

Table of Contents

Backing Up and Restoring Databases and Transaction Logs
Configuring SQL Server in SMO
Creating and Updating Statistics
Creating, Altering, and Removing Database Objects
Creating, Altering, and Removing Databases
Creating, Altering, and Removing Defaults
Creating, Altering, and Removing Foreign Keys
Creating, Altering, and Removing Indexes
Creating, Altering, and Removing Rules
Creating, Altering, and Removing Schemas
Creating, Altering, and Removing Stored Procedures
Creating, Altering, and Removing Tables
Creating, Altering, and Removing Triggers
Creating, Altering, and Removing User-Defined Functions
Creating, Altering, and Removing Views
Granting, Revoking, and Denying Permissions
Implementing Endpoints
Implementing Full-Text Search
Managing Service Broker
Managing Services and Network Settings by Using WMI Provider
Managing Users, Roles, and Logins
Programming Specific Tasks
Scheduling Automatic Administrative Tasks in SQL Server Agent
Scripting
Tracing and Replaying Events
Transferring Data
Using Database Mail
Using Encryption
Using Filegroups and Files to Store Data

Using Linked Servers in SMO

Using Messages

Using Synonyms





Using Table and Index Partitioning

Using User-Defined Tables

Using XML Schemas

Backing Up and Restoring Databases and Transaction Logs

11/16/2017 • 11 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the [Backup](#) class and the [Restore](#) class are utility classes that provide the tools to accomplish the specific tasks of backing up and restoring. A [Backup](#) object represents a specific backup task that is required instead of a Microsoft SQL Server object on the server instance.

If data loss or corruption occurs, the backup must be restored, either fully or partially. Partial restoration uses the [FileGroupCollection](#) collection to segment the data to be restored. If the backup is of a transaction log, the data can be restored up to a particular point in time by using the [ToPointInTime](#) property of the [Restore](#) object. The data can also be validated by using the [SqlVerify](#) method. The recommended backup procedure is to check the integrity of the backup by doing a restore operation and checking the data in the database on a regular basis.

Like the [Backup](#) object, the [Restore](#) object does not need to be created by using a **Create** method because it does not represent any object on the instance of SQL Server. The [Restore](#) object is a set of properties and methods used to restore a database.

Examples

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Backing Up Databases and Transaction Logs in Visual Basic

This code example shows how to back up an existing database to a file, and then how to restore it.

```
Imports Microsoft.SqlServer.Management.Common
Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.VisualBasic.MyServices

Module SMO_VBBackup3

    Sub Main()
        'Connect to the local, default instance of SQL Server.
        Dim srv As Server
        srv = New Server

        'Reference the AdventureWorks2012 database.
        Dim db As Database
        db = srv.Databases("AdventureWorks2012")

        'Store the current recovery model in a variable.
        Dim recoverymod As Integer
        recoverymod = db.DatabaseOptions.RecoveryModel

        'Define a Backup object variable.
        Dim bk As New Backup

        'Specify the type of backup, the description, the name, and the database to be backed up.
        bk.Action = BackupActionType.Database
        bk.BackupSetDescription = "Full backup of AdventureWorks2012"
```

```
bk.BackupSetName = "AdventureWorks 2012 Backup"  
bk.Database = "AdventureWorks2012"
```

'Declare a BackupDeviceItem by supplying the backup device file name in the constructor, and the type of device is a file.

```
Dim bdi As BackupDeviceItem  
bdi = New BackupDeviceItem("Test_Full_Backup1", DeviceType.File)
```

'Add the device to the Backup object.
bk.Devices.Add(bdi)

'Set the Incremental property to False to specify that this is a full database backup.
bk.Incremental = False

'Set the expiration date.
Dim backupdate As New Date
backupdate = New Date(2006, 10, 5)
bk.ExpirationDate = backupdate

'Specify that the log must be truncated after the backup is complete.
bk.LogTruncation = BackupTruncateLogType.Truncate

'Run SqlBackup to perform the full database backup on the instance of SQL Server.
bk.SqlBackup(srv)

'Inform the user that the backup has been completed.
Console.WriteLine("Full Backup complete.")

'Remove the backup device from the Backup object.
bk.Devices.Remove(bdi)

'Make a change to the database, in this case, add a table called test_table.
Dim t As Table
t = New Table(db, "test_table")
Dim c As Column
c = New Column(t, "col", DataType.Int)
t.Columns.Add(c)
t.Create()

'Create another file device for the differential backup and add the Backup object.
Dim bdid As BackupDeviceItem
bdid = New BackupDeviceItem("Test_Differential_Backup1", DeviceType.File)

'Add the device to the Backup object.
bk.Devices.Add(bdid)

'Set the Incremental property to True for a differential backup.
bk.Incremental = True

'Run SqlBackup to perform the incremental database backup on the instance of SQL Server.
bk.SqlBackup(srv)

'Inform the user that the differential backup is complete.
Console.WriteLine("Differential Backup complete.")

'Remove the device from the Backup object.
bk.Devices.Remove(bdid)

'Delete the AdventureWorks2012 database before restoring it.
srv.Databases("AdventureWorks2012").Drop()

'Define a Restore object variable.
Dim rs As Restore
rs = New Restore

'Set the NoRecovery property to true, so the transactions are not recovered.
rs.NoRecovery = True

'Add the device that contains the full database backup to the Restore object

```

'Add the device that contains the full database backup to the restore object.
rs.Devices.Add(bdi)

'Specify the database name.
rs.Database = "AdventureWorks2012"

'Restore the full database backup with no recovery.
rs.SqlRestore(srv)

'Inform the user that the Full Database Restore is complete.
Console.WriteLine("Full Database Restore complete.")

'Remove the device from the Restore object.
rs.Devices.Remove(bdi)

'Set the NoRecovery property to False.
rs.NoRecovery = False

'Add the device that contains the differential backup to the Restore object.
rs.Devices.Add(bdid)

'Restore the differential database backup with recovery.
rs.SqlRestore(srv)

'Inform the user that the differential database restore is complete.
Console.WriteLine("Differential Database Restore complete.")

'Remove the device.
rs.Devices.Remove(bdid)

'Set the database recovery mode back to its original value.
srv.Databases("AdventureWorks2012").DatabaseOptions.RecoveryModel = recoverymod

'Drop the table that was added.
srv.Databases("AdventureWorks2012").Tables("test_table").Drop()
srv.Databases("AdventureWorks2012").Alter()

'Remove the backup files from the hard disk.
My.Computer.FileSystem.DeleteFile("C:\Program Files\Microsoft SQL
Server\MSSQL.12\MSSQL\Backup\Test_Full_Backup1")
My.Computer.FileSystem.DeleteFile("C:\Program Files\Microsoft SQL
Server\MSSQL.12\MSSQL\Backup\Test_Differential_Backup1")
End Sub
End Module

```

Backing Up Databases and Transaction Logs in Visual C#

This code example shows how to back up an existing database to a file, and then how to restore it.

```

using Microsoft.SqlServer.Management.Common;
using Microsoft.SqlServer.Management.Smo;

class A {
    public static void Main() {
        // Connect to the local, default instance of SQL Server.
        Server srv = new Server();
        // Reference the AdventureWorks2012 database.
        Database db = default(Database);
        db = srv.Databases["AdventureWorks2012"];

        // Store the current recovery model in a variable.
        int recoverymod;
        recoverymod = (int)db.DatabaseOptions.RecoveryModel;

        // Define a Backup object variable.
        Backup bk = new Backup();
    }
}

```

```

// Specify the type of backup, the description, the name, and the database to be backed up.
bk.Action = BackupActionType.Database;
bk.BackupSetDescription = "Full backup of Adventureworks2012";
bk.BackupSetName = "AdventureWorks2012 Backup";
bk.Database = "AdventureWorks2012";

// Declare a BackupDeviceItem by supplying the backup device file name in the constructor, and the type
of device is a file.
BackupDeviceItem bdi = default(BackupDeviceItem);
bdi = new BackupDeviceItem("Test_Full_Backup1", DeviceType.File);

// Add the device to the Backup object.
bk.Devices.Add(bdi);
// Set the Incremental property to False to specify that this is a full database backup.
bk.Incremental = false;

// Set the expiration date.
System.DateTime backupdate = new System.DateTime();
backupdate = new System.DateTime(2006, 10, 5);
bk.ExpirationDate = backupdate;

// Specify that the log must be truncated after the backup is complete.
bk.LogTruncation = BackupTruncateLogType.Truncate;

// Run SqlBackup to perform the full database backup on the instance of SQL Server.
bk.SqlBackup(srv);

// Inform the user that the backup has been completed.
System.Console.WriteLine("Full Backup complete.");

// Remove the backup device from the Backup object.
bk.Devices.Remove(bdi);

// Make a change to the database, in this case, add a table called test_table.
Table t = default(Table);
t = new Table(db, "test_table");
Column c = default(Column);
c = new Column(t, "col", DataType.Int);
t.Columns.Add(c);
t.Create();

// Create another file device for the differential backup and add the Backup object.
BackupDeviceItem bdid = default(BackupDeviceItem);
bdid = new BackupDeviceItem("Test_Differential_Backup1", DeviceType.File);

// Add the device to the Backup object.
bk.Devices.Add(bdid);

// Set the Incremental property to True for a differential backup.
bk.Incremental = true;

// Run SqlBackup to perform the incremental database backup on the instance of SQL Server.
bk.SqlBackup(srv);

// Inform the user that the differential backup is complete.
System.Console.WriteLine("Differential Backup complete.");

// Remove the device from the Backup object.
bk.Devices.Remove(bdid);

// Delete the AdventureWorks2012 database before restoring it
// db.Drop();

// Define a Restore object variable.
Restore rs = new Restore();

// Set the NoRecovery property to true, so the transactions are not recovered.
rs.NoRecovery = true;

```

```

// Add the device that contains the full database backup to the Restore object.
rs.Devices.Add(bdi);

// Specify the database name.
rs.Database = "AdventureWorks2012";

// Restore the full database backup with no recovery.
rs.SqlRestore(srv);

// Inform the user that the Full Database Restore is complete.
Console.WriteLine("Full Database Restore complete.");

// reacquire a reference to the database
db = srv.Databases["AdventureWorks2012"];

// Remove the device from the Restore object.
rs.Devices.Remove(bdi);

// Set the NoRecovery property to False.
rs.NoRecovery = false;

// Add the device that contains the differential backup to the Restore object.
rs.Devices.Add(bdid);

// Restore the differential database backup with recovery.
rs.SqlRestore(srv);

// Inform the user that the differential database restore is complete.
System.Console.WriteLine("Differential Database Restore complete.");

// Remove the device.
rs.Devices.Remove(bdid);

// Set the database recovery mode back to its original value.
db.RecoveryModel = (RecoveryModel)recoverymod;

// Drop the table that was added.
db.Tables["test_table"].Drop();
db.Alter();

// Remove the backup files from the hard disk.
// This location is dependent on the installation of SQL Server
System.IO.File.Delete("C:\\Program Files\\Microsoft SQL
Server\\MSSQL12.MSSQLSERVER\\MSSQL\\Backup\\Test_Full_Backup1");
System.IO.File.Delete("C:\\Program Files\\Microsoft SQL
Server\\MSSQL12.MSSQLSERVER\\MSSQL\\Backup\\Test_Differential_Backup1");
}
}

```

Backing Up Databases and Transaction Logs in PowerShell

This code example shows how to back up an existing database to a file, and then how to restore it.

```

#Backing up and restoring a Database from PowerShell

#Connect to the local, default instance of SQL Server.

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#Reference the AdventureWorks database.
$db = $srv.Databases["AdventureWorks"]

#Store the current recovery model in a variable.
$recoverymod = $db.DatabaseOptions.RecoveryModel

```

```

#Create a Backup object
$bk = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Backup

#set to backup the database
$bk.Action = [Microsoft.SqlServer.Management.SMO.BackupActionType]::Database

#Set back up properties
$bk.BackupSetDescription = "Full backup of AdventureWorks"
$bk.BackupSetName = "AdventureWorks Backup"
$bk.Database = "AdventureWorks"

#Declare a BackupDeviceItem by supplying the backup device file name in the constructor,
#and the type of device is a file.
$dt = [Microsoft.SqlServer.Management.SMO.DeviceType]::File
$bdi = New-Object -TypeName Microsoft.SqlServer.Management.SMO.BackupDeviceItem `
-argumentlist "Test_FullBackup1", $dt

#Add the device to the Backup object.
$bk.Devices.Add($bdi)

#Set the Incremental property to False to specify that this is a full database backup.
$bk.Incremental = $false

#Set the expiration date.
$bk.ExpirationDate = get-date "10/05/2006"

#Specify that the log must be truncated after the backup is complete.
$bk.LogTruncation = [Microsoft.SqlServer.Management.SMO.BackupTruncateLogType]::Truncate

#Run SqlBackup to perform the full database backup on the instance of SQL Server.
$bk.SqlBackup($srv)

#Inform the user that the backup has been completed.
"Full Backup complete."

#Remove the backup device from the Backup object.
$bk.Devices.Remove($bdi)

#Make a change to the database, in this case, add a table called test_table.
$t = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Table -argumentlist $db, "test_table"
$type = [Microsoft.SqlServer.Management.SMO.DataType]::int
$c = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $t, "col", $type
$t.Columns.Add($c)
$t.Create()

#Create another file device for the differential backup and add the Backup object.
# $dt is file backup device
$bdid = New-Object -TypeName Microsoft.SqlServer.Management.SMO.BackupDeviceItem `
-argumentlist "Test_DifferentialBackup1", $dt
#Add this device to the backup set
$bk.Devices.Add($bdid)

#Set the Incremental property to True for a differential backup.
$bk.Incremental = $true

#Run SqlBackup to perform the incremental database backup on the instance of SQL Server.
$bk.SqlBackup($srv)

#Inform the user that the differential backup is complete.
"Differential Backup complete."

#Remove the device from the Backup object.
$bk.Devices.Remove($bdid)

#Delete the AdventureWorks database before restoring it.
$db.Drop()

#Define a Restore object variable.

```



```

$rs = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Restore

#Set the NoRecovery property to true, so the transactions are not recovered.
$rs.NoRecovery = $true

#Add the device that contains the full database backup to the Restore object.
$rs.Devices.Add($bdi)

#Specify the database name.
$rs.Database = "AdventureWorks"
#Restore the full database backup with no recovery.
$rs.SqlRestore($srv)

#Inform the user that the Full Database Restore is complete.
"Full Database Restore complete."

#Remove the device from the Restore object.
$rs.Devices.Remove($bdi)

#Set the NoRecovery property to False.
$rs.NoRecovery = $false

#Add the device that contains the differential backup to the Restore object.
$rs.Devices.Add($bdi2)

#Restore the differential database backup with recovery.
$rs.SqlRestore($srv)

#Inform the user that the differential database restore is complete.
"Differential Database Restore complete."

#Remove the device.
$rs.Devices.Remove($bdi2)

#Set the database recovery mode back to its original value.
$db = $srv.Databases["AdventureWorks"]
$db.DatabaseOptions.RecoveryModel = $recoverymod

#Drop the table that was added.
$db.Tables["test_table"].Drop()
$db.Alter()

#Delete the backup files - the exact location depends on your installation
del "C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\Test_FullBackup1"
del "C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\Test_DifferentialBackup1"

```

Running Database Integrity Checks in Visual Basic

SQL Server provides data integrity checking. This code example runs a database consistency type check on the specified database. In this example, [CheckTables](#) is used, but [CheckAllocations](#), [CheckCatalog](#), or [CheckIdentityValues](#) can be used similarly.

NOTE

The [StringCollection](#) object requires a reference to the namespace using the `imports System.Collections.Specialized` statement.

```
Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Common
Imports System.Collections.Specialized
Module S
    Sub Main()
        'Connect to the local, default instance of SQL Server.
        Dim srv As Server
        srv = New Server
        'Reference the AdventureWorks2012 database.
        Dim db As Database
        db = srv.Databases("AdventureWorks2012")
        'Note, to use the StringCollection type the System.Collections.Specialized system namespace must be
        included in the imports statements.
        Dim sc As StringCollection
        'Run the CheckTables method and display the results from the returned StringCollection variable.
        sc = db.CheckTables(RepairType.None)
        Dim c As Integer
        For c = 0 To sc.Count - 1
            Console.WriteLine(sc.Item(c))
        Next
    End Sub
End Module
```

Running Database Integrity Checks in Visual C#

SQL Server provides data integrity checking. This code example runs a database consistency type check on the specified database. In this example, [CheckTables](#) is used, but [CheckAllocations](#), [CheckCatalog](#), or [CheckIdentityValues](#) can be used similarly.

NOTE

The [StringCollection](#) object requires a reference to the namespace using the `imports System.Collections.Specialized` statement.

```
using Microsoft.SqlServer.Management.Common;
using Microsoft.SqlServer.Management.Smo;
using System;

class A {
    public static void Main() {
        // Connect to the local, default instance of SQL Server.
        Server srv = new Server();

        // Reference the AdventureWorks2012 database.
        Database db = srv.Databases["AdventureWorks2012"];

        // Note, to use the StringCollection type the System.Collections.Specialized system namespace must be
        included in the imports statements.
        System.Collections.Specialized.StringCollection sc;

        // Run the CheckTables method and display the results from the returned StringCollection variable.
        sc = db.CheckTables(RepairType.None);

        foreach (string c in sc) {
            Console.WriteLine(c);
        }
    }
}
```

Running Database Integrity Checks in PowerShell

SQL Server provides data integrity checking. This code example runs a database consistency type check on the specified database. In this example, [CheckTables](#) is used, but [CheckAllocations](#), [CheckCatalog](#), or [CheckIdentityValues](#) can be used similarly.

NOTE





The [StringCollection](#) object requires a reference to the namespace using the `imports System.Collections.Specialized` statement.

```
# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

$sc = $db.CheckTables([Microsoft.SqlServer.Management.SMO.RepairType]::None)
foreach ($c in $sc)
{
    $c
}
```

Configuring SQL Server in SMO

11/16/2017 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the [Information](#) object, the [Settings](#) object, the [UserOptions](#) object, and the [Configuration](#) object contain settings and information for the instance of Microsoft SQL Server.

SQL Server has numerous properties that describe the behavior of the installed instance. The properties describe the startup options, the server defaults, files and directories, system and processor information, product and versions, connection information, memory options, language and collation selections, and the authentication mode.

SQL Server Configuration

The [Information](#) object properties contain information about the instance of SQL Server, such as processor and platform.

The [Settings](#) object properties contain information about the instance of SQL Server. The default database file and directory can be modified in addition to the Mail Profile and the Server Account. These properties remain for the duration of the connection.

The [UserOptions](#) object properties contain information about the current connections behavior relating to arithmetic, ANSI standards, and transactions.

There is also a set of configuration options that is represented by the [Configuration](#) object. It contains a set of properties that represent the options that can be modified by the **sp_configure** stored procedure. Options such as **Priority Boost**, **Recovery Interval** and **Network Packet Size** control the performance of the instance of SQL Server. Many of these options can be changed dynamically, but in some cases the value is first configured and then changed when the instance of SQL Server is restarted.

There is a [Configuration](#) object property for every configuration option. Using the [ConfigProperty](#) object you can modify the global configuration setting. Many properties have maximum and minimum values that are also stored as [ConfigProperty](#) properties. These properties require the [Alter](#) method to commit the change to the instance of SQL Server.

All of the configuration options in the [Configuration](#) object must be changed by the system administrator.

Examples

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Modifying SQL Server Configuration Options in Visual Basic

The code example shows how to update a configuration option in Visual Basic .NET. It also retrieves and displays information about maximum and minimum values for the specified configuration option. Finally, the program informs the user if the change has been made dynamically, or if it is stored until the instance of SQL Server is restarted.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Display all the configuration options.
Dim p As ConfigProperty
For Each p In srv.Configuration.Properties
    Console.WriteLine(p.DisplayName)
Next
Console.WriteLine("There are " & srv.Configuration.Properties.Count.ToString & " configuration options.")
'Display the maximum and minimum values for ShowAdvancedOptions.
Dim min As Integer
Dim max As Integer
min = srv.Configuration.ShowAdvancedOptions.Minimum
max = srv.Configuration.ShowAdvancedOptions.Maximum
Console.WriteLine("Minimum and Maximum values are " & min & " and " & max & ".")
'Modify the value of ShowAdvancedOptions and run the Alter method.
srv.Configuration.ShowAdvancedOptions.ConfigValue = 0
srv.Configuration.Alter()
'Display when the change takes place according to the IsDynamic property.
If srv.Configuration.ShowAdvancedOptions.IsDynamic = True Then
    Console.WriteLine("Configuration option has been updated.")
Else
    Console.WriteLine("Configuration option will be updated when SQL Server is restarted.")
End If

```

Modifying SQL Server Settings in Visual Basic

The code example displays information about the instance of SQL Server in [Information](#) and [Settings](#), and modifies settings in [Settings](#) and [UserOptions](#) object properties.

In the example the [UserOptions](#) object and the [Settings](#) object both have an [Alter](#) method. You can run the [Alter](#) methods for these individually.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Display information about the instance of SQL Server in Information and Settings.
Console.WriteLine("OS Version = " & srv.Information.OSVersion)
Console.WriteLine("State = " & srv.Settings.State.ToString)
'Display information specific to the current user in UserOptions.
Console.WriteLine("Quoted Identifier support = " & srv.UserOptions.QuotedIdentifier)
'Modify server settings in Settings.

srv.Settings.LoginMode = ServerLoginMode.Integrated
'Modify settings specific to the current connection in UserOptions.
srv.UserOptions.AbortOnArithmeticErrors = True
'Run the Alter method to make the changes on the instance of SQL Server.
srv.Alter()

```

Modifying SQL Server Settings in Visual C#

The code example displays information about the instance of SQL Server in [Information](#) and [Settings](#), and modifies settings in [Settings](#) and [UserOptions](#) object properties.

In the example the [UserOptions](#) object and the [Settings](#) object both have an [Alter](#) method. You can run the [Alter](#) methods for these individually.

```
//Connect to the local, default instance of SQL Server.
```

```

{
    Server srv = new Server();
    //Display all the configuration options.

    foreach (ConfigProperty p in srv.Configuration.Properties)
    {
        Console.WriteLine(p.DisplayName);
    }
    Console.WriteLine("There are " + srv.Configuration.Properties.Count.ToString() + " configuration
options.");
    //Display the maximum and minimum values for ShowAdvancedOptions.
    int min = 0;
    int max = 0;
    min = srv.Configuration.ShowAdvancedOptions.Minimum;
    max = srv.Configuration.ShowAdvancedOptions.Maximum;
    Console.WriteLine("Minimum and Maximum values are " + min + " and " + max + ".");
    //Modify the value of ShowAdvancedOptions and run the Alter method.
    srv.Configuration.ShowAdvancedOptions.ConfigValue = 0;
    srv.Configuration.Alter();
    //Display when the change takes place according to the IsDynamic property.
    if (srv.Configuration.ShowAdvancedOptions.IsDynamic == true)
    {
        Console.WriteLine("Configuration option has been updated.");
    }
    else
    {
        Console.WriteLine("Configuration option will be updated when SQL Server is restarted.");
    }
}

```

Modifying SQL Server Settings in PowerShell

The code example displays information about the instance of SQL Server in [Information](#) and [Settings](#), and modifies settings in [Settings](#) and [UserOptions](#) object properties.

