Windows PowerShell Get-Help on Cmdlet 'Write-SqlTableData'

PS:\>Get-HELP Write-SqlTableData -Full

NAME

Write-SqlTableData

SYNOPSIS

Writes data to a table of a SQL database.

SYNTAX

Write-SqlTableData [[-ServerInstance] <String[]>] [-AccessToken <PSObject>] [-ConnectionTimeout <Int32>] [-ConnectToDatabase] [-Credential <PSCredential>]

[-DatabaseName <String>] [-Encrypt {Mandatory | Optional | Strict}] [-Force] [-HostNameInCertificate <String>] [-IgnoreProviderContext] -InputData <PSObject>

[-Passthru] [-ProgressAction <ActionPreference>] [-SchemaName <String>] [-SuppressProviderContextWarning] [-TableName <String>] [-Timeout <Int32>]

[-TrustServerCertificate] [<CommonParameters>]

Write-SqlTableData [-InputObject] <Table[]> [-AccessToken <PSObject>] [-Encrypt {Mandatory | Optional | Strict}] [-Force] [-HostNameInCertificate <String>] -InputData

<PSObject> [-Passthru] [-ProgressAction <ActionPreference>] [-Timeout <Int32>] [-TrustServerCertificate] [<CommonParameters>]

Write-SqlTableData [[-Path] <String[]>] [-AccessToken <PSObject>] [-Encrypt {Mandatory | Optional | Strict}] [-Force] [-HostNameInCertificate <String>] -InputData

<PSObject> [-Passthru] [-ProgressAction <ActionPreference>] [-Timeout <Int32>] [-TrustServerCertificate] [<CommonParameters>]

DESCRIPTION

The Write-SqlTableData cmdlet inserts data into a table of a SQL database. This cmdlet accepts the following input types the follow output formats:

- System.Data.DataSet

- System.Data.DataTable

- System.Data.DateRow objects

- Collection of objects

If you provide a DataSet , only the first table in the dataset is written to the database.

You can use this cmdlet with the Windows PowerShell SQL provider.

This cmdlet can infer information such as server, database, schema, and table from its current path.

This cmdlet expects the table to exist. By default, the cmdlet appends data to that table.

If you specify the Force parameter, the cmdlet generate missing objects, which include the database, the table schema, and the table itself. This usage enables quick

transfer of data into a database. The cmdlet infers the schema of the table from the data. The result may not be optimal. For example, strings are mapped to

NVARCHAR(MAX).

PARAMETERS

-AccessToken <PSObject>

The access token used to authenticate to SQL Server, as an alternative to user/password or Windows Authentication.

This can be used, for example, to connect to `SQL Azure DB` and `SQL Azure Managed Instance` using a `Service Principal` or a `Managed Identity`.

The parameter to use can be either a string representing the token or a `PSAccessToken` object as returned by running `Get-AzAccessToken -ResourceUrl

https://database.windows.net`.

> This parameter is new in v22 of the module.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-ConnectionTimeout <Int32>

Specifies the number of seconds to wait for a server connection before a time-out failure. The time-out value must be an integer between 0 and 65534. If 0 is

specified, connection attempts do not time out.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-ConnectToDatabase [<SwitchParameter>]

When establishing the connection, force the cmdlet to use the passed in database (`-DatabaseName`) as the initial catalog. This parameter may be useful to allow

low-privileged users to connect to an existing database for the write operation. Do not use when the database needs to be created.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-Credential <PSCredential>

Specifies a PSCredential object for the connection to SQL Server. To obtain a credential object, use the Get-Credential cmdlet. For more information, type

Get-Help Get-Credential.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-DatabaseName <String>

Specifies the name of the database that contains the table.

The cmdlet supports quoting the value. You do not have to quote or escape special characters.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Encrypt <String>

The encryption type to use when connecting to SQL Server.

This value maps to the `Encrypt` property `SqlConnectionEncryptOption` on the SqlConnection object of the Microsoft.Data.SqlClient driver.

