxy.

ProxyExceptions

A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

Data Type

string

Default Value

""

Remarks

The [ProxyServer](#) is used for all addresses, except for addresses defined in this property. Use semicolons to separate entries.

Note that the adapter uses the system proxy settings by default, without further configuration needed; if you want to explicitly configure proxy exceptions for this connection, you need to set [ProxyAutoDetect](#) = false, and configure [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

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.
Schema	Specify the Azure Cosmos DB database you want to work with.

Location

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

Data Type

string

Default Value

"%APPDATA%\\CData\\CosmosDB 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\\CosmosDB 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

Schema

Specify the Azure Cosmos DB database you want to work with.

Data Type

string

Default Value

""

Remarks

Specify the Cosmos DB database you want to work with.

Miscellaneous

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

Property	Description
CalculateAggregates	Specifies whether will return the calculated value of the aggregates or grouped by partiton range.
ConsistencyLevel	Denotes the type of token: master or resource.
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 rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
MaxThreads	Specifies the maximum number of concurrent requests for Batch CUD (Create, Update, Delete) operations.
MultiThreadCount	Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Azure

	Cosmos DB.
Readonly	You can use this property to enforce read-only access to Azure Cosmos DB from the provider.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
SeparatorCharacter	The character or characters used to denote hierarchy.
SetPartitionKeyAsPK	Whether or not to use the collection's Partition Key field as part of composite Primary Key for the corresponding exposed table.
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.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
UseRidAsPk	Set this property to false to switch using the id column as primary key instead the default _rid.
WriteThroughputBudget	Defines the Requests Units (RU) budget per Second that the Batch CUD (Create, Update, Delete) operations should not exceed.

CalculateAggregates

Specifies whether will return the calculated value of the aggregates or grouped by partiton range.

Data Type

bool

Default Value

true

Remarks

Specifies whether will return the calculated value of the aggregates or grouped by partiton range.

ConsistencyLevel

Denotes the type of token: master or resource.

Possible Values

STRONG, BOUNDED, SESSION, CONSISTENTPREFIX, EVENTUAL

Data Type

string

Default Value

"SESSION"

Remarks

The consistency level override for read options against documents and attachments. The valid values are: Strong, Bounded, Session, or Eventual (in order of strongest to weakest). The override must be the same or weaker than the account's configured consistency level.

The consistency level override for read options against documents and attachments. The valid values are: Strong, Bounded, Session, or Eventual (in order of strongest to weakest). The override must be the same or weaker than the account's configured consistency level.

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

"0"

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. The property name is concatenated onto the object name with a dot to generate the column name.

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 rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Data Type

int

Default Value

-1

Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

MaxThreads

Specifies the maximum number of concurrent requests for Batch CUD (Create, Update, Delete) operations.

Data Type

int

Default Value

200

Remarks

This property should be used in conjunction with the [WriteThroughputBudget](#) connection property. The adapter may execute less parallel requests than the configured [MaxThreads](#) value, since it always aims to not exceed the [WriteThroughputBudget](#) limit. The number of concurrent requests will also depend on the running machine's resources.

Note: This property is applicable only when executing batch CUD operations.

MultiThreadCount

Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.

Data Type

string

Default Value

"5"

Remarks

Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.

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 Azure Cosmos DB.

Data Type

int

Default Value

1000

Remarks

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

Readonly

You can use this property to enforce read-only access to Azure Cosmos DB 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.

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.

SeparatorCharacter

The character or characters used to denote hierarchy.

Data Type

string

Default Value

". "

Remarks

In order to flatten out hierarchical structures, the adapter needs some specifier that states the path to a column through the hierarchy. If this value is "." and a column comes back with the name address.city, this indicates that there is a mapped attribute with a child called city. If your data has columns that already use a single period within the attribute name, set the SeparatorCharacter to a different character or characters.

SetPartitionKeyAsPK

Whether or not to use the collection's Partition Key field as part of composite Primary Key for the corresponding exposed table.

Data Type

bool

Default Value

true

Remarks

By default, this is set to TRUE, and the collection's Partition Key is used as part of the table's composite Primary Key along with the _rid column. If this is set to FALSE, only the _rid column will serve as the Primary Key for the exposed table.

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,Recent"

Remarks

None	Setting <u>TypeDetectionScheme</u> to None will return all columns as a string type. Cannot be combined with other options.
RowScan	Setting <u>TypeDetectionScheme</u> 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 <u>TypeDetectionScheme</u> to Recent will determine whether RowScan is

	executed on the most recent documents in the collection. Can be used with RowScan.
RawValue	Setting <u>TypeDetectionScheme</u> to RawValue will push each document as single aggregate on a column named JsonData, along with its resource identifier on the separate Primary Key column. Cannot be combined with other options.

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].[Entities].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"
```

UseRidAsPk

Set this property to false to switch using the id column as primary key instead the default `_rid`.

Data Type

bool

Default Value

true

Remarks

Since CosmosDB allows you to use both `_rid` and `id` fields as unique values for retrieving resource data, you can set this property to false to switch using the id column as primary key instead the default `_rid`.

WriteThroughputBudget

Defines the Requests Units (RU) budget per Second that the Batch CUD (Create, Update, Delete) operations should not exceed.

Data Type

int

Default Value

1000

Remarks

The adapter will dynamically adjust the maximum number of requests per second depending on the configured RU budget. Although the adapter always aims to not exceed the RU budget, since the requests throttling logic is applied client-side, it may be exceeded by a relatively small amount in a few cases. These cases include Inserting, Updating and Deleting records with highly variable column count and input value length per column.

Note: This property is applicable only when executing batch CUD operations.

TIBCO Product Documentation and Support Services

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

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO 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

- *TIBCO 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.

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

Release Version Support

TDV 8.5 is designated as a Long Term Support (LTS) version. 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

https://docs.tibco.com/pub/tdv/general/LTS/tdv_LTS_releases.htm.

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, visit [TIBCO Community](#).

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

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

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

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 Corporation 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.

This 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 this software version 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. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2002-2023 Cloud Software Group, Inc All Rights Reserved.