In the example the [UserOptions](#) object and the [Settings](#) object both have an [Alter](#) method. You can run the [Alter](#) methods for these individually.

```

# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\
$srv = get-item default

#Display information about the instance of SQL Server in Information and Settings.
"OS Version = " + $srv.Information.OSVersion
"State = "+ $srv.Settings.State.ToString()

#Display information specific to the current user in UserOptions.
"Quoted Identifier support = " + $srv.UserOptions.QuotedIdentifier

#Modify server settings in Settings.
$srv.Settings.LoginMode = [Microsoft.SqlServer.Management.SMO.ServerLoginMode]::Integrated

#Modify settings specific to the current connection in UserOptions.
$srv.UserOptions.AbortOnArithmeticErrors = $true

#Run the Alter method to make the changes on the instance of SQL Server.
$srv.Alter()

```

Modifying SQL Server Configuration Options in PowerShell

The code example shows how to update a configuration option in Visual Basic .NET. It also retrieves and displays information about maximum and minimum values for the specified configuration option. Finally, the program

informs the user if the change has been made dynamically, or if it is stored until the instance of SQL Server is restarted.

```
#Get a server object which corresponds to the default instance replace LocalMachine with the physical server
cd \sql\LocalMachine
$svr = get-item default

#enumerate its properties
foreach ($Item in $Svr.Configuration.Properties)
{
    $Item.DisplayName
}

"There are " + $svr.Configuration.Properties.Count.ToString() + " configuration options."





#Display the maximum and minimum values for ShowAdvancedOptions.
$min = $svr.Configuration.ShowAdvancedOptions.Minimum
$max = $svr.Configuration.ShowAdvancedOptions.Maximum
"Minimum and Maximum values are " + $min.ToString() + " and " + $max.ToString() + "."

#Modify the value of ShowAdvancedOptions and run the Alter method.
$svr.Configuration.ShowAdvancedOptions.ConfigValue = 0
$svr.Configuration.Alter()

#Display when the change takes place according to the IsDynamic property.
If ($svr.Configuration.ShowAdvancedOptions.IsDynamic -eq $true)
{
    "Configuration option has been updated."
}
Else
{
    "Configuration option will be updated when SQL Server is restarted."
}
```

Creating and Updating Statistics

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, statistical information about processing queries in the database can be collected by using the [Statistic](#) object.

It is possible to create statistics for any column by using the [Statistic](#) and [StatisticColumn](#) object. The [Update](#) method can be run to update the statistics in the [Statistic](#) object. The results can be viewed in the Query Optimizer.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating and Update Statistics in Visual Basic

This code example creates a new table on an existing database for which the [Statistic](#) object and the [StatisticColumn](#) object are created.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Reference the CreditCard table.
Dim tb As Table
tb = db.Tables("CreditCard", "Sales")
'Define a Statistic object by supplying the parent table and name arguments in the constructor.
Dim stat As Statistic
stat = New Statistic(tb, "Test_Statistics")
'Define a StatisticColumn object variable for the CardType column and add to the Statistic object variable.
Dim statcol As StatisticColumn
statcol = New StatisticColumn(stat, "CardType")
stat.StatisticColumns.Add(statcol)
'Create the statistic counter on the instance of SQL Server.
stat.Create()
```

Creating and Update Statistics in Visual C#

This code example creates a new table on an existing database for which the [Statistic](#) object and the [StatisticColumn](#) object are created.


```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db = default(Database);
    db = srv.Databases["AdventureWorks2012"];
    //Reference the CreditCard table.

    Table tb = db.Tables["CreditCard", "Sales"];
    //Define a Statistic object by supplying the parent table and name
    //arguments in the constructor.
    Statistic stat = default(Statistic);
    stat = new Statistic(tb, "Test_Statistics");
    //Define a StatisticColumn object variable for the CardType column
    //and add to the Statistic object variable.
    StatisticColumn statcol = default(StatisticColumn);
    statcol = new StatisticColumn(stat, "CardType");
    stat.StatisticColumns.Add(statcol);
    //Create the statistic counter on the instance of SQL Server.
    stat.Create();
}

```

Creating and Update Statistics in PowerShell

This code example creates a new table on an existing database for which the [Statistic](#) object and the [StatisticColumn](#) object are created.

```
# Example of implementing a full text search on the default instance.
# Set the path context to the local, default instance of SQL Server and database tables

CD \sql\localhost\default\databases
$db = get-item AdventureWorks2012

CD AdventureWorks2012\tables

#Get a reference to the table
$tb = get-item Production.ProductCategory

# Define a FullTextCatalog object variable by specifying the parent database and name arguments in the
constructor.

$ftc = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextCatalog -argumentlist $db,
"Test_Catalog2"
$ftc.IsDefault = $true

# Create the Full Text Search catalog on the instance of SQL Server.
$ftc.Create()

# Define a FullTextIndex object variable by supplying the parent table argument in the constructor.
$fti = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextIndex -argumentlist $tb

# Define a FullTextIndexColumn object variable by supplying the parent index
# and column name arguments in the constructor.

$ftic = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextIndexColumn -argumentlist $fti, "Name"

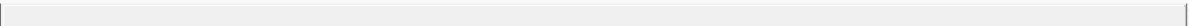
# Add the indexed column to the index.
$fti.IndexedColumns.Add($ftic)

# Set change tracking
$fti.ChangeTracking = [Microsoft.SqlServer.Management.SMO.ChangeTracking]::Automatic

# Specify the unique index on the table that is required by the Full Text Search index.
$fti.UniqueIndexName = "AK_ProductCategory_Name"





# Specify the catalog associated with the index.
$fti.CatalogName = "Test_Catalog2"

# Create the Full Text Search Index
$fti.Create()
```



Creating, Altering, and Removing Database Objects

11/16/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The stages of SMO object creation are as follows:

1. Create an instance of the object.
2. Set the object properties.
3. Create instances of the child objects.
4. Set the child object properties.
5. Create the object.

When instances of SMO objects are created in an SMO application, they do not exist on the instance of SQL Server until the **Create** method is issued. However, it is not necessary to issue a **Create** method for every individual object. If an object has a set of child objects, only the parent object is required to run the **Create** method. For example, the definition of a table requires that it contains at least one column to exist. Also, a column cannot exist in isolation without a table. There is a codependent relationship between the table and its columns.

The **Alter** method lets you make changes to an object. Several changes to an object, such as adding child objects to one of the object's collections or changing a property value, are batched together and run as one. The **Alter** method reduces network traffic and improves overall performance.





The **Drop** statement is used to remove an object and all its codependent child objects that were required to create the object initially.

See Also

[SMO Object Model](#)

Creating, Altering, and Removing Databases

11/16/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, a database is represented by the [Database](#) object.

It is not necessary to create a [Database](#) object to modify or remove it. The database can be referenced by using a collection.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Database in Visual Basic

This code example creates a new database. Files and file groups are automatically created for the database.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Define a Database object variable by supplying the server and the database name arguments in the constructor.
Dim db As Database
db = New Database(srv, "Test_SMO_Database")
'Create the database on the instance of SQL Server.
db.Create()
'Reference the database and display the date when it was created.
db = srv.Databases("Test_SMO_Database")
Console.WriteLine(db.CreateDate)
'Remove the database.
db.Drop()
```

Creating, Altering, and Removing a Database in Visual C#

This code example creates a new database. Files and file groups are automatically created for the database.

```
{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Define a Database object variable by supplying the server and the database name arguments in
the constructor.
    Database db;
    db = new Database(srv, "Test_SMO_Database");
    //Create the database on the instance of SQL Server.
    db.Create();
    //Reference the database and display the date when it was created.
    db = srv.Databases["Test_SMO_Database"];
    Console.WriteLine(db.CreateDate);
    //Remove the database.
    db.Drop();
}
```

Creating, Altering, and Removing a Database in PowerShell

This code example creates a new database. Files and file groups are automatically created for the database.

```
#Get a server object which corresponds to the default instance
cd \sql\localhost\
$srv = get-item default

#Create a new database
$db = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Database -argumentlist $srv, "Test_SMO_Database"
$db.Create()

#Reference the database and display the date when it was created.
$db = $srv.Databases["Test_SMO_Database"]
$db.CreateDate





#Drop the database
$db.Drop()
```

See Also

[Database](#)

Creating, Altering, and Removing Defaults

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SQL Server Management Objects (SMO), the default constraint is represented by the [Default](#) object.

The [TextBody](#) property of the [Default](#) object is used to set the value to be inserted. This can be a constant or a Transact-SQL statement that returns a constant value, such as GETDATE(). The [TextBody](#) property cannot be modified by using the [Alter](#) method. Instead, the [Default](#) object must be dropped and re-created.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Default in Visual Basic

This code example shows how to create one default that is simple text, and another default that is a Transact-SQL statement. The default must be attached to the column by using the [BindToColumn](#) method and detached by using the [UnbindFromColumn](#) method.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a Default object variable by supplying the parent database and the default name
'in the constructor.
Dim def As [Default]
def = New [Default](db, "Test_Default2")
'Set the TextHeader and TextBody properties that define the default.
def.TextHeader = "CREATE DEFAULT [Test_Default2] AS"
def.TextBody = "GetDate()"
'Create the default on the instance of SQL Server.
def.Create()
'Declare a Column object variable and reference a column in the AdventureWorks2012 database.
Dim col As Column
col = db.Tables("SpecialOffer", "Sales").Columns("StartDate")
'Bind the default to the column.
def.BindToColumn("SpecialOffer", "StartDate", "Sales")
'Unbind the default from the column and remove it from the database.
def.UnbindFromColumn("SpecialOffer", "StartDate", "Sales")
def.Drop()
```

Creating, Altering, and Removing a Default in Visual C#

This code example shows how to create one default that is simple text, and another default that is a Transact-SQL statement. The default must be attached to the column by using the [BindToColumn](#) method and detached by using the [UnbindFromColumn](#) method.

```

{

    Server srv = new Server();

    //Reference the AdventureWorks2012 database.
    Database db = srv.Databases["AdventureWorks2012"];

    //Define a Default object variable by supplying the parent database and the default name
    //in the constructor.
    Default def = new Default(db, "Test_Default2");

    //Set the TextHeader and TextBody properties that define the default.
    def.TextHeader = "CREATE DEFAULT [Test_Default2] AS";
    def.TextBody = "GetDate()";

    //Create the default on the instance of SQL Server.
    def.Create();

    //Bind the default to a column in a table in AdventureWorks2012
    def.BindToColumn("SpecialOffer", "StartDate", "Sales");

    //Unbind the default from the column and remove it from the database.
    def.UnbindFromColumn("SpecialOffer", "StartDate", "Sales");
    def.Drop();
}

```

Creating, Altering, and Removing a Default in PowerShell

This code example shows how to create one default that is simple text, and another default that is a Transact-SQL statement. The default must be attached to the column by using the [BindToColumn](#) method and detached by using the [UnbindFromColumn](#) method.

```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

#Define a Default object variable by supplying the parent database and the default name in the constructor.
$def = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Default `
    -argumentlist $db, "Test_Default2"

#Set the TextHeader and TextBody properties that define the default.
$def.TextHeader = "CREATE DEFAULT [Test_Default2] AS"
$def.TextBody = "GetDate()"

#Create the default on the instance of SQL Server.
$def.Create()

#Bind the default to the column.
$def.BindToColumn("SpecialOffer", "StartDate", "Sales")

#Unbind the default from the column and remove it from the database.
$def.UnbindFromColumn("SpecialOffer", "StartDate", "Sales")
$def.Drop()





```

See Also

[Default](#)

Creating, Altering, and Removing Foreign Keys

11/16/2017 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SQL Server Management Objects (SMO), foreign keys are represented by the [ForeignKey](#) object.

To create a foreign key in SMO, you must specify the table on which the foreign key is defined in the constructor of the [ForeignKey](#) object. From the table, you must select at least one column to be the foreign key. To do this, create a [ForeignKeyColumn](#) object variable and specify the name of the column that is the foreign key. Then, specify the referenced table and referenced column. Use the [Add](#) method to add the column to the **Columns** object property.

The columns that represent the foreign key are listed in the **Columns** object property of the [ForeignKey](#) object. The primary key that is referenced by the foreign key is represented by the [ReferencedKey](#) property that is in the table specified in the [ReferencedTable](#) property.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Foreign Key in Visual Basic

This code example shows how to create a foreign key relationship between one or more columns in one table to a primary key column in another table.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Declare a Table object variable and reference the Employee table.
Dim tbe As Table
tbe = db.Tables("Employee", "HumanResources")
'Declare another Table object variable and reference the EmployeeDepartmentHistory table.
Dim tbea As Table
tbea = db.Tables("EmployeeDepartmentHistory", "HumanResources")
'Define a Foreign Key object variable by supplying the EmployeeDepartmentHistory as the parent table and the
foreign key name in the constructor.
Dim fk As ForeignKey
fk = New ForeignKey(tbea, "test_foreignkey")
'Add BusinessEntityID as the foreign key column.
Dim fkc As ForeignKeyColumn
fkc = New ForeignKeyColumn(fk, "BusinessEntityID", "BusinessEntityID")
fk.Columns.Add(fkc)
'Set the referenced table and schema.
fk.ReferencedTable = "Employee"
fk.ReferencedTableSchema = "HumanResources"
'Create the foreign key on the instance of SQL Server.
fk.Create()
```

Creating, Altering, and Removing a Foreign Key in Visual C#

This code example shows how to create a foreign key relationship between one or more columns in one table to a primary key column in another table.

```
{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db;
    db = srv.Databases["AdventureWorks2012"];
    //Declare another Table object variable and reference the EmployeeDepartmentHistory table.
    Table tbea;
    tbea = db.Tables["EmployeeDepartmentHistory", "HumanResources"];
    //Define a Foreign Key object variable by supplying the EmployeeDepartmentHistory as the parent
table and the foreign key name in the constructor.
    ForeignKey fk;
    fk = new ForeignKey(tbea, "test_foreignkey");
    //Add BusinessEntityID as the foreign key column.
    ForeignKeyColumn fkc;
    fkc = new ForeignKeyColumn(fk, "BusinessEntityID", "BusinessEntityID");
    fk.Columns.Add(fkc);
    //Set the referenced table and schema.
    fk.ReferencedTable = "Employee";
    fk.ReferencedTableSchema = "HumanResources";
    //Create the foreign key on the instance of SQL Server.
    fk.Create();
}
```

Creating, Altering, and Removing a Foreign Key in PowerShell

This code example shows how to create a foreign key relationship between one or more columns in one table to a primary key column in another table.

```
# Set the path context to the local, default instance of SQL Server and to the
#database tables in Adventureworks2012
CD \sql\localhost\default\databases\AdventureWorks2012\Tables\

#Get reference to the FK table
$tbea = get-item HumanResources.EmployeeDepartmentHistory

# Define a Foreign Key object variable by supplying the EmployeeDepartmentHistory
# as the parent table and the foreign key name in the constructor.
$fk = New-Object -TypeName Microsoft.SqlServer.Management.SMO.ForeignKey `
-argumentlist $tbea, "test_foreignkey"

#Add BusinessEntityID as the foreign key column.
$fkc = New-Object -TypeName Microsoft.SqlServer.Management.SMO.ForeignKeyColumn `
-argumentlist $fk, "BusinessEntityID", "BusinessEntityID"
$fkc.Columns.Add($fkc)

#Set the referenced table and schema.
$fk.ReferencedTable = "Employee"
$fk.ReferencedTableSchema = "HumanResources"

#Create the foreign key on the instance of SQL Server.
$fk.Create()
```

Sample: Foreign Keys, Primary Keys, and Unique Constraint Columns

This sample demonstrates:

- Finding a foreign key on an existing object.

- How to create a primary key.
- How to create a unique constraint column.

The C# version of this sample:

```
// compile with:
// /r:Microsoft.SqlServer.Smo.dll
// /r:microsoft.sqlserver.management.sdk.sfc.dll
// /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.SqlEnum.dll

using Microsoft.SqlServer.Management.Smo;
using Microsoft.SqlServer.Management.Sdk.Sfc;
using Microsoft.SqlServer.Management.Common;
using System;

public class A {
    public static void Main() {
        Server svr = new Server();
        Database db = new Database(svr, "TESTDB");
        db.Create();

        // PK Table
        Table tab1 = new Table(db, "Table1");

        // Define Columns and add them to the table
        Column col1 = new Column(tab1, "Col1", DataType.Int);

        col1.Nullable = false;
        tab1.Columns.Add(col1);
        Column col2 = new Column(tab1, "Col2", DataType.NVarChar(50));
        tab1.Columns.Add(col2);
        Column col3 = new Column(tab1, "Col3", DataType.DateTime);
        tab1.Columns.Add(col3);

        // Create the ftable
        tab1.Create();

        // Define Index object on the table by supplying the Table1 as the parent table and the primary key name
        // in the constructor.
        Index pk = new Index(tab1, "Table1_PK");
        pk.IndexKeyType = IndexKeyType.DriPrimaryKey;

        // Add Col1 as the Index Column
        IndexedColumn idxCol1 = new IndexedColumn(pk, "Col1");
        pk.IndexedColumns.Add(idxCol1);

        // Create the Primary Key
        pk.Create();

        // Create Unique Index on the table
        Index unique = new Index(tab1, "Table1_Unique");
        unique.IndexKeyType = IndexKeyType.DriUniqueKey;

        // Add Col1 as the Unique Index Column
        IndexedColumn idxCol2 = new IndexedColumn(unique, "Col2");
        unique.IndexedColumns.Add(idxCol2);

        // Create the Unique Index
        unique.Create();

        // Create Table2
        Table tab2 = new Table(db, "Table2");
        Column col21 = new Column(tab2, "Col21", DataType.NChar(20));
        tab2.Columns.Add(col21);
        Column col22 = new Column(tab2, "Col22", DataType.Int);
        tab2.Columns.Add(col22);
    }
}
```

```

        tab2.Columns.Add(col22),
        tab2.Create();

        // Define a Foreign Key object variable by supplying the Table2 as the parent table and the foreign key
name in the constructor.
        ForeignKey fk = new ForeignKey(tab2, "Table2_FK");

        // Add Col22 as the foreign key column.
        ForeignKeyColumn fkc = new ForeignKeyColumn(fk, "Col22", "Col1");
        fk.Columns.Add(fkc);
        fk.ReferencedTable = "Table1";

        // Create the foreign key on the instance of SQL Server.
        fk.Create();

        // Get list of Foreign Keys on Table2
        foreach (ForeignKey f in tab2.ForeignKeys) {
            Console.WriteLine(f.Name + " " + f.ReferencedTable + " " + f.ReferencedKey);
        }

        // Get list of Foreign Keys referencing table1
        foreach (Table tab in db.Tables) {
            if (tab == tab1)
                continue;
            foreach (ForeignKey f in tab.ForeignKeys) {
                if (f.ReferencedTable.Equals(tab1.Name))
                    Console.WriteLine(f.Name + " " + f.Parent.Name);
            }
        }
    }
}

```

The Visual Basic version of the sample:

```

' compile with:
' /r:Microsoft.SqlServer.Smo.dll
' /r:microsoft.sqlserver.management.sdk.sfc.dll
' /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.SqlEnum.dll

Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Sdk.Sfc
Imports Microsoft.SqlServer.Management.Common

Public Class A
    Public Shared Sub Main()
        Dim svr As New Server()
        Dim db As New Database(svr, "TESTDB")
        db.Create()

        ' PK Table
        Dim tab1 As New Table(db, "Table1")

        ' Define Columns and add them to the table
        Dim col1 As New Column(tab1, "Col1", DataType.Int)

        col1.Nullable = False
        tab1.Columns.Add(col1)
        Dim col2 As New Column(tab1, "Col2", DataType.NVarChar(50))
        tab1.Columns.Add(col2)
        Dim col3 As New Column(tab1, "Col3", DataType.DateTime)
        tab1.Columns.Add(col3)

        ' Create the ftable
        tab1.Create()

        ' Define Index object on the table by supplying the Table1 as the parent table and the primary key name
in the constructor

```

in the constructor.

```
Dim pk As New Index(tab1, "Table1_PK")
pk.IndexKeyType = IndexKeyType.DriPrimaryKey

' Add Col1 as the Index Column
Dim idxCol1 As New IndexedColumn(pk, "Col1")
pk.IndexedColumns.Add(idxCol1)

' Create the Primary Key
pk.Create()

' Create Unique Index on the table
Dim unique As New Index(tab1, "Table1_Unique")
unique.IndexKeyType = IndexKeyType.DriUniqueKey

' Add Col1 as the Unique Index Column
Dim idxCol2 As New IndexedColumn(unique, "Col2")
unique.IndexedColumns.Add(idxCol2)

' Create the Unique Index
unique.Create()

' Create Table2
Dim tab2 As New Table(db, "Table2")
Dim col21 As New Column(tab2, "Col21", DataType.NChar(20))
tab2.Columns.Add(col21)
Dim col22 As New Column(tab2, "Col22", DataType.Int)
tab2.Columns.Add(col22)
tab2.Create()

' Define a Foreign Key object variable by supplying the Table2 as the parent table and the foreign key
name in the constructor.
Dim fk As New ForeignKey(tab2, "Table2_FK")

' Add Col22 as the foreign key column.
Dim fkc As New ForeignKeyColumn(fk, "Col22", "Col1")
fk.Columns.Add(fkc)
fk.ReferencedTable = "Table1"





' Create the foreign key on the instance of SQL Server.
fk.Create()

' Get list of Foreign Keys on Table2
For Each f As ForeignKey In tab2.ForeignKeys
    Console.WriteLine((f.Name + " " + f.ReferencedTable & " ") + f.ReferencedKey)
Next

' Get list of Foreign Keys referencing table1
For Each tab As Table In db.Tables
    If (tab.Name.Equals(tab1.Name)) Then
        Continue For
    End If
    For Each f As ForeignKey In tab.ForeignKeys
        If f.ReferencedTable.Equals(tab1.Name) Then
            Console.WriteLine(f.Name + " " + f.Parent.Name)
        End If
    Next
Next
End Sub
End Class
```

Creating, Altering, and Removing Indexes

11/16/2017 • 8 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In the SQL Server Management Objects (SMO) hierarchy, indexes are represented by the [Index](#) object. The indexed columns are represented by a collection of [IndexedColumn](#) objects represented by the [IndexedColumns](#) property.

You can create an index on a XML column by specifying the [IsXmlIndex](#) property of the [Index](#) object.