In v22 of the module, the default is `Optional` (for compatibility with v21). In v23+ of the module, the default value will be 'Mandatory', which may create a

breaking change for existing scripts.

> This parameter is new in v22 of the module.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Force [<SwitchParameter>]

Indicates that this cmdlet creates missing SQL Server objects. These include the database, schema, and table. You must have appropriate credentials to create

these objects.

If you do not specify this parameter for missing objects, the cmdlet returns an error.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-HostNameInCertificate <String>

The host name to be used in validating the SQL Server TLS/SSL certificate. You must pass this parameter if your SQL

Server instance is enabled for Force

Encryption and you want to connect to an instance using hostname/shortname. If this parameter is omitted then passing the Fully Qualified Domain Name (FQDN) to

-ServerInstance is necessary to connect to a SQL Server instance enabled for Force Encryption.

> This parameter is new in v22 of the module.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-IgnoreProviderContext [<SwitchParameter>]

Indicates that this cmdlet does not use the current context to override the values of the ServerInstance , DatabaseName , SchemaName , and TableName parameters.

If you do not specify this parameter, the cmdlet ignores the values of these parameters, if possible, in favor of the context in which you run the cmdlet.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-InputData <PSObject>

Specifies the data to write to the database.

Typical input data is a System.Data.DataTable , but you can specify System.Data.DataSet or System.Data.DateRow * objects.

Required? true

Position? named

Default value None

Accept pipeline input? True (ByValue)

Accept wildcard characters? false

-InputObject <Table[]>

Specifies an array of SQL Server Management Objects (SMO) objects that represent the table to which this cmdlet writes.

Required? true

Position? 1

Default value None

Accept pipeline input? True (ByValue)

Accept wildcard characters? false

-Passthru [<SwitchParameter>]

Indicates that this cmdlet returns an SMO.Table object. This object represents the table that includes the added data. You can operate on the table after the

write operation.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-Path <String[]>

Specifies the full path in the context of the SQL Provider of the table where this cmdlet writes data.

Required? false

Position? 1

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-ProgressAction <ActionPreference>

Determines how PowerShell responds to progress updates generated by a script, cmdlet, or provider, such as the progress bars generated by the Write-Progress

cmdlet. The Write-Progress cmdlet creates progress bars that show a command's status.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-SchemaName <String>

Specifies the name of the schema for the table.

If you run this cmdlet in the context of a database or a child item of a database, the cmdlet ignores this parameter value. Specify the IgnoreProviderContext

parameter for the cmdlet to use the value of the SchemaName parameter anyway.

The cmdlet supports quoting the value. You do not have to quote or escape special characters.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-ServerInstance <String[]>

Specifies the name of an instance of SQL Server. For the default instance, specify the computer name. For named instances, use the format

`ComputerName\InstanceName`.

If you run this cmdlet in the context of a database or a child item of a database, the cmdlet ignores this parameter

value. Specify the IgnoreProviderContext

parameter for the cmdlet to use the value of the ServerInstance parameter anyway.

Required? false

Position? 1

Default value None

Accept pipeline input? True (ByValue)

Accept wildcard characters? false

-SuppressProviderContextWarning [<SwitchParameter>]

Indicates that this cmdlet suppresses the warning message that states that the cmdlet uses the provider context.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-TableName <String>

Specifies the name of the table from which this cmdlet reads.

If you run this cmdlet in the context of a database or a child item of a database, the cmdlet ignores this parameter value. Specify the IgnoreProviderContext

parameter for the cmdlet to use the value of the TableName parameter anyway.

The cmdlet supports quoting the value. You do not have to quote or escape special characters.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Timeout <Int32>

Specifies a time-out value, in seconds, for the write operation. If you do not specify a value, the cmdlet uses a default value (typically, 30s). In order to

avoid a timeout, pass 0. The timeout must be an integer value between 0 and 65535.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-TrustServerCertificate [<SwitchParameter>]

Indicates whether the channel will be encrypted while bypassing walking the certificate chain to validate trust.