Examples

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a Non-Clustered, Composite Index in Visual Basic

This code example demonstrates how to create a composite, non-clustered index. For a composite index, add more than one column to the index. Set the [IsClustered](#) property to **False** for a nonclustered index.

```

' /r:Microsoft.SqlServer.Smo.dll
' /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.SqlEnum.dll
' /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

Imports Microsoft.SqlServer.Management.Smo
Public Class A
    Public Shared Sub Main()
        ' Connect to the local, default instance of SQL Server.
        Dim srv As Server
        srv = New Server()

        ' Reference the AdventureWorks2012 database.
        Dim db As Database
        db = srv.Databases("AdventureWorks2012")

        ' Declare a Table object and reference the HumanResources table.
        Dim tb As Table
        tb = db.Tables("Employee", "HumanResources")

        ' Define an Index object variable by providing the parent table and index name in the constructor.
        Dim idx As Index
        idx = New Index(tb, "TestIndex")

        ' Add indexed columns to the index.
        Dim icol1 As IndexedColumn
        icol1 = New IndexedColumn(idx, "BusinessEntityID", True)
        idx.IndexedColumns.Add(icol1)
        Dim icol2 As IndexedColumn
        icol2 = New IndexedColumn(idx, "HireDate", True)
        idx.IndexedColumns.Add(icol2)

        ' Set the index properties.
        idx.IndexKeyType = IndexKeyType.DriUniqueKey
        idx.IsClustered = False
        idx.FillFactor = 50

        ' Create the index on the instance of SQL Server.
        idx.Create()

        ' Modify the page locks property.
        idx.DisallowPageLocks = True

        ' Run the Alter method to make the change on the instance of SQL Server.
        idx.Alter()

        ' Remove the index from the table.
        idx.Drop()
    End Sub
End Class

```

Creating a Non-Clustered, Composite Index in Visual C#

This code example demonstrates how to create a composite, non-clustered index. For a composite index, add more than one column to the index. Set the `IsClustered` property to **False** for a nonclustered index.

```

// /r:Microsoft.SqlServer.Smo.dll
// /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.SqlEnum.dll
// /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

using Microsoft.SqlServer.Management.Smo;

public class A {
    public static void Main() {
        // Connect to the local, default instance of SQL Server.
        Server srv;
        srv = new Server();

        // Reference the AdventureWorks2012 database.
        Database db;
        db = srv.Databases["AdventureWorks2012"];

        // Declare a Table object and reference the HumanResources table.
        Table tb;
        tb = db.Tables["Employee", "HumanResources"];

        // Define an Index object variable by providing the parent table and index name in the constructor.
        Index idx;
        idx = new Index(tb, "TestIndex");

        // Add indexed columns to the index.
        IndexedColumn icol1;
        icol1 = new IndexedColumn(idx, "BusinessEntityID", true);
        idx.IndexedColumns.Add(icol1);
        IndexedColumn icol2;
        icol2 = new IndexedColumn(idx, "HireDate", true);
        idx.IndexedColumns.Add(icol2);

        // Set the index properties.
        idx.IndexKeyType = IndexKeyType.DriUniqueKey;
        idx.IsClustered = false;
        idx.FillFactor = 50;

        // Create the index on the instance of SQL Server.
        idx.Create();

        // Modify the page locks property.
        idx.DisallowPageLocks = true;

        // Run the Alter method to make the change on the instance of SQL Server.
        idx.Alter();

        // Remove the index from the table.
        idx.Drop();
    }
}

```

Creating a Non-Clustered, Composite Index in PowerShell

This code example demonstrates how to create a composite, non-clustered index. For a composite index, add more than one column to the index. Set the `IsClustered` property to **False** for a nonclustered index.

```

# Set the path context to the local, default instance of SQL Server and to the
#database tables in Adventureworks2012
CD \sql\localhost\default\databases\AdventureWorks2012\Tables\

#Get a reference to the table
$tb = get-item HumanResources.Employee

#Define an Index object variable by providing the parent table and index name in the constructor.
$idx = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Index -argumentlist $tb, "TestIndex"

#Add indexed columns to the index.
$ico11 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.IndexedColumn `
-argumentlist $idx, "BusinessEntityId", $true
$idx.IndexedColumns.Add($ico11)

$ico12 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.IndexedColumn `
-argumentlist $idx, "HireDate", $true
$idx.IndexedColumns.Add($ico12)

#Set the index properties.
$idx.IndexKeyType = [Microsoft.SqlServer.Management.SMO.IndexKeyType]::DriUniqueKey
$idx.IsClustered = $false
$idx.FillFactor = 50

#Create the index on the instance of SQL Server.
$idx.Create()

#Modify the page locks property.
$idx.DisallowPageLocks = $true

#Run the Alter method to make the change on the instance of SQL Server.
$idx.Alter()

#Remove the index from the table.
$idx.Drop();

```

Creating an XML Index in Visual Basic

This code example shows how to create an XML index on an XML data type. The XML data type is an XML schema collection called MySampleCollection, which is created in [Using XML Schemas](#). XML indexes have some restrictions, one of which is that it must be created on a table that already has a clustered, primary key.


```

' /r:Microsoft.SqlServer.Smo.dll
' /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.SqlEnum.dll
' /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

Imports Microsoft.SqlServer.Management.Smo

Public Class A
    Public Shared Sub Main()
        ' Connect to the local, default instance of SQL Server.
        Dim srv As Server
        srv = New Server()
        Dim db1 As Database = srv.Databases("TESTDB")
        ' Define a Table object variable and add an XML type column.
        Dim tb As New Table(db1, "XmlTable3")

        Dim mySample As New XmlSchemaCollection(db1, "Sample4", "dbo")
        mySample.Text = "<xsd:schema xmlns:xsd=""http://www.w3.org/2001/XMLSchema"" targetNamespace=""NS2"">
<xsd:element name=""elem1"" type=""xsd:integer""/></xsd:schema>"
        mySample.Create()

        Dim col11 As Column

        ' This sample requires that an XML schema type called MySampleCollection exists on the database.
        col11 = New Column(tb, "XMLValue", DataType.Xml("Sample4"))

        ' Add another integer column that can be made into a unique, primary key.
        tb.Columns.Add(col11)
        Dim col21 As Column
        col21 = New Column(tb, "Number", DataType.Int)
        col21.Nullable = False
        tb.Columns.Add(col21)

        ' Create the table of the instance of SQL Server.
        tb.Create()

        ' Create a unique, clustered, primary key index on the integer column. This is required for an XML
index.
        Dim cp As Index
        cp = New Index(tb, "clusprimindex3")
        cp.IsClustered = True
        cp.IndexKeyType = IndexKeyType.DriPrimaryKey
        Dim cpcol As IndexedColumn
        cpcol = New IndexedColumn(cp, "Number", True)
        cp.IndexedColumns.Add(cpcol)
        cp.Create()

        ' Define and XML Index object variable by supplying the parent table and the XML index name arguments
in the constructor.
        Dim i As Index
        i = New Index(tb, "xmlindex")
        Dim ic As IndexedColumn
        ic = New IndexedColumn(i, "XMLValue", True)
        i.IndexedColumns.Add(ic)

        ' Create the XML index on the instance of SQL Server.
        i.Create()
    End Sub
End Class

```

Creating an XML Index in Visual C#

This code example shows how to create an XML index on an XML data type. The XML data type is an XML schema collection called MySampleCollection, which is created in [Using XML Schemas](#). XML indexes have some restrictions, one of which is that it must be created on a table that already has a clustered, primary key.

```

// /r:Microsoft.SqlServer.Smo.dll
// /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.SqlEnum.dll
// /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

using Microsoft.SqlServer.Management.Smo;

public class A {
    public static void Main() {
        // Connect to the local, default instance of SQL Server.
        Server srv;
        srv = new Server();
        Database db1 = srv.Databases["TESTDB"];
        // Define a Table object variable and add an XML type column.
        Table tb = new Table(db1, "XmlTable3");

        XmlSchemaCollection mySample = new XmlSchemaCollection(db1, "Sample4", "dbo");
        mySample.Text = "<xsd:schema xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" targetNamespace=\"NS2\">
<xsd:element name=\"elem1\" type=\"xsd:integer\"/></xsd:schema>";
        mySample.Create();

        Column col11;

        // This sample requires that an XML schema type called MySampleCollection exists on the database.
        col11 = new Column(tb, "XMLValue", DataType.Xml("Sample4"));

        // Add another integer column that can be made into a unique, primary key.
        tb.Columns.Add(col11);
        Column col21;
        col21 = new Column(tb, "Number", DataType.Int);
        col21.Nullable = false;
        tb.Columns.Add(col21);

        // Create the table of the instance of SQL Server.
        tb.Create();

        // Create a unique, clustered, primary key index on the integer column. This is required for an XML
        index.
        Index cp;
        cp = new Index(tb, "clusprimindex3");
        cp.IsClustered = true;
        cp.IndexKeyType = IndexKeyType.DriPrimaryKey;
        IndexedColumn cpcol;
        cpcol = new IndexedColumn(cp, "Number", true);
        cp.IndexedColumns.Add(cpcol);
        cp.Create();

        // Define and XML Index object variable by supplying the parent table and the XML index name arguments in
        the constructor.
        Index i;
        i = new Index(tb, "xmlindex");
        IndexedColumn ic;
        ic = new IndexedColumn(i, "XMLValue", true);
        i.IndexedColumns.Add(ic);

        // Create the XML index on the instance of SQL Server.
        i.Create();
    }
}

```

Creating an XML Index in PowerShell

This code example shows how to create an XML index on an XML data type. The XML data type is an XML schema collection called MySampleCollection, which is created in [Using XML Schemas](#). XML indexes have some restrictions, one of which is that it must be created on a table that already has a clustered, primary key.

```

# Set the path context to the local, default instance of SQL Server and get a reference to adventureworks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

#Define a Table object variable and add an XML type column.
#This sample requires that an XML schema type called MySampleCollection exists on the database.
#See sample on Creating an XML schema to do this
$tb = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Table -argumentlist $db, "XmlTable"
$Type = [Microsoft.SqlServer.Management.SMO.DataType]::Xml("MySampleCollection")
$col1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"XMLValue", $Type
$tb.Columns.Add($col1)

#Add another integer column that can be made into a unique, primary key.
$Type = [Microsoft.SqlServer.Management.SMO.DataType]::Int
$col2 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"Number", $Type
$col2.Nullable = $false
$tb.Columns.Add($col2)

#Create the table of the instance of SQL Server.
$tb.Create()

#Create a unique, clustered, primary key index on the integer column. This is required for an XML index.
#Define an Index object variable by providing the parent table and index name in the constructor.
$cp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Index -argumentlist $tb, "clusprimindex"
$cp.IsClustered = $true;
$cp.IndexKeyType = [Microsoft.SqlServer.Management.SMO.IndexKeyType]::DriPrimaryKey;

#Create and add an indexed column to the index.
$cpcol = New-Object -TypeName Microsoft.SqlServer.Management.SMO.IndexedColumn `
    -argumentlist $cp, "Number", $true
$cp.IndexedColumns.Add($cpcol)
$cp.Create()

#Define and XML Index object variable by supplying the parent table and
# the XML index name arguments in the constructor.
$i = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Index -argumentlist $tb, "xmlindex"

#Create and add an indexed column to the index.
$ic = New-Object -TypeName Microsoft.SqlServer.Management.SMO.IndexedColumn `
    -argumentlist $i, "XMLValue", $true
$i.IndexedColumns.Add($ic)

#Create the XML index on the instance of SQL Server
$i.Create()





```

See Also

[Index](#)

Creating, Altering, and Removing Rules

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, rules are represented by the [Rule](#) object. The rule is defined by the [TextBody](#) property, which is a text string that contains a condition expression that uses operators or predicates, such as IN, LIKE, or BETWEEN. A rule cannot reference columns or other database objects. Built-in functions that do not reference database objects can be included.

The definition in the [TextBody](#) property must contain a variable that refers to the data value entered. Any name or symbol can be used to represent the value when creating the rule, but the first character must be the @ symbol.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Rule in Visual Basic

This code sample shows how to create a rule, attach it to a column, modify properties of the [Rule](#) object, detach it from the column, and then drop it.

The **Dim** statement for the [Rule](#) object is specified with the full assembly path to avoid ambiguity with a [Rule](#) object in the System.Data assembly.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Declare a Table object variable and reference the Product table.
Dim tb As Table
tb = db.Tables("Product", "Production")
'Define a Rule object variable by supplying the parent database, name and schema in the constructor.
'Note that the full namespace must be given for the Rule type to differentiate it from other Rule types.
Dim ru As Microsoft.SqlServer.Management.Smo.Rule
ru = New Rule(db, "TestRule", "Production")
'Set the TextHeader and TextBody properties to define the rule.
ru.TextHeader = "CREATE RULE [Production].[TestRule] AS"
ru.TextBody = "@value BETWEEN GETDATE() AND DATEADD(year,4,GETDATE())"
'Create the rule on the instance of SQL Server.
ru.Create()
'Bind the rule to a column in the Product table by supplying the table, schema, and
'column as arguments in the BindToColumn method.
ru.BindToColumn("Product", "SellEndDate", "Production")
'Unbind from the column before removing the rule from the database.
ru.UnbindFromColumn("Product", "SellEndDate", "Production")
ru.Drop()
```

Creating, Altering, and Removing a Rule in Visual C#

This code sample shows how to create a rule, attach it to a column, modify properties of the [Rule](#) object, detach it from the column, and then drop it.

The **Dim** statement for the [Rule](#) object is specified with the full assembly path to avoid ambiguity with a [Rule](#) object in the System.Data assembly.

```
{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db;
    db = srv.Databases["AdventureWorks2012"];

    //Define a Rule object variable by supplying the parent database, name and schema in the
    constructor.
    //Note that the full namespace must be given for the Rule type to differentiate it from other Rule
    types.
    Microsoft.SqlServer.Management.Smo.Rule ru;
    ru = new Rule(db, "TestRule", "Production");
    //Set the TextHeader and TextBody properties to define the rule.
    ru.TextHeader = "CREATE RULE [Production].[TestRule] AS";
    ru.TextBody = "@value BETWEEN GETDATE() AND DATEADD(year,4,GETDATE())";
    //Create the rule on the instance of SQL Server.
    ru.Create();
    //Bind the rule to a column in the Product table by supplying the table, schema, and
    //column as arguments in the BindToColumn method.
    ru.BindToColumn("Product", "SellEndDate", "Production");
    //Unbind from the column before removing the rule from the database.
    ru.UnbindFromColumn("Product", "SellEndDate", "Production");
    ru.Drop();
}
```

Creating, Altering, and Removing a Rule in PowerShell

This code sample shows how to create a rule, attach it to a column, modify properties of the [Rule](#) object, detach it from the column, and then drop it.

The **Dim** statement for the [Rule](#) object is specified with the full assembly path to avoid ambiguity with a [Rule](#) object in the System.Data assembly.

```
# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

# Define a Rule object variable by supplying the parent database, name and schema in the constructor.
$ru = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Rule `
-argumentlist $db, "TestRule", "Production"

#Set the TextHeader and TextBody properties to define the rule.
$ru.TextHeader = "CREATE RULE [Production].[TestRule] AS"
$ru.TextBody = "@value BETWEEN GETDATE() AND DATEADD(year,4,GETDATE())"

#Create the rule on the instance of SQL Server.
$ru.Create()

# Bind the rule to a column in the Product table by supplying the table, schema, and
# column as arguments in the BindToColumn method.
$ru.BindToColumn("Product", "SellEndDate", "Production")





#Unbind from the column before removing the rule from the database.
$ru.UnbindFromColumn("Product", "SellEndDate", "Production")
$ru.Drop()
```

See Also

[Rule](#)

Creating, Altering, and Removing Schemas

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The [Schema](#) object represents an ownership context for database object. The [Schemas](#) property of the [Database](#) object represents a collection of [Schema](#) objects.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Schema in Visual Basic

This code example demonstrates how to create a schema and assign it to a database object. The program then grants permission to a user, and then creates a new table in the schema.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a Schema object variable by supplying the parent database and name arguments in the constructor.
Dim sch As Schema
sch = New Schema(db, "MySchema1")
sch.Owner = "dbo"
'Create the schema on the instance of SQL Server.
sch.Create()
'Define an ObjectPermissionSet that contains the Update and Select object permissions.
Dim obperset As ObjectPermissionSet
obperset = New ObjectPermissionSet()
obperset.Add(ObjectPermission.Select)
obperset.Add(ObjectPermission.Update)
'Grant the set of permissions on the schema to the guest account.
sch.Grant(obperset, "guest")
'Define a Table object variable by supplying the parent database, name and schema arguments in the constructor.
Dim tb As Table
tb = New Table(db, "MyTable", "MySchema1")
Dim mycol As Column
mycol = New Column(tb, "Date", DataType.DateTime)
tb.Columns.Add(mycol)
tb.Create()
'Modify the owner of the schema and run the Alter method to make the change on the instance of SQL Server.
sch.Owner = "guest"
sch.Alter()
'Run the Drop method for the table and the schema to remove them.
tb.Drop()
sch.Drop()
```

Creating, Altering, and Removing a Schema in Visual C#

This code example demonstrates how to create a schema and assign it to a database object. The program then

grants permission to a user, and then creates a new table in the schema.

```
{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db = srv.Databases["AdventureWorks2012"];
    //Define a Schema object variable by supplying the parent database and name arguments in the
    constructor.
    Schema sch = new Schema(db, "MySchema1");
    sch.Owner = "dbo";

    //Create the schema on the instance of SQL Server.
    sch.Create();

    //Define an ObjectPermissionSet that contains the Update and Select object permissions.
    ObjectPermissionSet obperset = new ObjectPermissionSet();
    obperset.Add(ObjectPermission.Select);
    obperset.Add(ObjectPermission.Update);

    //Grant the set of permissions on the schema to the guest account.
    sch.Grant(obperset, "guest");

    //Define a Table object variable by supplying the parent database, name and schema arguments in the
    constructor.
    Table tb = new Table(db, "MyTable", "MySchema1");
    Column mycol = new Column(tb, "Date", DataType.DateTime);
    tb.Columns.Add(mycol);
    tb.Create();

    //Modify the owner of the schema and run the Alter method to make the change on the instance of SQL
    Server.
    sch.Owner = "guest";
    sch.Alter();

    //Run the Drop method for the table and the schema to remove them.
    tb.Drop();
    sch.Drop();
}
```

Creating, Altering, and Removing a Schema in PowerShell

This code example demonstrates how to create a schema and assign it to a database object. The program then grants permission to a user, and then creates a new table in the schema.


```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

# Define a schema object variable by supplying the parent database and name arguments in the constructor.
$sch = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Schema `
    -argumentlist $db, "MySchema1"

# Set schema owner
$sch.Owner = "dbo"

# Create the schema on the instance of SQL Server.
$sch.Create()

# Define an ObjectPermissionSet that contains the Update and Select object permissions.
$objperset = New-Object -TypeName Microsoft.SqlServer.Management.SMO.ObjectPermissionSet
$objperset.Add([Microsoft.SqlServer.Management.SMO.ObjectPermission]::Select)
$objperset.Add([Microsoft.SqlServer.Management.SMO.ObjectPermission]::Update)

# Grant the set of permissions on the schema to the guest account.
$sch.Grant($objperset, "guest")

#Create a SMO Table with one column and add it to the database
$tb = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Table -argumentlist $db, "MyTable", "MySchema1"
$Type = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime
$mycol = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"Date", $Type
$tb.Columns.Add($mycol)
$tb.Create()





# Modify the owner of the schema and run the Alter method to make the change on the instance of SQL Server.
$sch.Owner = "guest"
$sch.Alter()

# Run the Drop method for the table and the schema to remove them.
$tb.Drop()
$sch.Drop()

```

Creating, Altering, and Removing Stored Procedures

11/16/2017 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SQL Server Management Objects (SMO), stored procedures are represented by the [StoredProcedure](#) object.

Creating a [StoredProcedure](#) object in SMO requires setting the [TextBody](#) property to the Transact-SQL script that defines the stored procedure. Parameters require the @ prefix and must be created individually by using [StoredProcedureParameter](#) objects and adding to the [StoredProcedureParameter](#) collection of the [StoredProcedure](#) object.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Stored Procedure in Visual Basic

This code example shows how to create a stored procedure for the AdventureWorks2012 database. The example returns the last name of an employee when it is given the employee ID number. The stored procedure requires one input parameter to specify the employee ID number and one output parameter to return the last name of the employee.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a StoredProcedure object variable by supplying the parent database and name arguments in the
constructor.
Dim sp As StoredProcedure
sp = New StoredProcedure(db, "GetLastNameByEmployeeID")
'Set the TextMode property to false and then set the other object properties.
sp.TextMode = False
sp.AnsiNullsStatus = False
sp.QuotedIdentifierStatus = False
'Add two parameters.
Dim param As StoredProcedureParameter
param = New StoredProcedureParameter(sp, "@empval", DataType.Int)
sp.Parameters.Add(param)
Dim param2 As StoredProcedureParameter
param2 = New StoredProcedureParameter(sp, "@retval", DataType.NVarChar(50))
param2.IsOutputParameter = True
sp.Parameters.Add(param2)
'Set the TextBody property to define the stored procedure.
Dim stmt As String
stmt = " SELECT @retval = (SELECT LastName FROM Person.Person AS p JOIN HumanResources.Employee AS e ON
p.BusinessEntityID = e.BusinessEntityID AND e.BusinessEntityID = @empval )"
sp.TextBody = stmt
'Create the stored procedure on the instance of SQL Server.
sp.Create()
'Modify a property and run the Alter method to make the change on the instance of SQL Server.
sp.QuotedIdentifierStatus = True
sp.Alter()
'Remove the stored procedure.
sp.Drop()

```

Creating, Altering, and Removing a Stored Procedure in Visual C#

This code example shows how to create a stored procedure for the AdventureWorks2012 database. The example returns the last name of an employee when it is given the employee ID number (`BusinessEntityID`). The stored procedure requires one input parameter to specify the employee ID number and one output parameter to return the last name of the employee.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db;
    db = srv.Databases["AdventureWorks2012"];
    //Define a StoredProcedure object variable by supplying the parent database and name arguments in
the constructor.
    StoredProcedure sp;
    sp = new StoredProcedure(db, "GetLastNameByBusinessEntityID");
    //Set the TextMode property to false and then set the other object properties.
    sp.TextMode = false;
    sp.AnsiNullsStatus = false;
    sp.QuotedIdentifierStatus = false;
    //Add two parameters.
    StoredProcedureParameter param;
    param = new StoredProcedureParameter(sp, "@empval", DataType.Int);
    sp.Parameters.Add(param);
    StoredProcedureParameter param2;
    param2 = new StoredProcedureParameter(sp, "@retval", DataType.NVarChar(50));
    param2.IsOutputParameter = true;
    sp.Parameters.Add(param2);
    //Set the TextBody property to define the stored procedure.
    string stmt;
    stmt = " SELECT @retval = (SELECT LastName FROM Person.Person,HumanResources.Employee WHERE
Person.Person.BusinessEntityID = HumanResources.Employee.BusinessEntityID AND
HumanResources.Employee.BusinessEntityID = @empval )";
    sp.TextBody = stmt;
    //Create the stored procedure on the instance of SQL Server.
    sp.Create();
    //Modify a property and run the Alter method to make the change on the instance of SQL Server.
    sp.QuotedIdentifierStatus = true;
    sp.Alter();
    //Remove the stored procedure.
    sp.Drop();
}

```

Creating, Altering, and Removing a Stored Procedure in PowerShell

This code example shows how to create a stored procedure for the AdventureWorks2012 database. The example returns the last name of an employee when it is given the employee ID number (`BusinessEntityID`). The stored procedure requires one input parameter to specify the employee ID number and one output parameter to return the last name of the employee.

```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

# Define a StoredProcedure object variable by supplying the parent database and name arguments in the
constructor.
$sp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.StoredProcedure `
-argumentlist $db, "GetLastNameByBusinessEntityID"

#Set the TextMode property to false and then set the other object properties.
$sp.TextMode = $false
$sp.AnsiNullsStatus = $false
$sp.QuotedIdentifierStatus = $false

# Add two parameters
$type = [Microsoft.SqlServer.Management.SMO.Datatype]::Int
$params = New-Object -TypeName Microsoft.SqlServer.Management.SMO.StoredProcedureParameter `
-argumentlist $sp,"@empval",$type
$sp.Parameters.Add($params)

$type = [Microsoft.SqlServer.Management.SMO.DataType]::NVarChar(50)
$params2 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.StoredProcedureParameter `
-argumentlist $sp,"@retval",$type
$params2.IsOutputParameter = $true
$sp.Parameters.Add($params2)

#Set the TextBody property to define the stored procedure.
$sp.TextBody = " SELECT @retval = (SELECT LastName FROM Person.Person,HumanResources.Employee WHERE
Person.Person.BusinessEntityID = HumanResources.Employee.BusinessEntityID AND
HumanResources.Employee.BusinessEntityID = @empval )"

# Create the stored procedure on the instance of SQL Server.
$sp.Create()

# Modify a property and run the Alter method to make the change on the instance of SQL Server.
$sp.QuotedIdentifierStatus = $true
$sp.Alter()

#Remove the stored procedure.
$sp.Drop()





```

See Also

[StoredProcedure](#)

Creating, Altering, and Removing Tables

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SQL Server Management Objects (SMO), tables are represented by the [Table](#) object. In the SMO object hierarchy, the [Table](#) object is below the [Database](#) object.

Example

To use any code example that is provided, you have to choose the programming environment, template, and language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Table in Visual Basic

This code example creates a table that has several columns with different types and purposes. The code also provides examples of how to create an identity field, how to create a primary key, and how to alter table properties.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a Table object variable by supplying the parent database and table name in the constructor.
Dim tb As Table
tb = New Table(db, "Test_Table")
'Add various columns to the table.
Dim col1 As Column
col1 = New Column(tb, "Name", DataType.NChar(50))
col1.Collation = "Latin1_General_CI_AS"
col1.Nullable = True
tb.Columns.Add(col1)
Dim col2 As Column
col2 = New Column(tb, "ID", DataType.Int)
col2.Identity = True
col2.IdentitySeed = 1
col2.IdentityIncrement = 1
tb.Columns.Add(col2)
Dim col3 As Column
col3 = New Column(tb, "Value", DataType.Real)
tb.Columns.Add(col3)
Dim col4 As Column
col4 = New Column(tb, "Date", DataType.DateTime)
col4.Nullable = False
tb.Columns.Add(col4)
'Create the table on the instance of SQL Server.
tb.Create()
'Add another column.
Dim col5 As Column
col5 = New Column(tb, "ExpiryDate", DataType.DateTime)
col5.Nullable = False
tb.Columns.Add(col5)
'Run the Alter method to make the change on the instance of SQL Server.
tb.Alter()
'Remove the table from the database.

tb.Drop()

```

Creating, Altering, and Removing a Table in Visual C#

This code example creates a table that has several columns with different types and purposes. The code also provides examples of how to create an identity field, how to create a primary key, and how to alter table properties.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db;
    db = srv.Databases["AdventureWorks2012"];
    //Define a Table object variable by supplying the parent database and table name in the constructor.
    Table tb;
    tb = new Table(db, "Test_Table");
    //Add various columns to the table.
    Column col1;
    col1 = new Column(tb, "Name", DataType.NChar(50));
    col1.Collation = "Latin1_General_CI_AS";
    col1.Nullable = true;
    tb.Columns.Add(col1);
    Column col2;
    col2 = new Column(tb, "ID", DataType.Int);
    col2.Identity = true;
    col2.IdentitySeed = 1;
    col2.IdentityIncrement = 1;
    tb.Columns.Add(col2);
    Column col3;
    col3 = new Column(tb, "Value", DataType.Real);
    tb.Columns.Add(col3);
    Column col4;
    col4 = new Column(tb, "Date", DataType.DateTime);
    col4.Nullable = false;
    tb.Columns.Add(col4);
    //Create the table on the instance of SQL Server.
    tb.Create();
    //Add another column.
    Column col5;
    col5 = new Column(tb, "ExpiryDate", DataType.DateTime);
    col5.Nullable = false;
    tb.Columns.Add(col5);
    //Run the Alter method to make the change on the instance of SQL Server.
    tb.Alter();
    //Remove the table from the database.
    tb.Drop();
}

```

Creating, Altering, and Removing a Table in PowerShell

This code example creates a table that has several columns with different types and purposes. The code also provides examples of how to create an identity field, how to create a primary key, and how to alter table properties.


```

#Load the assembly containing the objects used in this example
[reflection.assembly]::LoadWithPartialName("Microsoft.SqlServer.Types")

# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\default\Databases\

#And the database object corresponding to AdventureWorks2012.
$db = get-item AdventureWorks2012

#Create a SMO Table
$tb = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Table -argumentlist $db, "Test_Table"

#Add various columns to the table.
$Type = [Microsoft.SqlServer.Management.SMO.DataType]::NChar(50)
$col1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"Name", $Type
$col1.Collation = "Latin1_General_CI_AS"
$col1.Nullable = $true
$tb.Columns.Add($col1)

$Type = [Microsoft.SqlServer.Management.SMO.DataType]::Int
$col2 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"ID", $Type
$col2.Identity = $true
$col2.IdentitySeed = 1
$col2.IdentityIncrement = 1
$tb.Columns.Add($col2)

$Type = [Microsoft.SqlServer.Management.SMO.DataType]::Real
$col3 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"Value", $Type
$tb.Columns.Add($col3)

$Type = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime
$col4 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"Date", $Type
$col4.Nullable = $false
$tb.Columns.Add($col4)

#Create the table
$tb.Create()

$Type = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime
$col5 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column -argumentlist $tb,"ExpiryDate", $Type
$col5.Nullable = $false
$tb.Columns.Add($col5)

#Run the Alter method to make the change on the instance of SQL Server.
$tb.Alter()

#Remove the table from the database.
$tb.Drop()





```

See Also

[Table](#)

Creating, Altering, and Removing Triggers

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, triggers are represented by using the [Trigger](#) object. The Transact-SQL code that runs when the trigger that is fired is set by the [TextBody](#) property of the Trigger object. The type of trigger is set by using other properties of the [Trigger](#) object, such as the [Update](#) property. This is a Boolean property that specifies whether the trigger is fired by an **UPDATE** of records on the parent table.

The [Trigger](#) object represents traditional, data manipulation language (DML) triggers. In SQL Server 2008 and later versions, data definition language (DDL) triggers are also supported. DDL triggers are represented by the [DatabaseDdlTrigger](#) object and the [ServerDdlTrigger](#) object.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a Trigger in Visual Basic

This code example shows how to create and insert an update trigger on an existing table, named `Sales`, in the AdventureWorks2012 database. The trigger sends a reminder message when the table is updated or a new record is inserted.

```
'Connect to the local, default instance of SQL Server.
Dim mysrv As Server
mysrv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim mydb As Database
mydb = mysrv.Databases("AdventureWorks2012")
'Declare a Table object variable and reference the Customer table.
Dim mytab As Table
mytab = mydb.Tables("Customer", "Sales")
'Define a Trigger object variable by supplying the parent table, schema ,and name in the constructor.
Dim tr As Trigger
tr = New Trigger(mytab, "Sales")
'Set TextMode property to False, then set other properties to define the trigger.
tr.TextMode = False
tr.Insert = True
tr.Update = True
tr.InsertOrder = Agent.ActivationOrder.First
Dim stmt As String
stmt = " RAISERROR('Notify Customer Relations',16,10) "
tr.TextBody = stmt
tr.ImplementationType = ImplementationType.TransactSql
'Create the trigger on the instance of SQL Server.
tr.Create()
'Remove the trigger.
tr.Drop()
```

Creating, Altering, and Removing a Trigger in Visual C#

This code example shows how to create and insert an update trigger on an existing table, named `Sales`, in the AdventureWorks2012 database. The trigger sends a reminder message when the table is updated or a new record is inserted.

```
{
    //Connect to the local, default instance of SQL Server.
    Server mysrv;
    mysrv = new Server();
    //Reference the AdventureWorks2012 database.
    Database mydb;
    mydb = mysrv.Databases["AdventureWorks2012"];
    //Declare a Table object variable and reference the Customer table.
    Table mytab;
    mytab = mydb.Tables["Customer", "Sales"];
    //Define a Trigger object variable by supplying the parent table, schema ,and name in the
    constructor.
    Trigger tr;
    tr = new Trigger(mytab, "Sales");
    //Set TextMode property to False, then set other properties to define the trigger.
    tr.TextMode = false;
    tr.Insert = true;
    tr.Update = true;
    tr.InsertOrder = ActivationOrder.First;
    string stmt;
    stmt = " RAISERROR('Notify Customer Relations',16,10) ";
    tr.TextBody = stmt;
    tr.ImplementationType = ImplementationType.TransactSql;
    //Create the trigger on the instance of SQL Server.
    tr.Create();
    //Remove the trigger.
    tr.Drop();
}
```

Creating, Altering, and Removing a Trigger in PowerShell

This code example shows how to create and insert an update trigger on an existing table, named `Sales`, in the AdventureWorks2012 database. The trigger sends a reminder message when the table is updated or a new record is inserted.

```
# Set the path context to the local, default instance of SQL Server and to the
#database tables in Adventureworks2012
CD \sql\localhost\default\databases\AdventureWorks2012\Tables\

#Get reference to the trigger's target table
$mytab = get-item Sales.Customer

# Define a Trigger object variable by supplying the parent table, schema ,and name in the constructor.
$str = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Trigger `
-argumentlist $mytab, "Sales"





# Set TextMode property to False, then set other properties to define the trigger.
$str.TextMode = $false
$str.Insert = $true
$str.Update = $true
$str.InsertOrder = [Microsoft.SqlServer.Management.SMO.Agent.ActivationOrder]::First
$str.TextBody = " RAISERROR('Notify Customer Relations',16,10) "
$str.ImplementationType = [Microsoft.SqlServer.Management.SMO.ImplementationType]::TransactSql

# Create the trigger on the instance of SQL Server.
$str.Create()

#Remove the trigger.
$str.Drop()
```

Creating, Altering, and Removing User-Defined Functions

11/16/2017 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The [UserDefinedFunction](#) object provides functionality that lets users programmatically manage user-defined functions in Microsoft SQL Server. User-defined functions support input and output parameters, and also support direct references to table columns.

SQL Server requires assemblies to be registered within a database before these can be used inside stored procedures, user defined functions, triggers, and user defined data types. SMO supports this feature with the [SqlAssembly](#) object.

The [UserDefinedFunction](#) object references the .NET assembly with the [AssemblyName](#), [ClassName](#), and [MethodName](#) properties.

When the [UserDefinedFunction](#) object references a .NET assembly, you must register the assembly by creating a [SqlAssembly](#) object and adding it to the [SqlAssemblyCollection](#) object, which belongs to the [Database](#) object.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a Scalar User-Defined Function in Visual Basic

This code example shows how to create and remove a scalar user-defined function that has an input [DateTime](#) object parameter and an integer return type in Visual Basic. The user-defined function is created on the AdventureWorks2012 database. The example creates a user-defined function, ISOweek, which takes a date argument and calculates the ISO week number. For this function to calculate correctly, the database DATEFIRST option must be set to 1 before the function is called.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a UserDefinedFunction object variable by supplying the parent database and the name arguments in the
constructor.
Dim udf As UserDefinedFunction
udf = New UserDefinedFunction(db, "IsOWeek")
'Set the TextMode property to false and then set the other properties.
udf.TextMode = False
udf.DataType = DataType.Int
udf.ExecutionContext = ExecutionContext.Caller
udf.FunctionType = UserDefinedFunctionType.Scalar
udf.ImplementationType = ImplementationType.TransactSql
'Add a parameter.
Dim par As UserDefinedFunctionParameter
par = New UserDefinedFunctionParameter(udf, "@DATE", DataType.DateTime)
udf.Parameters.Add(par)
'Set the TextBody property to define the user defined function.
udf.TextBody = "BEGIN DECLARE @ISOweek int SET @ISOweek= DATEPART(wk,@DATE)+1 -
DATEPART(wk,CAST(DATEPART(yy,@DATE) as CHAR(4))+ '0104') IF (@ISOweek=0) SET
@ISOweek=dbo.ISOweek(CAST(DATEPART(yy,@DATE)-1 AS CHAR(4))+ '12' + CAST(24+DATEPART(DAY,@DATE) AS CHAR(2)))+1 IF
((DATEPART(mm,@DATE)=12) AND ((DATEPART(dd,@DATE)-DATEPART(dw,@DATE))>= 28)) SET @ISOweek=1 RETURN(@ISOweek)
END;"
'Create the user defined function on the instance of SQL Server.
udf.Create()
'Remove the user defined function.
udf.Drop()

```

Creating a Scalar User-Defined Function in Visual C#

This code example shows how to create and remove a scalar user-defined function that has an input [DateTime](#) object parameter and an integer return type in Visual C#. The user-defined function is created on the AdventureWorks2012 database. The example creates the user-defined function. `ISOweek`. This function takes a date argument and calculates the ISO week number. For this function to calculate correctly, the database `DATEFIRST` option must be set to `1` before the function is called.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db = srv.Databases["AdventureWorks2012"];

    //Define a UserDefinedFunction object variable by supplying the parent database and the name
arguments in the constructor.
    UserDefinedFunction udf = new UserDefinedFunction(db, "IsOWeek");

    //Set the TextMode property to false and then set the other properties.
    udf.TextMode = false;
    udf.DataType = DataType.Int;
    udf.ExecutionContext = ExecutionContext.Caller;
    udf.FunctionType = UserDefinedFunctionType.Scalar;
    udf.ImplementationType = ImplementationType.TransactSql;

    //Add a parameter.

    UserDefinedFunctionParameter par = new UserDefinedFunctionParameter(udf, "@DATE", DataType.DateTime);
    udf.Parameters.Add(par);

    //Set the TextBody property to define the user-defined function.
    udf.TextBody = "BEGIN DECLARE @ISOweek int SET @ISOweek= DATEPART(wk,@DATE)+1 -
DATEPART(wk,CAST(DATEPART(yy,@DATE) as CHAR(4))+ '0104') IF (@ISOweek=0) SET
@ISOweek=dbo.ISOweek(CAST(DATEPART(yy,@DATE)-1 AS CHAR(4))+ '12' + CAST(24+DATEPART(DAY,@DATE) AS CHAR(2)))+1 IF
((DATEPART(mm,@DATE)=12) AND ((DATEPART(dd,@DATE)-DATEPART(dw,@DATE))>= 28)) SET @ISOweek=1 RETURN(@ISOweek)
END;";

    //Create the user-defined function on the instance of SQL Server.
    udf.Create();

    //Remove the user-defined function.
    udf.Drop();
}

```

Creating a Scalar User-Defined Function in PowerShell

This code example shows how to create and remove a scalar user-defined function that has an input [DateTime](#) object parameter and an integer return type in Visual C#. The user-defined function is created on the AdventureWorks2012 database. The example creates the user-defined function. `ISOweek`. This function takes a date argument and calculates the ISO week number. For this function to calculate correctly, the database `DATEFIRST` option must be set to `1` before the function is called.

```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

# Define a user defined function object variable by supplying the parent database and name arguments in the
constructor.
$udf = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedFunction `
-argumentlist $db, "IsOWeek"

# Set the TextMode property to false and then set the other properties.
$udf.TextMode = $false
$udf.DataType = [Microsoft.SqlServer.Management.SMO.DataType]::Int
$udf.ExecutionContext = [Microsoft.SqlServer.Management.SMO.ExecutionContext]::Caller
$udf.FunctionType = [Microsoft.SqlServer.Management.SMO.UserDefinedFunctionType]::Scalar
$udf.ImplementationType = [Microsoft.SqlServer.Management.SMO.ImplementationType]::TransactSql

# Define a Parameter object variable by supplying the parent function, name and type arguments in the
constructor.
$type = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime
$par = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedFunctionParameter `
-argumentlist $udf, "@DATE",$type

# Add the parameter to the function
$udf.Parameters.Add($par)

#Set the TextBody property to define the user-defined function.
$udf.TextBody = "BEGIN DECLARE @ISOweek int SET @ISOweek= DATEPART(wk,@DATE)+1 -
DATEPART(wk,CAST(DATEPART(yy,@DATE) as CHAR(4))+ '0104') IF (@ISOweek=0) SET
@ISOweek=dbo.ISOweek(CAST(DATEPART(yy,@DATE)-1 AS CHAR(4))+ '12' + CAST(24+DATEPART(DAY,@DATE) AS CHAR(2)))+1 IF
((DATEPART(mm,@DATE)=12) AND ((DATEPART(dd,@DATE)-DATEPART(dw,@DATE))>= 28)) SET @ISOweek=1 RETURN(@ISOweek)
END;"

# Create the user-defined function on the instance of SQL Server.
$udf.Create()

# Remove the user-defined function.
$udf.Drop()





```

See Also

[UserDefinedFunction](#)

Creating, Altering, and Removing Views

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SQL Server Management Objects (SMO), SQL Server views are represented by the [View](#) object.

The [TextBody](#) property of the [View](#) object defines the view. It is the equivalent of the Transact-SQL SELECT statement for creating a view.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating, Altering, and Removing a View in Visual Basic

This code sample shows how to create a view of two tables by using an inner join. The view is created by using text mode, so the [TextHeader](#) property must be set.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a View object variable by supplying the parent database, view name and schema in the constructor.
Dim myview As View
myview = New View(db, "Test_View", "Sales")
'Set the TextHeader and TextBody property to define the view.
myview.TextHeader = "CREATE VIEW [Sales].[Test_View] AS"
myview.TextBody = "SELECT h.SalesOrderID, d.OrderQty FROM Sales.SalesOrderHeader AS h INNER JOIN
Sales.SalesOrderDetail AS d ON h.SalesOrderID = d.SalesOrderID"
'Create the view on the instance of SQL Server.
myview.Create()
'Remove the view.
myview.Drop()
```

Creating, Altering, and Removing a View in Visual C#

This code sample shows how to create a view of two tables by using an inner join. The view is created by using text mode, so the [TextHeader](#) property must be set.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db;
    db = srv.Databases["AdventureWorks2012"];
    //Define a View object variable by supplying the parent database, view name and schema in the
    constructor.
    View myview;
    myview = new View(db, "Test_View", "Sales");
    //Set the TextHeader and TextBody property to define the view.
    myview.TextHeader = "CREATE VIEW [Sales].[Test_View] AS";
    myview.TextBody = "SELECT h.SalesOrderID, d.OrderQty FROM Sales.SalesOrderHeader AS h INNER JOIN
    Sales.SalesOrderDetail AS d ON h.SalesOrderID = d.SalesOrderID";
    //Create the view on the instance of SQL Server.
    myview.Create();
    //Remove the view.
    myview.Drop();
}

```

Creating, Altering, and Removing a View in PowerShell

This code sample shows how to create a view of two tables by using an inner join. The view is created by using text mode, so the [TextHeader](#) property must be set.

```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

# Define a View object variable by supplying the parent database, view name and schema in the constructor.
$myview = New-Object -TypeName Microsoft.SqlServer.Management.SMO.View `
    -argumentlist $db, "Test_View", "Sales"

# Set the TextHeader and TextBody property to define the view.
$myview.TextHeader = "CREATE VIEW [Sales].[Test_View] AS"
$myview.TextBody ="SELECT h.SalesOrderID, d.OrderQty FROM Sales.SalesOrderHeader AS h INNER JOIN
Sales.SalesOrderDetail AS d ON h.SalesOrderID = d.SalesOrderID"

# Create the view on the instance of SQL Server.
$myview.Create()

# Remove the view.
$myview.Drop();





```

See Also

[View](#)

Granting, Revoking, and Denying Permissions

11/16/2017 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The [ServerPermission](#) object is used to assign a set of permissions or an individual server permission to the [ServerPermissionSet](#) object. For server level permissions, the grantee refers to a login. Logons authenticated by Windows are listed as Windows user names. When this code sample runs, it revokes the permission from the grantee and verifies it has been removed with the [EnumServerPermissions](#) method.