In v22 of the module, the default is `$true` (for compatibility with v21). In v23+ of the module, the default value will be '$false', which may create a breaking

change for existing scripts.

> This parameter is new in v22 of the module.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug,

ErrorAction, ErrorVariable, WarningAction, WarningVariable,

OutBuffer, PipelineVariable, and OutVariable. For more information, see

about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

INPUTS

System.Management.Automation.PSObject

System.String[]

Microsoft.SqlServer.Management.Smo.Table[]

OUTPUTS

NOTES

--- Example 1: Write information about processes to a table ---

PS C:\> (Get-Process | Select-Object -Property Id,ProcessName,StartTime,UserProcessorTime,WorkingSet,Description) |
Write-SqlTableData -ServerInstance "MyServer\MyInstance" -DatabaseName "MyDatabase" -SchemaName "dbo" -TableName "TaskManagerDump" -Force

This example gets information about processes that run on a system and writes it to a table.

The current cmdlet writes the data to `MyDatabase.dbo.TaskManagerDump` on `MyServer\MyInstance`. Because you specify the Force parameter, if the database, schema, and

table do not exist, this cmdlet creates them.

---------- Example 2: Write data to a table ----------

PS C:\> cd SQLSERVER:\SQL\MyServer\MyInstance\Databases\MyDatabase\Tables
PS SQLSERVER:\SQL\MyServer\MyInstance\Databases\MyDatabase\Tables> $Table = Write-SqlTableData -TableName "KeyValuePairs" -SchemaName "dbo" -InputData @{ cca=10;

cac='Hello'; aac=1.2 } -PassThru

PS SQLSERVER:\SQL\MyServer\MyInstance\Databases\MyDatabase\Tables> Read-SqlTableData -InputObject $Table

WARNING: Using provider context. Server = MyServer\MyInstance, Database = [MyDatabase].

Key Value

--- -----

aac 1.2

cac Hello

cca 10

The first command changes the location to be a location in the SQLSERVER provider. The command prompt reflects the new location. For more information, type Get-Help

about_Providers.

The final command displays the contents of the `$Table` variable by using the Read-SqlTableData cmdlet.

-------- Example 3: Import data from a file to a table --------

PS C:\> ,(Import-Csv -Path ".\a.csv" -Header "Id","Name","Amount") | Write-SqlTableData -ServerInstance "MyServer\MyInstance" -DatabaseName "MyDatabase" -SchemaName

"dbo" -TableName "CSVTable" -Force

PS C:\> Read-SqlTableData -ServerInstance "MyServer\MyInstance" -DatabaseName "MyDatabase" -SchemaName "dbo" -TableName "CSVTable"

Id Name Amount

-- ---- ------

10 AAAA -1.2

11 BBBB 1.2

12 CCCC -1.0

The first command imports the contents of a file by using the Import-Csv cmdlet. The file contains the following content:

10,AAAA,-1.2

11,BBBB,1.2

12,CCCC,-1.0

This example runs completely from the file prompt. It cannot use context information. Therefore, you must specify all relevant parameters.

Note the use of the "," in front of the line: this is to force PowerShell to pass the entire content of the file directly to the Write-SqlTableData cmdlet, which in

turn can do a bulk-insert (which is way more performant than a row by row insert)

Example 4: Retrieve data from one instance and push to table of a database on another instance

```
PS C:\> (Invoke-Sqlcmd -query "SELECT @@SERVERNAME AS 'ServerName', DB_NAME(dbid) AS 'Database',
                        name, CONVERT(BIGINT, size) * 8 AS 'size_in_kb', filename
                        FROM master..sysaltfiles" `
    -ServerInstance MyServer\MyInstance -database master -OutputAs DataTables) |
          Write-SqlTableData -ServerInstance MyServer\MyOtherInstance -Database ServerStats -SchemaName dbo
-TableName DatabasesSizes -Force
```

This example gets information about database files sizes from one instance using the Invoke-SqlCmd cmdlet, and inserts the resulting rows into a database on a

different instance of SQL Server. Note that while this example moves data between instances of SQL Server on the same machine, the instances could be on two

completely different servers.