Database permissions and database object permissions can be assigned similarly by using the [DatabasePermissionSet](#) object and the [ObjectPermissionSet](#) object.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Granting Server Permissions in Visual Basic

This code example grants the Create Endpoint and Alter Any Endpoint permissions to the specified login, and then enumerates and displays the permissions. One of the permissions is revoked, and then the permissions are enumerated again. This example assumes that the specified login has the specified permissions to start with.

```
' compile with: /r:Microsoft.SqlServer.Smo.dll /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll /r:Microsoft.SqlServer.SqlEnum.dll
Imports Microsoft.SqlServer.Management.Smo

Public Class A
    Public Shared Sub Main()
        Dim svr As New Server()

        ' Creating the logins (Grantee)
        Dim vGrantee As [String] = "Grantee1"
        Dim login As New Login(svr, vGrantee)
        login.LoginType = LoginType.SqlLogin
        login.Create("password@1")

        Dim vGrantee2 As [String] = "Grantee2"
        Dim login2 As New Login(svr, vGrantee2)
        login2.LoginType = LoginType.SqlLogin
        login2.Create("password@2")

        ' Define a ServerPermissionSet that contains permission to Create Endpoint and Alter Any Endpoint.
        Dim sps As New ServerPermissionSet(ServerPermission.CreateEndpoint)
        sps.Add(ServerPermission.AlterAnyEndpoint)

        ' Grant Create Endpoint and Alter Any Endpoint permissions to Grantee
        svr.Grant(sps, vGrantee)
        svr.Grant(sps, vGrantee2)

        ' Enumerate and display the server permissions in the set for the grantee specified in the vGrantee
        string variable.
        Dim spis As ServerPermissionInfo() = svr.EnumServerPermissions(vGrantee, sps)
        'enumerates all server permissions for the Grantee from the specified permission set
        Console.WriteLine("====Before revoke=====")
```

```

For Each spi As ServerPermissionInfo In spis
    Console.WriteLine(spi.Grantee + " has " & spi.PermissionType.ToString() & " permission.")
Next
Console.WriteLine(" ")

' Revoke the create endpoint permission from the grantee.
svr.Revoke(New ServerPermissionSet(ServerPermission.CreateEndpoint), vGrantee)

' Enumerate and display the server permissions in the set for the grantee specified in the vGrantee
string variable.
spis = svr.EnumServerPermissions(vGrantee, spis)

Console.WriteLine("==After revoke=====")
For Each spi As ServerPermissionInfo In spis
    Console.WriteLine(spi.Grantee + " has " & spi.PermissionType.ToString() & " permission.")
Next
Console.WriteLine(" ")

' Grant the Create Server Role permission to the grantee.
svr.Grant(New ServerPermissionSet(ServerPermission.ViewAnyDatabase), vGrantee)
' Enumerate and display the server permissions for the grantee specified in the vGrantee string variable.

' enumerates all server permissions for the Grantee
spis = svr.EnumServerPermissions(vGrantee)

Console.WriteLine("==After grant=====")

For Each spi As ServerPermissionInfo In spis
    Console.WriteLine(spi.Grantee + " has " & spi.PermissionType.ToString() & " permission.")
Next
Console.WriteLine("")

' Enumerate and display the server permissions in the set for all logins.
spis = svr.EnumServerPermissions(sps)
'enumerates all server permissions in the set for all logins
Console.WriteLine("==After grant=====")

For Each spi As ServerPermissionInfo In spis
    Console.WriteLine(spi.Grantee + " has " & spi.PermissionType.ToString() & " permission.")
Next
Console.WriteLine("")
End Sub
End Class

```

Granting Server Permissions in Visual C#

This code example grants the Create Endpoint and Alter Any Endpoint permissions to the specified login, and then enumerates and displays the permissions. One of the permissions is revoked, and then the permissions are enumerated again. This example assumes that the specified login has the specified permissions to start with.

```

// compile with: /r:Microsoft.SqlServer.Smo.dll /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll /r:Microsoft.SqlServer.SqlEnum.dll
using System;
using Microsoft.SqlServer.Management.Smo;

public class A {
    public static void Main() {
        Server svr = new Server();

        // Creating the logins (Grantee)
        String vGrantee = "Grantee1";
        Login login = new Login(svr, vGrantee);
        login.LoginType = LoginType.SqlLogin;
        login.Create("password@1");
    }
}

```

```

String vGrantee2 = "Grantee2";
Login login2 = new Login(svr, vGrantee2);
login2.LoginType = LoginType.SqlLogin;
login2.Create("password@2");

// Define a ServerPermissionSet that contains permission to Create Endpoint and Alter Any Endpoint.
ServerPermissionSet sps = new ServerPermissionSet(ServerPermission.CreateEndpoint);
sps.Add(ServerPermission.AlterAnyEndpoint);

// Grant Create Endpoint and Alter Any Endpoint permissions to Grantee
svr.Grant(sps, vGrantee);
svr.Grant(sps, vGrantee2);

// Enumerate and display the server permissions in the set for the grantee specified in the vGrantee
string variable.
ServerPermissionInfo[] spis = svr.EnumServerPermissions(vGrantee, sps); //enumerates all server
permissions for the Grantee from the specified permission set

Console.WriteLine("====Before revoke=====");
foreach (ServerPermissionInfo spi in spis) {
    Console.WriteLine(spi.Grantee + " has " + spi.PermissionType.ToString() + " permission.");
}
Console.WriteLine(" ");

// Revoke the create endpoint permission from the grantee.
svr.Revoke(new ServerPermissionSet(ServerPermission.CreateEndpoint), vGrantee);

// Enumerate and display the server permissions in the set for the grantee specified in the vGrantee
string variable.
spis = svr.EnumServerPermissions(vGrantee, sps);

Console.WriteLine("==After revoke=====");
foreach (ServerPermissionInfo spi in spis) {
    Console.WriteLine(spi.Grantee + " has " + spi.PermissionType.ToString() + " permission.");
}
Console.WriteLine(" ");

// Grant the Create Server Role permission to the grantee.
svr.Grant(new ServerPermissionSet(ServerPermission.ViewAnyDatabase), vGrantee);
// Enumerate and display the server permissions for the grantee specified in the vGrantee string
variable.

// enumerates all server permissions for the Grantee
spis = svr.EnumServerPermissions(vGrantee);

Console.WriteLine("==After grant=====");

foreach (ServerPermissionInfo spi in spis) {
    Console.WriteLine(spi.Grantee + " has " + spi.PermissionType.ToString() + " permission.");
}
Console.WriteLine("");

// Enumerate and display the server permissions in the set for all logins.
spis = svr.EnumServerPermissions(sps); //enumerates all server permissions in the set for all logins

Console.WriteLine("==After grant=====");

foreach (ServerPermissionInfo spi in spis) {
    Console.WriteLine(spi.Grantee + " has " + spi.PermissionType.ToString() + " permission.");
}
Console.WriteLine("");
}
}

```

Granting Server Permissions in PowerShell

This code example grants the Create Endpoint and Alter Any Endpoint permissions to the specified login, and then enumerates and displays the permissions. One of the permissions is revoked, and then the permissions are enumerated again. This example assumes that the specified login has the specified permissions to start with.

```
# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\
$srv = get-item default

#The subject login:
# "Place Login Name here - has permission to Create Endpoints"
$vGrantee = "LoginName"

#This sample assumes that the grantee already has permission to Create Endpoints.

$sps = New-Object -TypeName Microsoft.SqlServer.Management.SMO.ServerPermissionSet

$sps.CreateEndpoint = $true
$sps.AlterAnyEndpoint = $true

#This sample assumes that the grantee already has permission to Create Endpoints.

#Enumerate and display the server permissions in the set for the grantee specified
# in the vGrantee string variable.
$spis = $srv.EnumServerPermissions($vGrantee)

"===Before revoke======"
foreach ( $spi in $spis)
{
    $spi.Grantee + " has " + $spi.PermissionType + " permission."
}
""

#remove perission to create an endpoint
$sps.CreateEndpoint = $false
$srv.Revoke($sps, $vGrantee)

#Enumerate and display the server permissions in the set for the grantee specified
# in the vGrantee string variable.
$spis = $srv.EnumServerPermissions($vGrantee)

"===After revoke======"
foreach ( $spi in $spis)
{
    $spi.Grantee + " has " + $spi.PermissionType + " permission."
}
""

#Grant the revoked permissions back
$sps.CreateEndpoint = $true
$sps.AlterAnyEndpoint = $true
$srv.Grant($sps, $vGrantee)

#Enumerate and display the server permissions in the set for the grantee specified
# in the vGrantee string variable.
$spis = $srv.EnumServerPermissions($vGrantee)





"===After grant======"
foreach ( $spi in $spis)
{
    $spi.Grantee + " has " + $spi.PermissionType + " permission."
}
}
```

See Also

[Permissions Hierarchy \(Database Engine\)](#)

Implementing Endpoints

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

An endpoint is a service that can listen natively for requests. SMO supports various types of endpoints by using the [Endpoint](#) object. You can create an endpoint service that handles a specific type of payload, which uses a specific protocol, by creating an instance of an [Endpoint](#) object and setting its properties.

The [EndpointType](#) property of the [Endpoint](#) object can be used to specify one of the following payload types:

- Database mirroring
- SOAP (support for SOAP endpoints is present in SQL Server 2008 R2 and earlier SQL Server versions)
- Service Broker
- Transact-SQL

Also, the [ProtocolType](#) property can be used to specify the following two supported protocols:

- HTTP protocol
- TCP protocol

Having specified the type of payload, the actual payload can be set by using the [Payload](#) object property. The [Payload](#) object property provides a reference to a payload object of the specified type, for which the properties can be modified.

For the [DatabaseMirroringPayload](#) object, you must specify the mirroring role and whether encryption is enabled. The [ServiceBrokerPayload](#) object requires information about message forwarding, maximum number of connections allowed and the authentication mode. The [SoapPayloadMethod](#) object requires various properties to be set including the [Add](#) object property that specifies the SOAP payload methods available to clients (stored procedures and user-defined functions).

Similarly, the actual protocol can be set by using the [Protocol](#) object property that references a protocol object of the type specified by [ProtocolType](#) property. The [HttpProtocol](#) object requires a list of restricted IP addresses, and port, website, and authentication information. The [TcpProtocol](#) object also requires a list of restricted IP addresses and port information.

When the endpoint has been created and fully defined, access can be granted to, revoked from, and denied to database users, groups, roles, and logons.

Example

For the following code example, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a Database Mirroring Endpoint Service in Visual Basic

The code example demonstrates how to create a Database Mirroring endpoint in SMO. This is necessary before you create a database mirror. Use the [IsMirroringEnabled](#) and other properties on the [Database](#) object to create a database mirror.

```

'Set up a database mirroring endpoint on the server before setting up a database mirror.
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Define an Endpoint object variable for database mirroring.
Dim ep As Endpoint
ep = New Endpoint(srv, "Mirroring_Endpoint")
ep.ProtocolType = ProtocolType.Tcp
ep.EndpointType = EndpointType.DatabaseMirroring
'Specify the protocol ports.
ep.Protocol.Http.SslPort = 5024
ep.Protocol.Tcp.ListenerPort = 6666
'Specify the role of the payload.
ep.Payload.DatabaseMirroring.ServerMirroringRole = ServerMirroringRole.All
'Create the endpoint on the instance of SQL Server.
ep.Create()
'Start the endpoint.
ep.Start()
Console.WriteLine(ep.EndpointState)

```

Creating a Database Mirroring Endpoint Service in Visual C#

The code example demonstrates how to create a Database Mirroring endpoint in SMO. This is necessary before you create a database mirror. Use the [IsMirroringEnabled](#) and other properties on the [Database](#) object to create a database mirror.

```

{
    //Set up a database mirroring endpoint on the server before
    //setting up a database mirror.
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();
    //Define an Endpoint object variable for database mirroring.
    Endpoint ep = default(Endpoint);
    ep = new Endpoint(srv, "Mirroring_Endpoint");
    ep.ProtocolType = ProtocolType.Tcp;
    ep.EndpointType = EndpointType.DatabaseMirroring;
    //Specify the protocol ports.
    ep.Protocol.Http.SslPort = 5024;
    ep.Protocol.Tcp.ListenerPort = 6666;
    //Specify the role of the payload.
    ep.Payload.DatabaseMirroring.ServerMirroringRole = ServerMirroringRole.All;
    //Create the endpoint on the instance of SQL Server.
    ep.Create();
    //Start the endpoint.
    ep.Start();
    Console.WriteLine(ep.EndpointState);
}

```

Creating a Database Mirroring Endpoint Service in PowerShell

The code example demonstrates how to create a Database Mirroring endpoint in SMO. This is necessary before you create a database mirror. Use the [IsMirroringEnabled](#) and other properties on the [Database](#) object to create a database mirror.


```
# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\
$srv = get-item default

#Get a new endpoint to configure and add
$ep = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Endpoint -argumentlist $srv,"Mirroring_Endpoint"

#Set some properties
$ep.ProtocolType = [Microsoft.SqlServer.Management.SMO.ProtocolType]::Tcp
$ep.EndpointType = [Microsoft.SqlServer.Management.SMO.EndpointType]::DatabaseMirroring
$ep.Protocol.Http.SslPort = 5024
$ep.Protocol.Tcp.ListenerPort = 6666 #inline comment
$ep.Payload.DatabaseMirroring.ServerMirroringRole =
[Microsoft.SqlServer.Management.SMO.ServerMirroringRole]::All

# Create the endpoint on the instance
$ep.Create()

# Start the endpoint
$ep.Start()





# Report its state
$ep.EndpointState;
```

See Also

[The Database Mirroring Endpoint \(SQL Server\)](#)

Implementing Full-Text Search

11/16/2017 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Full-text search is available per instance of SQL Server and is represented in SMO by the [FullTextService](#) object. The [FullTextService](#) object resides under the **Server** object. It is used to manage the configuration options for Microsoft Full Text Search service. The [FullTextCatalogCollection](#) object belongs to the [Database](#) object and it is a collection of [FullTextCatalog](#) objects that represent full-text catalogs defined for the database. You can only have one full-text index defined for each table, unlike normal indexes. This is represented by a [FullTextIndexColumn](#) object in the [Table](#) object.

To create a full-text search service, you must have a full-text catalog defined on the database and a full-text search index defined on one of the tables in the database.

First, create a full-text catalog on the database by calling the [FullTextCatalog](#) constructor and specifying the catalog name. Then, create the full-text index by calling the constructor and specifying the table on which it is to be created. You can then add index columns for the full-text index, by using the [FullTextIndexColumn](#) object and providing the name of the column within the table. Then, set the [CatalogName](#) property to the catalog you have created. Finally, call the [Create](#) method and create the full-text index on the instance of SQL Server.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a Full-Text Search Service in Visual Basic

This code example creates a full-text search catalog for the `ProductCategory` table in the AdventureWorks2012 sample database. It then creates a full-text search index on the Name column in the `ProductCategory` table. The full-text search index requires that there is a unique index already defined on the column.

```

' compile with:
' /r:Microsoft.SqlServer.SqlEnum.dll
' /r:Microsoft.SqlServer.Smo.dll
' /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Sdk.Sfc
Imports Microsoft.SqlServer.Management.Common

Public Class A
    Public Shared Sub Main()
        ' Connect to the local, default instance of SQL Server.
        Dim srv As Server = Nothing
        srv = New Server()

        ' Reference the AdventureWorks database.
        Dim db As Database = Nothing
        db = srv.Databases("AdventureWorks")

        ' Reference the ProductCategory table.
        Dim tb As Table = Nothing
        tb = db.Tables("ProductCategory", "Production")

        ' Define a FullTextCatalog object variable by specifying the parent database and name arguments in the
        constructor.
        Dim ftc As FullTextCatalog = Nothing
        ftc = New FullTextCatalog(db, "Test_Catalog")
        ftc.IsDefault = True

        ' Create the Full-Text Search catalog on the instance of SQL Server.
        ftc.Create()

        ' Define a FullTextIndex object variable by supplying the parent table argument in the constructor.
        Dim fti As FullTextIndex = Nothing
        fti = New FullTextIndex(tb)

        ' Define a FullTextIndexColumn object variable by supplying the parent index and column name arguments
        in the constructor.
        Dim ftic As FullTextIndexColumn = Nothing
        ftic = New FullTextIndexColumn(fti, "Name")

        ' Add the indexed column to the index.
        fti.IndexedColumns.Add(ftic)
        fti.ChangeTracking = ChangeTracking.Automatic

        ' Specify the unique index on the table that is required by the Full Text Search index.
        fti.UniqueIndexName = "AK_ProductCategory_Name"

        ' Specify the catalog associated with the index.
        fti.CatalogName = "Test_Catalog"

        ' Create the Full Text Search index on the instance of SQL Server.
        fti.Create()
    End Sub
End Class

```

Creating a Full-Text Search Service in Visual C#

This code example creates a full-text search catalog for the `ProductCategory` table in the AdventureWorks2012 sample database. It then creates a full-text search index on the Name column in the `ProductCategory` table. The full-text search index requires that there is a unique index already defined on the column.

```

// compile with:
// /r:Microsoft.SqlServer.SqlEnum.dll
// /r:Microsoft.SqlServer.Smo.dll
// /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

using Microsoft.SqlServer.Management.Smo;
using Microsoft.SqlServer.Management.Sdk.Sfc;
using Microsoft.SqlServer.Management.Common;

public class A {
    public static void Main() {
        // Connect to the local, default instance of SQL Server.
        Server srv = default(Server);
        srv = new Server();

        // Reference the AdventureWorks database.
        Database db = default(Database);
        db = srv.Databases ["AdventureWorks"];

        // Reference the ProductCategory table.
        Table tb = default(Table);
        tb = db.Tables["ProductCategory", "Production"];

        // Define a FullTextCatalog object variable by specifying the parent database and name arguments in the
        constructor.
        FullTextCatalog ftc = default(FullTextCatalog);
        ftc = new FullTextCatalog(db, "Test_Catalog");
        ftc.IsDefault = true;

        // Create the Full-Text Search catalog on the instance of SQL Server.
        ftc.Create();

        // Define a FullTextIndex object variable by supplying the parent table argument in the constructor.
        FullTextIndex fti = default(FullTextIndex);
        fti = new FullTextIndex(tb);

        // Define a FullTextIndexColumn object variable by supplying the parent index and column name arguments
        in the constructor.
        FullTextIndexColumn ftic = default(FullTextIndexColumn);
        ftic = new FullTextIndexColumn(fti, "Name");

        // Add the indexed column to the index.
        fti.IndexedColumns.Add(ftic);
        fti.ChangeTracking = ChangeTracking Automatic;

        // Specify the unique index on the table that is required by the Full Text Search index.
        fti.UniqueIndexName = "AK_ProductCategory_Name";

        // Specify the catalog associated with the index.
        fti.CatalogName = "Test_Catalog";

        // Create the Full Text Search index on the instance of SQL Server.
        fti.Create();
    }
}

```

Creating a Full-Text Search Service in PowerShell

This code example creates a full-text search catalog for the `ProductCategory` table in the AdventureWorks2012 sample database. It then creates a full-text search index on the Name column in the `ProductCategory` table. The full-text search index requires that there is a unique index already defined on the column.

```
# Example of implementing a full text search on the default instance.
# Set the path context to the local, default instance of SQL Server and database tables

CD \sql\localhost\default\databases
$db = get-item AdventureWorks2012

CD AdventureWorks\tables

#Get a reference to the table
$tb = get-item Production.ProductCategory

# Define a FullTextCatalog object variable by specifying the parent database and name arguments in the
constructor.

$ftc = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextCatalog -argumentlist $db,
"Test_Catalog2"
$ftc.IsDefault = $true

# Create the Full Text Search catalog on the instance of SQL Server.
$ftc.Create()

# Define a FullTextIndex object variable by supplying the parent table argument in the constructor.
$fti = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextIndex -argumentlist $tb

# Define a FullTextIndexColumn object variable by supplying the parent index
# and column name arguments in the constructor.

$ftic = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FullTextIndexColumn -argumentlist $fti, "Name"

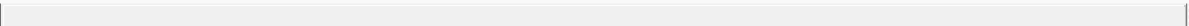
# Add the indexed column to the index.
$fti.IndexedColumns.Add($ftic)

# Set change tracking
$fti.ChangeTracking = [Microsoft.SqlServer.Management.SMO.ChangeTracking]::Automatic

# Specify the unique index on the table that is required by the Full Text Search index.
$fti.UniqueIndexName = "AK_ProductCategory_Name"





# Specify the catalog associated with the index.
$fti.CatalogName = "Test_Catalog2"

# Create the Full Text Search Index
$fti.Create()
```



Managing Service Broker

11/16/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the Service Broker objects are found in the **Microsoft.SqlServer.Management.Smo.Broker** namespace, which requires a reference to the Microsoft.SqlServer.Smo.dll. A reference to the Microsoft.SqlServer.ServiceBrokerEnum.dll is also required for supporting class information.

SMO provides a set of Service Broker objects that permit programmatic management (DDL) of the Service Broker implementation. This includes defining the message types, contracts, queues, and services. Because SMO is a management tool that is not intended for data manipulation, sending and receiving Service Broker messages is not supported by SMO.

In SMO, the [ServiceBroker](#) object is the top-level class under which all the Service Broker functionality resides. A Service Broker implementation is required for each database that is participating in the distributed messaging application. Therefore, the [ServiceBroker](#) object is a child of the [Database](#) object.

The [ServiceBroker](#) object contains collections of the following objects that are used to define the Service Broker implementation:





- [MessageType](#) objects represent message types that define the content of messages.
- [MessageTypeMapping](#) objects represent contracts that specify the direction and type of messages in a given conversation.
- [ServiceQueue](#) objects store messages prior to sending and after they are received. They provide asynchronous communication between services, as well as other benefits, such as automatically locking messages in the same conversation group.
- [BrokerService](#) objects represent Service Broker services, which are the addressable endpoints for conversations. Service Broker messages are sent from one service to another service. A service specifies a queue to hold messages, and specifies the contracts for which the service can be the target.
- [RemoteServiceBinding](#) objects represent the settings that Service Broker uses for security and authentication when communicating with a remote service.
- [ServiceRoute](#) objects represents a Service Broker route, which contains the location information for the service and the database on which it is defined. A route is required for message delivery. By default, each database contains a route that specifies the location as the current instance of SQL Server.

See Also

[Microsoft.SqlServer.Management.Smo.Broker](#)
[SQL Server Service Broker](#)

Managing Services and Network Settings by Using WMI Provider

11/16/2017 • 4 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The WMI provider is a published interface that is used by Microsoft Management Console (MMC) to manage the SQL Server services and network protocols. In SMO, the [ManagedComputer](#) object represents the WMI Provider.

The [ManagedComputer](#) object operates independently of the connection established with the [Server](#) object to an instance of SQL Server, and uses Windows credentials to connect to the WMI service.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

For programs that use the SQL Server WMI provider, you must include the **Imports** statement to qualify the WMI namespace. Insert the statement after the other **Imports** statements, before any declarations in the application, such as:

```
Imports Microsoft.SqlServer.Management.Smo
```

```
Imports Microsoft.SqlServer.Management.Common
```

```
Imports Microsoft.SqlServer.Management.Smo.Wmi
```

Stopping and Restarting the Microsoft SQL Server Service to the Instance of SQL Server in Visual Basic

This code example shows how to stop and start services by using the SMO [ManagedComputer](#) object. This provides an interface to the WMI Provider for Configuration Management.

```

'Declare and create an instance of the ManagedComputer object that represents the WMI Provider services.
Dim mc As ManagedComputer
mc = New ManagedComputer()
'Iterate through each service registered with the WMI Provider.
Dim svc As Service
For Each svc In mc.Services
    Console.WriteLine(svc.Name)
Next
'Reference the Microsoft SQL Server service.
svc = mc.Services("MSSQLSERVER")
'Stop the service if it is running and report on the status continuously until it has stopped.
If svc.ServiceState = ServiceState.Running Then
    svc.Stop()

    Console.WriteLine(String.Format("{0} service state is {1}", svc.Name, svc.ServiceState))
    Do Until String.Format("{0}", svc.ServiceState) = "Stopped"
        Console.WriteLine(String.Format("{0}", svc.ServiceState))
        svc.Refresh()
    Loop
    Console.WriteLine(String.Format("{0} service state is {1}", svc.Name, svc.ServiceState))
    'Start the service and report on the status continuously until it has started.
    svc.Start()
    Do Until String.Format("{0}", svc.ServiceState) = "Running"
        Console.WriteLine(String.Format("{0}", svc.ServiceState))
        svc.Refresh()
    Loop
    Console.WriteLine(String.Format("{0} service state is {1}", svc.Name, svc.ServiceState))

Else
    Console.WriteLine("SQL Server service is not running.")
End If

```

Enabling a Server Protocol using a URN String in Visual Basic

The code example shows how to identify a server protocol using a URN object, and then enable the protocol.

```

'This program must run with administrator privileges.
'Declare the ManagedComputer WMI interface.
Dim mc As New ManagedComputer()

'Create a URN object that represents the TCP server protocol.
Dim u As New Urn("ManagedComputer[@Name='V-ROBMA3']/ServerInstance[@Name='MSSQLSERVER']/ServerProtocol[@Name='Tcp']")

'Declare the serverProtocol variable and return the ServerProtocol object.
Dim sp As ServerProtocol
sp = mc.GetSmoObject(u)

'Enable the protocol.
sp.IsEnabled = True

'propagate back to the service
sp.Alter()

```

Enabling a Server Protocol using a URN String in PowerShell

The code example shows how to identify a server protocol using a URN object, and then enable the protocol.


```
#This example shows how to identify a server protocol using a URN object, and then enable the protocol
#This program must run with administrator privileges.

#Load the assembly containing the classes used in this example
[reflection.assembly]::LoadWithPartialName("Microsoft.SqlServer.SqlWmiManagement")

#Get a managed computer instance
$mc = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Wmi.ManagedComputer

#Create a URN object that represents the TCP server protocol
#Change 'MyPC' to the name of the your computer
$urn = New-Object -TypeName Microsoft.SqlServer.Management.Sdk.Sfc.Urn -argumentlist
"ManagedComputer[@Name='MyPC']/ServerInstance[@Name='MSSQLSERVER']/ServerProtocol[@Name='Tcp']"

#Get the protocol object
$sp = $mc.GetSmoObject($urn)

#enable the protocol on the object
$sp.IsEnabled = $true

#propagate back to actual service
$sp.Alter()
```

Starting and stopping a service in Visual C#

The code example shows how to stop and start an instance of SQL Server.

```

{
    //Declare and create an instance of the ManagedComputer
    //object that represents the WMI Provider services.
    ManagedComputer mc;
    mc = new ManagedComputer();
    //Iterate through each service registered with the WMI Provider.

    foreach (Service svc in mc.Services)
    {
        Console.WriteLine(svc.Name);
    }
    //Reference the Microsoft SQL Server service.
    Service Mysvc = mc.Services["MSSQLSERVER"];
    //Stop the service if it is running and report on the status
    // continuously until it has stopped.
    if (Mysvc.ServiceState == ServiceState.Running) {
        Mysvc.Stop();
        Console.WriteLine(string.Format("{0} service state is {1}", Mysvc.Name, Mysvc.ServiceState));
        while (!(string.Format("{0}", Mysvc.ServiceState) == "Stopped")) {
            Console.WriteLine(string.Format("{0}", Mysvc.ServiceState));
            Mysvc.Refresh();
        }
        Console.WriteLine(string.Format("{0} service state is {1}", Mysvc.Name, Mysvc.ServiceState));
    }
    //Start the service and report on the status continuously
    //until it has started.
    Mysvc.Start();
    while (!(string.Format("{0}", Mysvc.ServiceState) == "Running")) {
        Console.WriteLine(string.Format("{0}", Mysvc.ServiceState));
        Mysvc.Refresh();
    }
    Console.WriteLine(string.Format("{0} service state is {1}", Mysvc.Name, Mysvc.ServiceState));
    Console.ReadLine();
}
else {
    Console.WriteLine("SQL Server service is not running.");
    Console.ReadLine();
}
}

```

Starting and stopping a service in PowerShell

The code example shows how to stop and start an instance of SQL Server.

```
#Load the assembly containing the objects used in this example
[reflection.assembly]::LoadWithPartialName("Microsoft.SqlServer.SqlWmiManagement")

#Get a managed computer instance
$mc = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Wmi.ManagedComputer

#List out all sql server instances running on this mc
foreach ($Item in $mc.Services){$Item.Name}

#Get the default sql server database engine service
$svc = $mc.Services["MSSQLSERVER"]

# for stopping and starting services PowerShell must run as administrator





#Stop this service
$svc.Stop()
$svc.Refresh()
while ($svc.ServiceState -ne "Stopped")
{
    $svc.Refresh()
    $svc.ServiceState
}
"Service" + $svc.Name + " is now stopped"
"Starting " + $svc.Name
$svc.Start()
$svc.Refresh()
while ($svc.ServiceState -ne "Running")
{
    $svc.Refresh()
    $svc.ServiceState
}
$svc.ServiceState
"Service" + $svc.Name + "is now started"
```

See Also

[WMI Provider for Configuration Management Concepts](#)

Managing Users, Roles, and Logins

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, logins are represented by the [Login](#) object. When the logon exists in SQL Server, it can be added to a server role. The server role is represented by the [ServerRole](#) object. The database role is represented by the [DatabaseRole](#) object and the application role is represented by the [ApplicationRole](#) object.

Privileges associated with the server level are listed as properties of the [ServerPermission](#) object. The server level privileges can be granted to, denied to, or revoked from individual logon accounts.

Every [Database](#) object has a [UserCollection](#) object that specifies all users in the database. Each user is associated with a logon. One logon can be associated with users in more than one database. The [Login](#) object's [EnumDatabaseMappings](#) method can be used to list all users in every database that is associated with the logon. Alternatively, the [User](#) object's [Login](#) property specifies the logon that is associated with the user.

SQL Server databases also have roles that specify a set of database level privileges that let a user perform specific tasks. Unlike server roles, database roles are not fixed. They can be created, modified, and removed. Privileges and users can be assigned to a database role for bulk administration.

Example

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Enumerating Logins and Associated Users in Visual C#

Every user in a database is associated with a logon. The logon can be associated with users in more than one database. The code example shows how to call the [EnumDatabaseMappings](#) method of the [Login](#) object to list all the database users who are associated with the logon. The example creates a logon and user in the AdventureWorks2012 database to make sure there is mapping information to enumerate.

```

{
Server srv = new Server();
//Iterate through each database and display.

foreach ( Database db in srv.Databases) {
    Console.WriteLine("=====");
    Console.WriteLine("Login Mappings for the database: " + db.Name);
    Console.WriteLine(" ");
    //Run the EnumLoginMappings method and return details of database user-login mappings to a DataTable object
    variable.
    DataTable d;
    d = db.EnumLoginMappings();
    //Display the mapping information.
    foreach (DataRow r in d.Rows) {
        foreach (DataColumn c in r.Table.Columns) {
            Console.WriteLine(c.ColumnName + " = " + r[c]);
        }
        Console.WriteLine(" ");
    }
}
}
}

```

Enumerating Logins and Associated Users in PowerShell

Every user in a database is associated with a login. The login can be associated with users in more than one database. The code example shows how to call the [EnumDatabaseMappings](#) method of the [Login](#) object to list all the database users who are associated with the login. The example creates a login and user in the AdventureWorks2012 database to make sure there is mapping information to enumerate.

```

# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\Default\Databases

#Iterate through all databases
foreach ($db in Get-ChildItem)
{
    "====="
    "Login Mappings for the database: "+ $db.Name

    #get the datatable containing the mapping from the smo database object
    $dt = $db.EnumLoginMappings()

    #display the results
    foreach($row in $dt.Rows)
    {
        foreach($col in $row.Table.Columns)
        {
            $col.ColumnName + "=" + $row[$col]
        }
    }
}

```

Managing Roles and Users

This sample demonstrates how to how to manage roles and users. To run this sample you will need to reference the following assemblies:

- Microsoft.SqlServer.Smo.dll
- Microsoft.SqlServer.Management.Sdk.Sfc.dll

- Microsoft.SqlServer.ConnectionInfo.dll
- Microsoft.SqlServer.SqlEnum.dll

```

using Microsoft.SqlServer.Management.Smo;
using System;

public class A {
    public static void Main() {
        Server svr = new Server();
        Database db = new Database(svr, "TESTDB");
        db.Create();

        // Creating Logins
        Login login = new Login(svr, "Login1");
        login.LoginType = LoginType.SqlLogin;
        login.Create("password@1");

        Login login2 = new Login(svr, "Login2");
        login2.LoginType = LoginType.SqlLogin;
        login2.Create("password@1");

        // Creating Users in the database for the logins created
        User user1 = new User(db, "User1");
        user1.Login = "Login1";
        user1.Create();

        User user2 = new User(db, "User2");
        user2.Login = "Login2";
        user2.Create();

        // Creating database permission Sets
        DatabasePermissionSet dbPermSet = new DatabasePermissionSet(DatabasePermission.AlterAnySchema);
        dbPermSet.Add(DatabasePermission.AlterAnyUser);

        DatabasePermissionSet dbPermSet2 = new DatabasePermissionSet(DatabasePermission.CreateType);
        dbPermSet2.Add(DatabasePermission.CreateSchema);
        dbPermSet2.Add(DatabasePermission.CreateTable);

        // Creating Database roles
        DatabaseRole role1 = new DatabaseRole(db, "Role1");
        role1.Create();

        DatabaseRole role2 = new DatabaseRole(db, "Role2");
        role2.Create();

        // Granting Database Permission Sets to Roles
        db.Grant(dbPermSet, role1.Name);
        db.Grant(dbPermSet2, role2.Name);

        // Adding members (Users / Roles) to Role
        role1.AddMember("User1");

        role2.AddMember("User2");





        // Role1 becomes a member of Role2
        role2.AddMember("Role1");

        // Enumerating through explicit permissions granted to Role1
        // enumerates all database permissions for the Grantee
        DatabasePermissionInfo[] dbPermsRole1 = db.EnumDatabasePermissions("Role1");
        foreach (DatabasePermissionInfo dbp in dbPermsRole1) {
            Console.WriteLine(dbp.Grantee + " has " + dbp.PermissionType.ToString() + " permission.");
        }
        Console.WriteLine(" ");
    }
}

```

Programming Specific Tasks

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse





Programming-specific tasks using SMO objects include complex subjects that would only be required by programs with a specific function, such as backing up, monitoring statistics, replication, managing instance objects, and setting configuration options.

TOPIC	DESCRIPTION
Using Linked Servers in SMO	Describes how SMO uses the LinkedServer object to link OLE-DB servers.
Configuring SQL Server in SMO	Describes how to view and modify configuration settings for the instance of SQL Server in SMO.
Using Table and Index Partitioning	Describes how to use index and table partitioning in SMO.
Using Filegroups and Files to Store Data	Describes how to use file groups in SMO.
Managing Services and Network Settings by Using WMI Provider	Describes several ways to keep track of the instance of SQL Server by using the ManagedComputer object that represents the WMI Provider for Configuration Management.
Working with Database Objects	Describes how to create instance classes that represent objects on the instance of SQL Server.
Managing Users, Roles, and Logins	Describes how to use security roles in SMO.
Granting, Revoking, and Denying Permissions	Describes how to use the SMO to grant, revoke, and deny permissions to users or members of a role.
Using Encryption	Describes how to protect data using encryption in SMO.
Scheduling Automatic Administrative Tasks in SQL Server Agent	Describes how to use the SQL Server Agent to monitor, report, and schedule jobs in SMO.
Backing Up and Restoring Databases and Transaction Logs	Describes how to back up and restore databases and transaction logs in SMO.
Scripting	Describes how to script objects and discover dependencies between objects in SMO.
Transferring Data	Describes how to transfer data in SMO.
Using Database Mail	Describes how SMO makes use of e-mail services.
Managing Service Broker	Describes how to set up Service Broker using SMO.

TOPIC	DESCRIPTION
Using XML Schemas	Describes how to use the XML data type in SMO.
Using Synonyms	Describes how to create synonyms in SMO.
Using Messages	Describes how to use system messages, and how to define your own user-defined messages.
Implementing Full-Text Search	Describes how to implement full-text search catalogs and indexes in SMO.
Implementing Endpoints	Describes how to create endpoints to handle payloads for Database Mirroring, SOAP requests, and Service Broker.
Creating and Updating Statistics	Describes how to set up and monitor statistics on a database in SMO.
Tracing and Replaying Events	Describes how to use the Trace and Replay objects in SMO to trace and replay events.

Scheduling Automatic Administrative Tasks in SQL Server Agent

11/16/2017 • 6 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the SQL Server Agent is represented by the following objects:

- The [JobServer](#) object has three collections of jobs, alerts and operators.
- The [OperatorCollection](#) object represents a list of pager, e-mail addresses and net send operators that can be notified of events automatically by the Microsoft SQL Server Agent.
- The [AlertCollection](#) object represents a list of circumstances such as system events or performance conditions that are monitored by SQL Server.
- The [JobCollection](#) object is slightly more complex. It represents a list of multi-step tasks that run at specified schedules. The steps and schedule information are stored in the [JobStep](#) and [JobSchedule](#) objects.

The SQL Server Agent objects are in the [Microsoft.SqlServer.Management.Smo.Agent](#) namespace.

Examples

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

For programs that use SQL Server Agent, you must include the **using** statement to qualify the Agent namespace. Insert the statement after the other **using** statements, before any declarations in the application, such as:

```
using Microsoft.SqlServer.Management.Smo;  
  
using Microsoft.SqlServer.Management.Common;  
  
using Imports Microsoft.SqlServer.Management.Smo.Agent;
```

Creating a Job with Steps and a Schedule in Visual C#

This code example creates a job with steps and a schedule, and then informs an operator.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();

    //Define an Operator object variable by supplying the Agent (parent JobServer object) and the name
in the constructor.
    Operator op = new Operator(srv.JobServer, "Test_Operator");

    //Set the Net send address.
    op.NetSendAddress = "Network1_PC";

    //Create the operator on the instance of SQL Server Agent.
    op.Create();

    //Define a Job object variable by supplying the Agent and the name arguments in the constructor
and setting properties.
    Job jb = new Job(srv.JobServer, "Test_Job");

    //Specify which operator to inform and the completion action.
    jb.OperatorToNetSend = "Test_Operator";
    jb.NetSendLevel = CompletionAction.Always;

    //Create the job on the instance of SQL Server Agent.
    jb.Create();

    //Define a JobStep object variable by supplying the parent job and name arguments in the
constructor.
    JobStep jbstp = new JobStep(jb, "Test_Job_Step");
    jbstp.Command = "Test_StoredProc";
    jbstp.OnSuccessAction = StepCompletionAction.QuitWithSuccess;
    jbstp.OnFailAction = StepCompletionAction.QuitWithFailure;

    //Create the job step on the instance of SQL Agent.
    jbstp.Create();

    //Define a JobSchedule object variable by supplying the parent job and name arguments in the
constructor.

    JobSchedule jbsch = new JobSchedule(jb, "Test_Job_Schedule");

    //Set properties to define the schedule frequency, and duration.
    jbsch.FrequencyTypes = FrequencyTypes.Daily;
    jbsch.FrequencySubDayTypes = FrequencySubDayTypes.Minute;
    jbsch.FrequencySubDayInterval = 30;
    TimeSpan ts1 = new TimeSpan(9, 0, 0);
    jbsch.ActiveStartTimeOfDay = ts1;

    TimeSpan ts2 = new TimeSpan(17, 0, 0);
    jbsch.ActiveEndTimeOfDay = ts2;
    jbsch.FrequencyInterval = 1;

    System.DateTime d = new System.DateTime(2003, 1, 1);
    jbsch.ActiveStartDate = d;

    //Create the job schedule on the instance of SQL Agent.
    jbsch.Create();
}

```

Creating a Job with Steps and a Schedule in PowerShell

This code example creates a job with steps and a schedule, and then informs an operator.

```

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#Define an Operator object variable by supplying the Agent (parent JobServer object) and the name in the
constructor.
$op = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.Operator -argumentlist $srv.JobServer,
"Test_Operator"

#Set the Net send address.
$op.NetSendAddress = "Network1_PC"

#Create the operator on the instance of SQL Agent.
$op.Create()

#Define a Job object variable by supplying the Agent and the name arguments in the constructor and setting
properties.
$jb = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.Job -argumentlist $srv.JobServer,
"Test_Job"

#Specify which operator to inform and the completion action.
$jb.OperatorToNetSend = "Test_Operator";
$jb.NetSendLevel = [Microsoft.SqlServer.Management.SMO.Agent.CompletionAction]::Always

#Create the job on the instance of SQL Server Agent.
$jb.Create()

#Define a JobStep object variable by supplying the parent job and name arguments in the constructor.
$jbstp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.JobStep -argumentlist $jb,
"Test_Job_Step"
$jbstp.Command = "Test_StoredProc";
$jbstp.OnSuccessAction = [Microsoft.SqlServer.Management.SMO.Agent.StepCompletionAction]::QuitWithSuccess;
$jbstp.OnFailAction = [Microsoft.SqlServer.Management.SMO.Agent.StepCompletionAction]::QuitWithFailure;

#Create the job step on the instance of SQL Agent.
$jbstp.Create();

#Define a JobSchedule object variable by supplying the parent job and name arguments in the constructor.
$jsch = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.JobSchedule -argumentlist $jb,
"Test_Job_Schedule"

#Set properties to define the schedule frequency, and duration.
$jsch.FrequencyTypes = [Microsoft.SqlServer.Management.SMO.Agent.FrequencyTypes]::Daily

$jsch.FrequencySubDayTypes = [Microsoft.SqlServer.Management.SMO.Agent.FrequencySubDayTypes]::Minute
$jsch.FrequencySubDayInterval = 30
$ts1 = New-Object -TypeName TimeSpan -argumentlist 9, 0, 0
$jsch.ActiveStartTimeOfDay = $ts1
$ts2 = New-Object -TypeName TimeSpan -argumentlist 17, 0, 0
$jsch.ActiveEndTimeOfDay = $ts2
$jsch.FrequencyInterval = 1
$jsch.ActiveStartDate = "01/01/2003"

#Create the job schedule on the instance of SQL Agent.
$jsch.Create();

```

Creating an Alert in Visual C#

This code example creates an alert that is triggered by a performance condition. The condition must be provided in a specific format:

ObjectName|CounterName|Instance|ComparisionOp|CompValue

An operator is required for the alert notification. The [Operator](#) type requires square parentheses because **operator** is a Visual C# keyword.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();

    //Define an Alert object variable by supplying the SQL Server Agent and the name arguments in the
    constructor.
    Alert al = new Alert(srv.JobServer, "Test_Alert");

    //Specify the performance condition string to define the alert.
    al.PerformanceCondition = "SQLServer:General Statistics|User Connections||>|3";

    //Create the alert on the SQL Agent.
    al.Create();

    //Define an Operator object variable by supplying the SQL Server Agent and the name arguments in
    the constructor.

    Operator op = new Operator(srv.JobServer, "Test_Operator");
    //Set the net send address.
    op.NetSendAddress = "NetworkPC";
    //Create the operator on the SQL Agent.
    op.Create();
    //Run the AddNotification method to specify the operator is notified when the alert is raised.
    al.AddNotification("Test_Operator", NotifyMethods.NetSend);
}

```

Creating an Alert in PowerShell

This code example creates an alert that is triggered by a performance condition. The condition must be provided in a specific format:

ObjectName|CounterName|Instance|ComparisionOp|CompValue

An operator is required for the alert notification. The [Operator](#) type requires square parentheses because **operator** is a Visual C# keyword.

```

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#Define an Alert object variable by supplying the SQL Agent and the name arguments in the constructor.
$al = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.Alert -argumentlist $srv.JobServer,
"Test_Alert"

#Specify the performance condition string to define the alert.
$al.PerformanceCondition = "SQLServer:General Statistics|User Connections||>|3"

#Create the alert on the SQL Agent.
$al.Create()

#Define an Operator object variable by supplying the Agent (parent JobServer object) and the name in the
constructor.
$op = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Agent.Operator -argumentlist $srv.JobServer,
"Test_Operator"

#Set the Net send address.
$op.NetSendAddress = "Network1_PC"

#Create the operator on the instance of SQL Agent.
$op.Create()

#Run the AddNotification method to specify the operator is notified when the alert is raised.
$ns = [Microsoft.SqlServer.Management.SMO.Agent.NotifyMethods]::NetSend
$al.AddNotification("Test_Operator", $ns)

#Drop the alert and the operator
$al.Drop()
$op.Drop()

```

Allowing User Access to Subsystem by Using a Proxy Account in Visual C#

This code example shows how to allow a user access to a specified subsystem by using the [AddSubSystem](#) method of the [ProxyAccount](#) object.

```

//Connect to the local, default instance of SQL Server.
{
Server srv = default(Server);
srv = new Server();
//Declare a JobServer object variable and reference the SQL Server Agent.
JobServer js = default(JobServer);
js = srv.JobServer;
//Define a Credential object variable by supplying the parent server and name arguments in the constructor.
Credential c = default(Credential);
c = new Credential(srv, "Proxy_acnt");
//Set the identity to a valid login represented by the vIdentity string variable.
//The sub system will run under this login.
c.Identity = vIdentity;
//Create the credential on the instance of SQL Server.
c.Create();
//Define a ProxyAccount object variable by supplying the SQL Server Agent, the name, the credential, the
description arguments in the constructor.
ProxyAccount pa = default(ProxyAccount);
pa = new ProxyAccount(js, "Test_proxy", "Proxy_acnt", true, "Proxy account for users to run job steps in
command shell.");
//Create the proxy account on the SQL Agent.
pa.Create();
//Add the login, represented by the vLogin string variable, to the proxy account.
pa.AddLogin(vLogin);
//Add the CmdExec subsystem to the proxy account.
pa.AddSubSystem(AgentSubSystem.CmdExec);
}
//Now users logged on as vLogin can run CmdExec job steps with the specified credentials.

```





See Also

[SQL Server Agent](#)

[Implement Jobs](#)

Scripting

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Scripting in SMO is controlled by the [Scripter](#) object and its child objects, or the **Script** method on individual objects. The [Scripter](#) object controls the mapping out of dependency relationships for objects on an instance of Microsoft SQL Server.

Advanced scripting by using the [Scripter](#) object and its child objects is a three phase process:

1. Discovery
2. List generation
3. Script generation

The discovery phase uses the [DependencyWalker](#) object. Given an URN list of objects, the [DiscoverDependencies](#) method of the [DependencyWalker](#) object returns a [DependencyTree](#) object for the objects in the URN list. The Boolean *fParents* parameter is used to select whether the parents or the children of the specified object are to be discovered. The dependency tree can be modified at this stage.

In the list generation phase, the tree is passed in and the resulting list is returned. This object list is in scripting order and can be manipulated.

The list generation phases use the [WalkDependencies](#) method to return a [DependencyTree](#). The [DependencyTree](#) can be modified at this stage.

In the third and final phase, a script is generated with the specified list and scripting options. The result is returned as a [StringCollection](#) system object. In this phase the dependent object names are then extracted from the Items collection of the [DependencyTree](#) object and properties such as [NumberOfSiblings](#) and [FirstChild](#).