Example 5: Write data to an existing table of an Azure SQL Database (or any database using SQL Auth)

```
Import-Module SqlServer

# Set your connection string to Azure SQL DB.
# If your server is not in Azure, just tweak the 'Data Source' field to point to your server.
# Warning: putting clear text passwords in your scripts is highly discoraged, so instead
# of using "User ID" and "Password" in the connection string, we prompt for the credentials.
$cred = Get-Credential -Message "Enter your SQL Auth credentials"
$cred.Password.MakeReadOnly()
```

```
# Get access to the SMO Server object.
$srv = Get-SqlInstance -ServerInstance "<your_server_name>.database.windows.net" -Credential $cred

# Get access to table 'MyTable1' on database 'MyDB'.
# Note: both objects are assumed to exists already.
$db = $srv.Databases["MyDB"]
$table = $db.Tables["MyTable1"]

# Write the first 4 integers into the table.
# Note: 'MyTable1' has a column 'Col1' of type 'int'
Write-SqlTableData -InputData (1..4) -InputObject $table

# Now, we read the data back to verify all went ok.
Read-SqlTableData -InputObject $table

# Output:
#
# Col1
# ----
#   1
#   2
#   3
#   4
```

This example shows you how to use the Write-SqlTableData cmdlet with SQL Authentication. Most of the code is just ADO.Net and SMO boilerplate required to create the

necessary object (the `SMO Table` object that represent the target table) needed which is passed to the cmdlet via the `-InputOject` parameter.

Example 6: Write (and read) data to an existing table of an Azure SQL Database using the managed identity of an Azure VM.

```
Import-Module Az.Accounts,SQLServer
```

```
# Change these 3 variables to match your configuration.
    # The example assumes you have a SQL Azure DB with the AdventureWorksLT sample DB on server sql-240627023957.
$Server = 'sql-240627023957.database.windows.net'
$Database = 'AdventureWorksLT'

# Connect to Azure using the system managed identify of the Azure VM this script is running on...
Add-AzAccount -Identity

# ... and fetch an access token to get to the DB.
$AccessToken = (Get-AzAccessToken -ResourceUrl 'https://database.windows.net').Token

# The assumption here is that the Microsoft Entra Admin on the server granted access
# to the managed identity of the VM by running something like:
#   CREATE USER [<Name of the VM>] FROM EXTERNAL PROVIDER
#   ALTER ROLE db_owner ADD MEMBER [<Name of the VM>]
# on the database ($Database).

# Insert a new record into the SalesLT.ProductDescription table
# Note that we are using -ConnectToDatabase to connect directly to the database, since it is unlikely for the
# managed identity of the VM to have access to anything but such database.
    Write-SqlTableData -ServerInstance $Server -Database $Database -AccessToken $AccessToken -SchemaName SalesLT -TableName ProductDescription -InputData @{ Description =
'Hello SQLServer' } -ConnectToDatabase

# Confirm that the new record was successfully added
# Note that -ConnectToDatabase it not necessary in this case, as the connection if done directly to the database.
    Read-SqlTableData -ServerInstance $Server -Database $Database -AccessToken $AccessToken -SchemaName SalesLT -TableName ProductDescription -OrderBy ModifiedDate -TopN
1 -ColumnOrderType DESC

# Output:
```

```
#
# ProductDescriptionID Description     rowguid                              ModifiedDate
# -------------------- -----------     -------                              ------------
#                 2011 Hello SQLServer f5f43821-aacd-4748-9d14-4a525c6a036b 6/30/2024 10:19:26 AM
#
```

RELATED LINKS

Online Version: https://learn.microsoft.com/powershell/module/sqlserver/write-sqltabledata

Read-SqlTableData