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

This code example requires an **Imports** statement for the System.Collections.Specialized namespace. Insert this with the other Imports statements, before any declarations in the application.

```
Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Common
Imports System.Collections.Specialized
```

Scripting Out the Dependencies for a Database in Visual Basic

This code example shows how to discover the dependencies and iterate through the list to display the results.


```

' compile with:
' /r:Microsoft.SqlServer.Smo.dll
' /r:Microsoft.SqlServer.ConnectionInfo.dll
' /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Sdk.Sfc

Public Class A
    Public Shared Sub Main()
        ' database name
        Dim dbName As [String] = "AdventureWorksLT2012" ' database name

        ' Connect to the local, default instance of SQL Server.
        Dim srv As New Server()

        ' Reference the database.
        Dim db As Database = srv.Databases(dbName)

        ' Define a Scripter object and set the required scripting options.
        Dim scrp As New Scripter(srv)
        scrp.Options.ScriptDrops = False
        scrp.Options.WithDependencies = True
        scrp.Options.Indexes = True ' To include indexes
        scrp.Options.DriAllConstraints = True ' to include referential constraints in the script

        ' Iterate through the tables in database and script each one. Display the script.
        For Each tb As Table In db.Tables
            ' check if the table is not a system table
            If tb.IsSystemObject = False Then
                Console.WriteLine("-- Scripting for table " + tb.Name)

                ' Generating script for table tb
                Dim sc As System.Collections.Specialized.StringCollection = scrp.Script(New Urn() {tb.Urn})
                For Each st As String In sc
                    Console.WriteLine(st)
                Next
                Console.WriteLine("--")
            End If
        Next
    End Sub
End Class

```

Scripting Out the Dependencies for a Database in Visual C#

This code example shows how to discover the dependencies and iterate through the list to display the results.

```

// compile with:
// /r:Microsoft.SqlServer.Smo.dll
// /r:Microsoft.SqlServer.ConnectionInfo.dll
// /r:Microsoft.SqlServer.Management.Sdk.Sfc.dll

using System;
using Microsoft.SqlServer.Management.Smo;
using Microsoft.SqlServer.Management.Sdk.Sfc;

public class A {
    public static void Main() {
        String dbName = "AdventureWorksLT2012"; // database name

        // Connect to the local, default instance of SQL Server.
        Server srv = new Server();

        // Reference the database.
        Database db = srv.Databases[dbName];

        // Define a Scripter object and set the required scripting options.
        Scripter scrp = new Scripter(srv);
        scrp.Options.ScriptDrops = false;
        scrp.Options.WithDependencies = true;
        scrp.Options.Indexes = true;    // To include indexes
        scrp.Options.DriAllConstraints = true; // to include referential constraints in the script

        // Iterate through the tables in database and script each one. Display the script.
        foreach (Table tb in db.Tables) {
            // check if the table is not a system table
            if (tb.IsSystemObject == false) {
                Console.WriteLine("-- Scripting for table " + tb.Name);

                // Generating script for table tb
                System.Collections.Specialized.StringCollection sc = scrp.Script(new Urn[]{tb.Urn});
                foreach (string st in sc) {
                    Console.WriteLine(st);
                }
                Console.WriteLine("--");
            }
        }
    }
}

```

Scripting Out the Dependencies for a Database in PowerShell

This code example shows how to discover the dependencies and iterate through the list to display the results.

```
# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\default

# Create a Scripter object and set the required scripting options.
$scrp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Scripter -ArgumentList (Get-Item .)
$scrp.Options.ScriptDrops = $false
$scrp.Options.WithDependencies = $true
$scrp.Options.IncludeIfExists = $true





# Set the path context to the tables in AdventureWorks2012.

CD Databases\AdventureWorks2012\Tables

foreach ($Item in Get-ChildItem)
{
    $scrp.Script($Item)
}
```

Tracing and Replaying Events

11/16/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the **Trace** and **Replay** objects in the [Microsoft.SqlServer.Management.Trace](#) namespace provide programmatic access to the SQL Server Profiler functionality, which is used for monitoring an instance of SQL Server or Analysis Services. You can capture and save data about each event to a file or table to analyze later. For example, you can monitor a production environment to see which procedures are impeding performance by executing too slowly.

The **Trace** and **Replay** objects provide a set of objects that can be used to create traces on an instance of SQL Server. These objects can be used from within your own applications to create traces manually for SQL Server or Analysis Services. Additionally, SMO **Trace** objects can be used to read SQL Trace files and tables that were created by monitoring SQL Server, Analysis Services, or DTS logging.

SMO **Trace** objects let you perform the following functions:

- Create a trace.
- Set filters on the trace.
- Set the events that are being traced.
- Stop or start a trace.
- Read trace files, and trace tables.
- Get information about events on a trace.
- Get information about filters on a trace.
- Manipulate trace data programmatically.
- Write trace tables and trace files.
- Replay trace files or trace tables.

The trace data from the **Trace** and **Replay** objects can be used by the SMO application, or it can be examined manually by using [SQL Server Profiler](#). The trace data is also compatible with the [SQL Trace](#) stored procedures that also provide tracing capabilities.

The SMO trace objects reside in the [Microsoft.SqlServer.Management.Trace](#) namespace, which requires a reference to the Microsoft.SqlServer.ConnectionInfo.dll file.





The **Trace** and **Replay** objects require a [ServerConnectionServer](#) object to establish a connection with the instance of SQL Server. The [ServerConnection](#) object resides in the [Microsoft.SqlServer.Management.Common](#) namespace, which requires a reference to the Microsoft.SqlServer.ConnectionInfo.dll file.

NOTE

The **Trace** and **Replay** objects are not supported on a 64-bit platform.

Transferring Data

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

The [Transfer](#) class is a utility class that provides tools to transfer objects and data.

Objects in the database schema are transferred by executing a generated script on the target server. [Table](#) data is transferred with a dynamically created DTS package.

The [Transfer](#) object uses the [SQLBulkCopy](#) API to transfer data. Also, the methods and properties that are used to perform data transfers reside on the [Transfer](#) object instead of the [Database](#) object. Moving functionality from the instance classes to utility classes is consistent with a lighter object model because the code for specific tasks is loaded only when it is required.

The [Transfer](#) object does not support data transfers to a target database that has a [CompatibilityLevel](#) less than the version of the instance of SQL Server.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Transferring Schema and Data from One Database to Another in Visual Basic

This code example shows how to transfer schema and data from one database to another using the [Transfer](#) object.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Create a new database that is to be destination database.
Dim dbCopy As Database
dbCopy = New Database(srv, "AdventureWorks2012Copy")
dbCopy.Create()
'Define a Transfer object and set the required options and properties.
Dim xfr As Transfer
xfr = New Transfer(db)
xfr.CopyAllTables = True
xfr.Options.WithDependencies = True
xfr.Options.ContinueScriptingOnError = True
xfr.DestinationDatabase = "AdventureWorks2012Copy"
xfr.DestinationServer = srv.Name
xfr.DestinationLoginSecure = True
xfr.CopySchema = True
'Script the transfer. Alternatively perform immediate data transfer with TransferData method.
xfr.ScriptTransfer()
```

Transferring Schema and Data from One Database to Another in Visual

C#

This code example shows how to transfer schema and data from one database to another using the [Transfer](#) object.

```
{
    Server srv;
    srv = new Server();
    //Reference the AdventureWorks2012 database
    Database db;
    db = srv.Databases["AdventureWorks2012"];
    //Create a new database that is to be destination database.
    Database dbCopy;
    dbCopy = new Database(srv, "AdventureWorks2012Copy");
    dbCopy.Create();
    //Define a Transfer object and set the required options and properties.
    Transfer xfr;
    xfr = new Transfer(db);
    xfr.CopyAllTables = true;
    xfr.Options.WithDependencies = true;
    xfr.Options.ContinueScriptingOnError = true;
    xfr.DestinationDatabase = "AdventureWorks2012Copy";
    xfr.DestinationServer = srv.Name;
    xfr.DestinationLoginSecure = true;
    xfr.CopySchema = true;
    //Script the transfer. Alternatively perform immediate data transfer
    // with TransferData method.
    xfr.ScriptTransfer();
}
```

Transferring Schema and Data from One Database to Another in PowerShell

This code example shows how to transfer schema and data from one database to another using the [Transfer](#) object.

```
#Connect to the local, default instance of SQL Server.

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#Reference the AdventureWorks2012 database.
$db = $srv.Databases["AdventureWorks2012"]





#Create a database to hold the copy of AdventureWorks
$dbCopy = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Database -argumentlist $srv,
"AdventureWorksCopy"
$dbCopy.Create()

#Define a Transfer object and set the required options and properties.
$xfr = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Transfer -argumentlist $db

#Set this objects properties
$xfr.CopyAllTables = $true
$xfr.Options.WithDependencies = $true
$xfr.Options.ContinueScriptingOnError = $true
$xfr.DestinationDatabase = "AdventureWorksCopy"
$xfr.DestinationServer = $srv.Name
$xfr.DestinationLoginSecure = $true
$xfr.CopySchema = $true
"Scripting Data Transfer"
#Script the transfer. Alternatively perform immediate data transfer with TransferData method.
$xfr.ScriptTransfer()
```

Using Database Mail

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the Database Mail subsystem is represented by the [SqlMail](#) object that is referenced by the [Mail](#) property. By using the SMO [SqlMail](#) object, you can configure the Database Mail subsystem and manage profiles and mail accounts. The SMO [SqlMail](#) object belongs to the **Server** object, meaning that scope of the mail accounts is at the server level.

Examples

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

For programs that use SQL Server Database Mail, you must include the **Imports** statement to qualify the Mail namespace. Insert the statement after the other **Imports** statements, before any declarations in the application, such as:

```
Imports Microsoft.SqlServer.Management.Smo
```

```
Imports Microsoft.SqlServer.Management.Common
```

```
Imports Microsoft.SqlServer.Management.Smo.Mail
```

Creating a Database Mail Account by Using Visual Basic

This code example shows how to create an e-mail account in SMO. Database Mail is represented by the [SqlMail](#) object and referenced by the [Mail](#) property of the [Server](#) object. SMO can be used to programmatically configure Database Mail, but it cannot be used to send or handle received e-mail.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server()
'Define the Database Mail service with a SqlMail object variable and reference it using the Server Mail
property.
Dim sm As SqlMail
sm = srv.Mail
'Define and create a mail account by supplying the Database Mail service, name, description, display name, and
email address arguments in the constructor.
Dim a As MailAccount
a = New MailAccount(sm, "AdventureWorks Administrator", "AdventureWorks Automated Mailer", "Mail account for
administrative e-mail.", "dba@Adventure-Works.com")
a.Create()
```

Creating a Database Mail Account by Using Visual C#

This code example shows how to create an e-mail account in SMO. Database Mail is represented by the [SqlMail](#) object and referenced by the [Mail](#) property of the [Server](#) object. SMO can be used to programmatically configure Database Mail, but it cannot be used to send or handle received e-mail.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = default(Server);
    srv = new Server();
    //Define the Database Mail service with a SqlMail object variable
    //and reference it using the Server Mail property.
    SqlMail sm;
    sm = srv.Mail;
    //Define and create a mail account by supplying the Database Mail
    //service, name, description, display name, and email address
    //arguments in the constructor.
    MailAccount a = default(MailAccount);
    a = new MailAccount(sm, "AdventureWorks2012 Administrator", "AdventureWorks2012 Automated Mailer",
"Mail account for administrative e-mail.", "dba@Adventure-Works.com");
    a.Create();
}

```

Creating a Database Mail Account by Using PowerShell

This code example shows how to create an e-mail account in SMO. Database Mail is represented by the [SqlMail](#) object and referenced by the [Mail](#) property of the [Server](#) object. SMO can be used to programmatically configure Database Mail, but it cannot be used to send or handle received e-mail.

```

#Connect to the local, default instance of SQL Server.

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server





#Define the Database Mail; reference it using the Server Mail property.
$sm = $srv.Mail

#Define and create a mail account by supplying the Database Mail service,
#name, description, display name, and email address arguments in the constructor.
$a = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Mail.MailAccount -argumentlist $sm, `
"Adventure Works Administrator", "Adventure Works Automated Mailer", `
"Mail account for administrative e-mail.", "dba@Adventure-Works.com"
$a.Create()

```


Using Encryption

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, the service master key is represented by the [ServiceMasterKey](#) object. This is referenced by the [ServiceMasterKey](#) property of the [Server](#) object. It can be regenerated by using the [Regenerate](#) method.

The database master key is represented by the [MasterKey](#) object. The [IsEncryptedByServer](#) property indicates whether or not the database master key is encrypted by the service master key. The encrypted copy in the master database is automatically updated whenever the database master key is changed.

It is possible to drop service key encryption using the [DropServiceKeyEncryption](#) method and encrypt the database master key with a password. In that situation, you will have to explicitly open the database master key before accessing private keys that it has secured.

When a database is being attached to an instance of SQL Server, you must either supply the password for the database master key or execute the [AddServiceKeyEncryption](#) method to make an unencrypted copy of the database master key available for encryption with the service master key. This step is recommended to avoid the need to explicitly open the database master key.

The [Regenerate](#) method regenerates the database master key. When the database master key is regenerated, all the keys that have been encrypted with the database master key are decrypted, and then encrypts them with the new database master key. The [DropServiceKeyEncryption](#) method removes the encryption of the database master key by the service master key. [AddServiceKeyEncryption](#) causes a copy of the master key to be encrypted using the service master key and stored in both the current database and in the master database.

In SMO, certificates are represented by the [Certificate](#) object. The [Certificate](#) object has properties that specify the public key, the name of the subject, period of validity, and information about the issuer. Permission to access the certificate is controlled by using the **Grant**, **Revoke** and **Deny** methods.

Example

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Adding a Certificate in Visual C#

The code example creates a simple certificate with an encryption password. Unlike other objects, the [Create](#) method has several overloads. The overload used in the example creates a new certificate with an encryption password.

```

{
    //Connect to the local, default instance of SQL Server.
    {
        Server srv = new Server();

        //Reference the AdventureWorks2012 database.
        Database db = srv.Databases["AdventureWorks2012"];

        //Define a Certificate object variable by supplying the parent database and name in the
        constructor.
        Certificate c = new Certificate(db, "Test_Certificate");

        //Set the start date, expiry date, and description.
        System.DateTime dt;
        DateTime.TryParse("January 01, 2010", out dt);
        c.StartDate = dt;
        DateTime.TryParse("January 01, 2015", out dt);
        c.ExpirationDate = dt;
        c.Subject = "This is a test certificate.";
        //Create the certificate on the instance of SQL Server by supplying the certificate password
        argument.
        c.Create("pGFD4bb925DGvbd2439587y");
    }
}

```

Adding a Certificate in PowerShell

The code example creates a simple certificate with an encryption password. Unlike other objects, the [Create](#) method has several overloads. The overload used in the example creates a new certificate with an encryption password.

```

# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item AdventureWorks2012

#Create a certificate

$c = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Certificate -argumentlist $db, "Test_Certificate"
$c.StartDate = "January 01, 2010"
$c.Subject = "This is a test certificate."
$c.ExpirationDate = "January 01, 2015"

#Create the certificate on the instance of SQL Server by supplying the certificate password argument.
$c.Create("pGFD4bb925DGvbd2439587y")





```

See Also

[Using Encryption Keys](#)

Using Filegroups and Files to Store Data

11/16/2017 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Data files are used to store database files. The data files are subdivided into file groups. The [Database](#) object has a [FileGroups](#) property that references a [FileGroupCollection](#) object. Each [FileGroup](#) object in that collection has a [Files](#) property. This property refers to a [DataFileCollection](#) collection that contains all the data files that belong to the database. A file group is principally used to group files together that are used to store a database object. One reason for spreading a database object over several files is that it can improve performance, especially if the files are stored on different disk drives.

Every database that is created automatically has a file group named "Primary" and a data file with the same name as the database. Additional files and groups can be added to the collections.

Examples

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Adding FileGroups and DataFiles to a Database in Visual Basic

The primary file group and data file are created automatically with default property values. The code example specifies some property values that you can use. Otherwise, the default property values are used.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a FileGroup object called SECONDARY on the database.
Dim fg1 As FileGroup
fg1 = New FileGroup(db, "SECONDARY")
'Call the Create method to create the file group on the instance of SQL Server.
fg1.Create()
'Define a DataFile object on the file group and set the FileName property.
Dim df1 As DataFile
df1 = New DataFile(fg1, "datafile1")
df1.FileName = "c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\datafile2.ndf"
'Call the Create method to create the data file on the instance of SQL Server.
df1.Create()
```

Adding FileGroups and DataFiles to a Database in Visual C#

The primary file group and data file are created automatically with default property values. The code example specifies some property values that you can use. Otherwise, the default property values are used.

```

{
    Server srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db = default(Database);
    db = srv.Databases["AdventureWorks2012"];
    //Define a FileGroup object called SECONDARY on the database.
    FileGroup fg1 = default(FileGroup);
    fg1 = new FileGroup(db, "SECONDARY");
    //Call the Create method to create the file group on the instance of SQL Server.
    fg1.Create();
    //Define a DataFile object on the file group and set the FileName property.
    DataFile df1 = default(DataFile);
    df1 = new DataFile(fg1, "datafile1");
    df1.FileName = "c:\\Program Files\\Microsoft SQL Server\\MSSQL.1\\MSSQL\\Data\\datafile2.ndf";
    //Call the Create method to create the data file on the instance of SQL Server.
    df1.Create();
}

```

Adding FileGroups and DataFiles to a Database in PowerShell

The primary file group and data file are created automatically with default property values. The code example specifies some property values that you can use. Otherwise, the default property values are used.

```

# Set the path context to the local, default instance of SQL Server.
CD sql\localhost\default\Databases\

#And the database object corresponding to AdventureWorks2012.
$db = get-item AdventureWorks2012

#Create a new filegroup
$fg1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.FileGroup -argumentlist $db, "SECONDARY"
$fg1.Create()

#Define a DataFile object on the file group and set the FileName property.
$df1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.DataFile -argumentlist $fg1, "datafile1"

#Make sure to have a directory created to hold the designated data file
$df1.FileName = "c:\\TestData\\datafile2.ndf"

#Call the Create method to create the data file on the instance of SQL Server.
$df1.Create()

```

Creating, Altering, and Removing a Log File in Visual Basic

The code example creates a [LogFile](#) object, changes one of the properties, and then removes it from the database.

```

'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a LogFile object and set the database, name, and file name properties in the constructor.
Dim lf1 As LogFile
lf1 = New LogFile(db, "logfile1", "c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\logfile1.ldf")
'Set the file growth to 6%.
lf1.GrowthType = FileGrowthType.Percent
lf1.Growth = 6
'Run the Create method to create the log file on the instance of SQL Server.
lf1.Create()
'Alter the growth percentage.
lf1.Growth = 7
lf1.Alter()
'Remove the log file.
lf1.Drop()

```

Creating, Altering, and Removing a Log File in Visual C#

The code example creates a [LogFile](#) object, changes one of the properties, and then removes it from the database.

```

//Connect to the local, default instance of SQL Server.
Server srv = new Server();
//Reference the AdventureWorks2012 database.
Database db = default(Database);
db = srv.Databases["AdventureWorks2012"];
//Define a LogFile object and set the database, name, and file name properties in the constructor.
LogFile lf1 = default(LogFile);
lf1 = new LogFile(db, "logfile1", "c:\\Program Files\\Microsoft SQL
Server\\MSSQL.10_50.MSSQLSERVER\\MSSQL\\Data\\logfile1.ldf");
//Set the file growth to 6%.
lf1.GrowthType = FileGrowthType.Percent;
lf1.Growth = 6;
//Run the Create method to create the log file on the instance of SQL Server.
lf1.Create();
//Alter the growth percentage.
lf1.Growth = 7;
lf1.Alter();
//Remove the log file.
lf1.Drop();

```

Creating, Altering, and Removing a Log File in PowerShell

The code example creates a [LogFile](#) object, changes one of the properties, and then removes it from the database.

```

#Load the assembly containing the enums used in this example
[reflection.assembly]::LoadWithPartialName("Microsoft.SqlServer.SqlEnum")

# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\default\Databases\

#And the database object corresponding to AdventureWorks2012
$db = get-item AdventureWorks2012

#Create a filegroup
$fg1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Filegroup -argumentlist $db, "Secondary"

#Call the Create method to create the file group on the instance of SQL Server.
$fg1.Create()

#Define a LogFile object on the file group and set the FileName property.
$lf1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.LogFile -argumentlist $db, "LogFile2"

#Set a location for it - make sure the directory exists
$lf1.FileName = "logfile1", "c:\\Program Files\\Microsoft SQL
Server\\MSSQL.10_50.MSSQLSERVER\\MSSQL\\Data\\logfile1.ldf"

#Set file growth to 6%
$lf1.GrowthType = [Microsoft.SqlServer.Management.Smo.FileGrowthType]::Percent
$lf1.Growth = 6.0

#Call the Create method to create the data file on the instance of SQL Server.
$lf1.Create()

#Alter a value and drop the log file
$lf1.Growth = 7.0
$lf1.Alter()
$lf1.Drop()

```





See Also

[FileGroup](#)

[Database Files and Filegroups](#)

Using Linked Servers in SMO

11/16/2017 • 1 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

A linked server represents an OLE DB data source on a remote server. Remote OLE DB data sources are linked to the instance of SQL Server by using the [LinkedServer](#) object.

Remote database servers can be linked to the current instance of Microsoft SQL Server by using an OLE DB Provider. In SMO, linked servers are represented by the [LinkedServer](#) object. The [LinkedServerLogins](#) property references a collection of [LinkedServerLogin](#) objects. These store the logon credentials that are required to establish a connection with the linked server.

OLE-DB Providers

In SMO, installed OLE-DB providers are represented by a collection of [OleDbProviderSettings](#) objects.

Example

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a link to an OLE-DB Provider Server in Visual C#

The code example shows how to create a link to a SQL Server OLE DB, heterogeneous data source by using the [LinkedServer](#) object. By specifying SQL Server as the product name, data is accessed on the linked server by using the SQL Server Client OLE DB Provider, which is the official OLE DB provider for SQL Server.

```
//Connect to the local, default instance of SQL Server.
{
    Server srv = new Server();
    //Create a linked server.
    LinkedServer lsrv = default(LinkedServer);
    lsrv = new LinkedServer(srv, "OLEDBSRV");
    //When the product name is SQL Server the remaining properties are
    //not required to be set.
    lsrv.ProductName = "SQL Server";
    lsrv.Create();
}
```

Creating a link to an OLE-DB Provider Server in PowerShell

The code example shows how to create a link to a SQL Server OLE DB, heterogeneous data source by using the [LinkedServer](#) object. By specifying SQL Server as the product name, data is accessed on the linked server by using the SQL Server Client OLE DB Provider, which is the official OLE DB provider for SQL Server.

```
#Get a server object which corresponds to the default instance
$svr = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server





#Create a linked server object which corresponds to an OLEDB type of SQL server product
$lsvr = New-Object -TypeName Microsoft.SqlServer.Management.SMO.LinkedServer -argumentlist $svr,"OLEDBSRV"

#When the product name is SQL Server the remaining properties are not required to be set.
$lsvr.ProductName = "SQL Server"

#Create the Database Object
$lsvr.Create()
```


Using Messages

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

In SMO, system messages are represented by the [SystemMessageCollection](#) object that belongs to the **Server** object. Because the system messages cannot be modified, **SystemMessage** object properties are read-only.

User-defined messages are represented programmatically in SMO by the [UserDefinedMessageCollection](#) object. Existing user-defined messages can be discovered by iterating through the collection. New user-defined messages can be created by instantiating a new **UserDefinedMessage** object and setting the appropriate properties.

Examples

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Finding a Particular System Message in Visual Basic

The code example shows how to identify a system message by ID number and display the message.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server
'Reference an existing system message using the ItemByIdAndLanguage method.
Dim msg As SystemMessage
msg = srv.SystemMessages.ItemByIdAndLanguage(14126, "us_english")
'Display the message ID and text.
Console.WriteLine(msg.ID.ToString + " " + msg.Text)
```

Finding a Particular System Message in Visual C#

The code example shows how to identify a system message by ID number and display the message.

```
{
    //Connect to the local, default instance of SQL Server.
    Server srv = new Server();
    //Reference an existing system message using the
    //ItemByIdAndLanguage method.
    SystemMessage msg = default(SystemMessage);
    msg = srv.SystemMessages.ItemByIdAndLanguage(14126, "us_english");
    //Display the message ID and text.
    Console.WriteLine(msg.ID.ToString() + " " + msg.Text);
}
```

Finding a Particular System Message in PowerShell

The code example shows how to identify a system message by ID number and display the message.

```
# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\
$srv = get-item default

#Get the message 14126 in US English and display it
$msg = $srv.SystemMessages.ItemByIdAndLanguage(14126, "us_english")
$msg.ID.ToString() + " " + $msg.Text
```

Adding a New User-Defined Message in Visual Basic

The code example demonstrates how to create a user-defined message with an ID greater than 50000.

```
Dim mysrv As Server
mysrv = New Server
Dim udm As UserDefinedMessage
udm = New UserDefinedMessage(mysrv, 50003, "us_english", 16, "Test message")
udm.Create()
```

Adding a New User-Defined Message in Visual C#

The code example demonstrates how to create a user-defined message with an ID greater than 50000.

```
{

    Server mysrv = new Server();

    UserDefinedMessage udm = new UserDefinedMessage(mysrv, 50030, "us_english",16, "Test message");
    udm.Create();
    UserDefinedMessage msg = mysrv.UserDefinedMessages.ItemByIdAndLanguage(50030, "us_english");
    //Display the message ID and text.
    Console.WriteLine(msg.ID.ToString() + " " + msg.Text);

}
```

Adding a New User-Defined Message in PowerShell

The code example demonstrates how to create a user-defined message with an ID greater than 50000.





```
#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#Create a new message

$udm = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedMessage -argumentlist `
$srv, 50030, "us_english", 16, "Test message"
$udm.Create()
$msg = $srv.UserDefinedMessages.ItemByIdAndLanguage(50030, "us_english");
$msg
```

Using Synonyms

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

A synonym is an alternative name for a schema-scoped object. In SMO, synonyms are represented by the [Synonym](#) object. The [Synonym](#) object is a child of the [Database](#) object. This means that synonyms are valid only within the scope of the database in which they are defined. However, the synonym can refer to objects on another database, or on a remote instance of SQL Server.

The object that is given an alternative name is known as the base object. The name property of the [Synonym](#) object is the alternative name given to the base object.

Example

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a Synonym in Visual C#

The code example shows how to create a synonym or an alternate name for a schema scoped object. Client applications can use a single reference for the base object via a synonym instead of using a multiple part name to reference the base object.

```
{  
    //Connect to the local, default instance of SQL Server.  
    Server srv = new Server();  
  
    //Reference the AdventureWorks2012 database.  
    Database db = srv.Databases["AdventureWorks2012"];  
  
    //Define a Synonym object variable by supplying the  
    //parent database, name, and schema arguments in the constructor.  
    //The name is also a synonym of the name of the base object.  
    Synonym syn = new Synonym(db, "Shop", "Sales");  
  
    //Specify the base object, which is the object on which  
    //the synonym is based.  
    syn.BaseDatabase = "AdventureWorks2012";  
    syn.BaseSchema = "Sales";  
    syn.BaseObject = "Store";  
    syn.BaseServer = srv.Name;  
  
    //Create the synonym on the instance of SQL Server.  
    syn.Create();  
}
```

Creating a Synonym in PowerShell

The code example shows how to create a synonym or an alternate name for a schema scoped object. Client applications can use a single reference for the base object via a synonym instead of using a multiple part name to reference the base object.

```
#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server

#And the database object corresponding to Adventureworks
$db = $srv.Databases["AdventureWorks2012"]

$syn = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Synonym `
-argumentlist $db, "Shop", "Sales"

#Specify the base object, which is the object on which the synonym is based.
$syn.BaseDatabase = "AdventureWorks2012"
$syn.BaseSchema = "Sales"
$syn.BaseObject = "Store"
$syn.BaseServer = $srv.Name





#Create the synonym on the instance of SQL Server.
$syn.Create()
```

See Also

[CREATE SYNONYM \(Transact-SQL\)](#)

Using Table and Index Partitioning

11/16/2017 • 3 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

Data can be stored by using the storage algorithms provided by [Partitioned Tables and Indexes](#). Partitioning can make large tables and indexes more manageable and scalable.

Index and Table Partitioning

The feature enables index and table data to be spread across multiple file groups in partitions. A partition function defines how the rows of a table or index are mapped to a set of partitions based on the values of certain columns, referred to as partitioning columns. A partition scheme maps each partition specified by the partition function to a file group. This lets you develop archiving strategies that enable tables to be scaled across file groups, and therefore physical devices.

The [Database](#) object contains a collection of [PartitionFunction](#) objects that represent the implemented partition functions and a collection of [PartitionScheme](#) objects that describe how data is mapped to file groups.

Each [Table](#) and [Index](#) object specifies which partition scheme it uses in the [PartitionScheme](#) property and specifies the columns in the [PartitionSchemeParameterCollection](#).

Example

For the following code examples, you will have to select the programming environment, programming template and the programming language to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Setting Up a Partition Scheme for a Table in Visual C#

The code example shows how to create a partition function and a partition scheme for the `TransactionHistory` table in the AdventureWorks2012 sample database. The partitions are divided by date with the intention of separating out old records into the `TransactionHistoryArchive` table.

```

{
//Connect to the local, default instance of SQL Server.
Server srv;
srv = new Server();
//Reference the AdventureWorks2012 database.
Database db;
db = srv.Databases("AdventureWorks2012");
//Define and create three new file groups on the database.
FileGroup fg2;
fg2 = new FileGroup(db, "Second");
fg2.Create();
FileGroup fg3;
fg3 = new FileGroup(db, "Third");
fg3.Create();
FileGroup fg4;
fg4 = new FileGroup(db, "Fourth");
fg4.Create();
//Define a partition function by supplying the parent database and name arguments in the constructor.
PartitionFunction pf;
pf = new PartitionFunction(db, "TransHistPF");
//Add a partition function parameter that specifies the function uses a DateTime range type.
PartitionFunctionParameter pfp;
pfp = new PartitionFunctionParameter(pf, DataType.DateTime);
pf.PartitionFunctionParameters.Add(pfp);
//Specify the three dates that divide the data into four partitions.
object[] val;
val = new object[] { "1/1/2003", "1/1/2004", "1/1/2005" };
pf.RangeValues = val;
//Create the partition function.
pf.Create();
//Define a partition scheme by supplying the parent database and name arguments in the constructor.
PartitionScheme ps;
ps = new PartitionScheme(db, "TransHistPS");
//Specify the partition function and the filegroups required by the partition scheme.
ps.PartitionFunction = "TransHistPF";
ps.FileGroups.Add("PRIMARY");
ps.FileGroups.Add("second");
ps.FileGroups.Add("Third");
ps.FileGroups.Add("Fourth");
//Create the partition scheme.
ps.Create();
}

```

Setting Up a Partition Scheme for a Table in PowerShell

The code example shows how to create a partition function and a partition scheme for the `TransactionHistory` table in the AdventureWorks2012 sample database. The partitions are divided by date with the intention of separating out old records into the `TransactionHistoryArchive` table.

```

# Set the path context to the local, default instance of SQL Server.
CD \sql\localhost\default

#Get a server object which corresponds to the default instance
$srv = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Server
$db = $srv.Databases["AdventureWorks"]
#Create four filegroups
$fg1 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Filegroup -argumentlist $db, "First"
$fg2 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Filegroup -argumentlist $db, "Second"
$fg3 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Filegroup -argumentlist $db, "Third"
$fg4 = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Filegroup -argumentlist $db, "Fourth"

#Define a partition function by supplying the parent database and name arguments in the constructor.
$pf = New-Object -TypeName Microsoft.SqlServer.Management.SMO.PartitionFunction -argumentlist $db,
"TransHistPF"
$T = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime
$T
$T.GetType()
#Add a partition function parameter that specifies the function uses a DateTime range type.
$pfp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.PartitionFunctionParameter -argumentlist $pf,
$T

#Specify the three dates that divide the data into four partitions.
#Create an array of type object to hold the partition data
$val = "1/1/2003"."1/1/2004"."1/1/2005"
$pf.RangeValues = $val
$pf
#Create the partition function
$pf.Create()

#Create partition scheme
$ps = New-Object -TypeName Microsoft.SqlServer.Management.SMO.PartitionScheme -argumentlist $db, "TransHistPS"
$ps.PartitionFunction = "TransHistPF"

#add the filegroups to the scheme
$ps.FileGroups.Add("PRIMARY")
$ps.FileGroups.Add("Second")
$ps.FileGroups.Add("Third")
$ps.FileGroups.Add("Fourth")

#Create it at the server
$ps.Create()





```

See Also

[Partitioned Tables and Indexes](#)

Using User-Defined Tables

11/16/2017 • 5 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

User-defined tables represent tabular information. They are used as parameters when you pass tabular data into stored procedures or user-defined functions. User-defined tables cannot be used to represent columns in a database table.

The [Database](#) object has a [UserDefinedTableTypes](#) property that references a [UserDefinedTableTypeCollection](#) object. Each [UserDefinedTableType](#) object in that collection has a **Columns** property that refers to a collection of [Column](#) objects that list the columns in the user-defined table. Use the Add method to add columns to the user-defined table.

When you define a new user-defined table by using the [UserDefinedTableType](#) object, you will have to supply columns and a primary key based on one of the columns.

User-defined table types cannot be altered after they are created. The [UserDefinedTableType](#) does not support the Alter method. User-defined table types can have check constraints, but some check operations will throw an exception because the type is not alterable.

The [DataType](#) class is used to specify the data type that is associated with columns and parameters. Use this type to specify the user-defined table type as a parameter for user-defined functions and stored procedures.

Examples

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating a User Defined Table in Visual Basic

For this example, you will have to include an imports statement for the class library that contains the **StringCollection** type.

```
Imports System.Collections.Specialized
```

The example demonstrates how to create a user-defined table, and then how to use it as a parameter in a user-defined function.


```

'Connect to the local, default instance of SQL Server
Dim srv As Server
srv = New Server
'Reference the AdventureWorks2012 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define a UserDefinedTableType object variable by supplying the 'database and name in the constructor.
Dim udt As UserDefinedTableType
udt = New UserDefinedTableType(db, "My_User_Defined_Table")
'Add three columns of different types to the
'UserDefinedTableType object.
udt.Columns.Add(New Column(udt, "Col1", DataType.Int))
udt.Columns.Add(New Column(udt, "Col2", DataType.VarCharMax))
udt.Columns.Add(New Column(udt, "Col3", DataType.Money))
'Define an Index object variable by supplying the user-defined
'table variable and name in the constructor.
Dim idx As Index
idx = New Index(udt, "PK_UdtTable")
'Add the first column in the user-defined table as
'the indexed column.
idx.IndexedColumns.Add(New IndexedColumn(idx, "Col1"))
'Specify that the index is a clustered, unique, primary key.
idx.IsClustered = True
idx.IsUnique = True
idx.IndexKeyType = IndexKeyType.DriPrimaryKey
udt.Indexes.Add(idx)
'Add the index and create the user-defined table.
udt.Create()
'Display the Transact-SQL creation script for the
'user-defined table.
Dim sc As StringCollection
sc = udt.Script()
For Each c As String In sc
    Console.WriteLine(c)
Next

'Define a new user-defined function with a single parameter.
Dim udf As UserDefinedFunction
udf = New UserDefinedFunction(db, "My_User_Defined_Function")
udf.TextMode = False
udf.FunctionType = UserDefinedFunctionType.Scalar
udf.ImplementationType = ImplementationType.TransactSql
udf.DataType = DataType.DateTime
'Specify the parameter as a UserDefinedTableType object.
Dim udfp As UserDefinedFunctionParameter
udfp = New UserDefinedFunctionParameter(udf, "@param")
udfp.DataType = New DataType(udt)
udfp.IsReadOnly = True
udf.Parameters.Add(udfp)
'Specify the TextBody property to the Transact-SQL definition of the
'user-defined function.
udf.TextBody = "BEGIN RETURN (GETDATE());end"
'Create the user-defined function.
udf.Create()

```

Creating a User Defined Table in Visual C#

For this example, you will have to include an imports statement for the class library that contains the **StringCollection** type.

```
using System.Collections.Specialized;
```

The example shows how to create a user-defined table, and then how to use it as a parameter in a user-defined function.

```

{
    //Connect to the local, default instance of SQL Server
    Server srv = new Server();

    //Reference the AdventureWorks2012 database.
    Database db = srv.Databases["AdventureWorks2012"];
    //Define a UserDefinedTableType object variable by supplying the
    //database and name in the constructor.
    UserDefinedTableType udt = new UserDefinedTableType(db, "My_User_Defined_Table");

    //Add three columns of different types to the
    //UserDefinedTableType object.
    udt.Columns.Add(new Column(udt, "Col1", DataType.Int));
    udt.Columns.Add(new Column(udt, "Col2", DataType.VarCharMax));
    udt.Columns.Add(new Column(udt, "Col3", DataType.Money));

    //Define an Index object variable by supplying the user-defined
    //table variable and name in the constructor.

    Index idx = new Index(udt, "PK_UdtTable");

    //Add the first column in the user-defined table as
    //the indexed column.
    idx.IndexedColumns.Add(new IndexedColumn(idx, "Col1"));
    //Specify that the index is a clustered, unique, primary key.
    idx.IsClustered = true;
    idx.IsUnique = true;
    idx.IndexKeyType = IndexKeyType.DriPrimaryKey;
    udt.Indexes.Add(idx);
    //Add the index and create the user-defined table.
    udt.Create();

    //Display the Transact-SQL creation script for the
    //user-defined table.
    System.Collections.Specialized.StringCollection sc;
    sc = udt.Script();
    foreach (string s in sc)
    {
        Console.WriteLine(s);
    }

    //Define a new user-defined function with a single parameter.
    UserDefinedFunction udf = new UserDefinedFunction(db, "My_User_Defined_Function");
    udf.TextMode = false;
    udf.FunctionType = UserDefinedFunctionType.Scalar;
    udf.ImplementationType = ImplementationType.TransactSql;
    udf.DataType = DataType.DateTime;

    //Specify the parameter as a UserDefinedTableType object.
    UserDefinedFunctionParameter udfp = new UserDefinedFunctionParameter(udf, "@param");
    udfp.DataType = new DataType(udt);
    udfp.IsReadOnly = true;
    udf.Parameters.Add(udfp);

    //Specify the TextBody property to the Transact-SQL definition of the
    //user-defined function.
    udf.TextBody = "BEGIN RETURN (GETDATE());end";
    //Create the user-defined function.
    udf.Create();
}

```

Creating a User Defined Table in PowerShell

For this example, you will have to include an imports statement for the class library that contains the **StringCollection** type.

```
using System.Collections.Specialized;
```

The example shows how to create a user-defined table, and then how to use it as a parameter in a user-defined function.

```
# Set the path context to the local, default instance of SQL Server and get a reference to AdventureWorks2012
CD \sql\localhost\default\databases
$db = get-item Adventureworks2012

#Define a UserDefinedTableType object variable by supplying the
#database and name in the constructor.
$udtt = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedTableType `
-argumentlist $db, "My_User_Defined_Table"

#Add three columns of different types to the UserDefinedTableType object.

$type = [Microsoft.SqlServer.Management.SMO.DataType]::Int
$col = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column `
-argumentlist $udtt, "col1", $type
$udtt.Columns.Add($col)

$type = [Microsoft.SqlServer.Management.SMO.DataType]::VarCharMax
$col = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column `
-argumentlist $udtt, "col2", $type
$udtt.Columns.Add($col)

$type = [Microsoft.SqlServer.Management.SMO.DataType]::Money
$col = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Column `
-argumentlist $udtt, "col3", $type
$udtt.Columns.Add($col)

#Define an Index object variable by supplying the user-defined
#table variable and name in the constructor.
$idx = New-Object -TypeName Microsoft.SqlServer.Management.SMO.Index `
-argumentlist $udtt, "PK_UddtTable"

#Add the first column in the user-defined table as
#the indexed column.
$idxcol = New-Object -TypeName Microsoft.SqlServer.Management.SMO.IndexedColumn `
-argumentlist $idx, "Col1"
$idx.IndexedColumns.Add($idxcol)

#Specify that the index is a clustered, unique, primary key.
$idx.IsClustered = $true
$idx.IsUnique = $true
$idx.IndexKeyType = [Microsoft.SqlServer.Management.SMO.IndexKeyType]::DriPrimaryKey;

#Add the index and create the user-defined table.
$udtt.Indexes.Add($idx)
$udtt.Create();

# Display the Transact-SQL creation script for the
# user-defined table.
$sc = $udtt.Script()
$sc

# Define a new user-defined function with a single parameter.
$udf = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedFunction `
-argumentlist $db, "My_User_Defined_Function"
$udf.TextMode = $false
$udf.FunctionType = [Microsoft.SqlServer.Management.SMO.UserDefinedFunctionType]::Scalar
$udf.ImplementationType = [Microsoft.SqlServer.Management.SMO.ImplementationType]::TransactSql
$udf.DataType = [Microsoft.SqlServer.Management.SMO.DataType]::DateTime

# Specify the parameter as a UserDefinedTableTable object.
$udfp = New-Object -TypeName Microsoft.SqlServer.Management.SMO.UserDefinedFunctionParameter `
-argumentlist $udf, "@param"
```

```
$type      = New-Object -TypeName Microsoft.SqlServer.Management.SMO.DataType
-argumentlist $udtt
$udfp.DataType = $type
$udfp.IsReadOnly = $true
$udf.Parameters.Add($udfp)

# Specify the TextBody property to the Transact-SQL definition of the
# user-defined function.
$udf.TextBody = "BEGIN RETURN (GETDATE());end"

# Create the user-defined function.
$udf.Create()
```





See Also

[FileGroup](#)

[Database Files and Filegroups](#)

Using XML Schemas

11/16/2017 • 2 min to read • [Edit Online](#)

THIS TOPIC APPLIES TO:  SQL Server  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

XML programming in SMO is limited to providing XML data types, XML namespaces, and simple indexing on XML data type columns.

Microsoft SQL Server provides native storage for XML document instances. XML schemas let you define complex XML data types, which can be used to validate XML documents to ensure data integrity. The XML schema is defined in the [XmlSchemaCollection](#) object.

Example

To use any code example that is provided, you will have to choose the programming environment, the programming template, and the programming language in which to create your application. For more information, see [Create a Visual C# SMO Project in Visual Studio .NET](#).

Creating an XML Schema in Visual Basic

This code example shows how to create an XML schema by using the [XmlSchemaCollection](#) object. The [Text](#) property, which defines the XML schema collection, contains several double quotation marks. These are replaced by the `chr(34)` string.

```
'Connect to the local, default instance of SQL Server.
Dim srv As Server
srv = New Server()
'Reference the AdventureWorks2012 2008R2 database.
Dim db As Database
db = srv.Databases("AdventureWorks2012")
'Define an XmlSchemaCollection object by supplying the parent database and name arguments in the constructor.
Dim xsc As XmlSchemaCollection
xsc = New XmlSchemaCollection(db, "MySampleCollection")
xsc.Text = "<\schema xmlns=" + Chr(34) + "http://www.w3.org/2001/XMLSchema" + Chr(34) + " xmlns:ns=" +
Chr(34) + "http://ns" + Chr(34) + ">\<element name=" + Chr(34) + "e" + Chr(34) + " type=" + Chr(34) +
"dateTime" + Chr(34) + "/></schema>"
'Create the XML schema collection on the instance of SQL Server.
xsc.Create()
```

Creating an XML Schema in Visual C#

This code example shows how to create an XML schema by using the [XmlSchemaCollection](#) object. The [Text](#) property, which defines the XML schema collection, contains several double quotation marks. These are replaced by the `chr(34)` string.

```

{
    //Connect to the local, default instance of SQL Server.
    Server srv = default(Server);
    srv = new Server();
    //Reference the AdventureWorks2012 database.
    Database db = default(Database);
    db = srv.Databases["AdventureWorks2012"];
    //Define an XmlSchemaCollection object by supplying the parent
    // database and name arguments in the constructor.
    XmlSchemaCollection xsc = default(XmlSchemaCollection);
    xsc = new XmlSchemaCollection(db, "MySampleCollection");
    xsc.Text = "<schema xmlns=" + Strings.Chr(34) + "http://www.w3.org/2001/XMLSchema" +
Strings.Chr(34) + " xmlns:ns=" + Strings.Chr(34) + "http://ns" + Strings.Chr(34) + "><element name=" +
Strings.Chr(34) + "e" + Strings.Chr(34) + " type=" + Strings.Chr(34) + "dateTime" + Strings.Chr(34) + "/>
</schema>";
    //Create the XML schema collection on the instance of SQL Server.
    xsc.Create();
}

```

Creating an XML Schema in PowerShell

This code example shows how to create an XML schema by using the [XmlSchemaCollection](#) object. The [Text](#) property, which defines the XML schema collection, contains several double quotation marks. These are replaced by the `chr(34)` string.

```

#Get a server object which corresponds to the default instance replace LocalMachine with the physical server
cd \sql\LocalHost
$srv = get-item default

#Reference the AdventureWorks database.
$db = $srv.Databases["AdventureWorks2012"]

#Create a new schema collection
$xsc = New-Object -TypeName Microsoft.SqlServer.Management.SMO.XmlSchemaCollection `
    -argumentlist $db,"MySampleCollection"

#Add the xml
$dq = '"' # the double quote character
$xsc.Text = "<schema xmlns=" + $dq + "http://www.w3.org/2001/XMLSchema" + $dq + `
"  xmlns:ns=" + $dq + "http://ns" + $dq + "><element name=" + $dq + "e" + $dq + `
" type=" + $dq + "dateTime" + $dq + "/></schema>"

#Create the XML schema collection on the instance of SQL Server.
$xsc.Create()

```