**如何在 Visual Studio .NET 中创建 Visual C# SMO 项目**

本节介绍了如何生成简单的 SMO 控制台应用程序。

此示例导入命名空间，这样，程序即可以引用 SMO 类型。可以选择导入 **Agent** 命名空间。当编写使用 SQL Server 代理的程序时使用此命名空间。需要使用 **Common** 命名空间来建立与 SQL Server 实例的安全连接。使用 **SqlClient** 命名空间处理 SQL 异常错误。

**http://i.msdn.microsoft.com/Global/Images/clear.gif  在 Visual Studio .NET 中创建 Visual C# SMO 项目**

1. 启动 Visual Studio 2008（或 Visual Studio 2005）。
2. 在**“文件”**菜单上，单击**“新建项目”**。此时将显示**“新建项目”**对话框。
3. 在**“项目类型”**对话框中，选择**“Visual C#”**，然后选择**“Windows”**。在“Visual Studio 已安装的模板”窗格中，选择**“Windows 应用程序”**。
4. （可选）在**“名称”**字段中，键入新应用程序的名称。
5. 选择 Visual C# 应用程序类型。在下面的示例中，请选择**“控制台应用程序”**。
6. 在**“项目”**菜单上，选择**“添加引用”**。此时将显示**“添加引用”**对话框。
7. 单击**“浏览”**，在 C:\Program Files\Microsoft SQL Server\100\SDK\Assemblies\ 文件夹中找到 SMO 程序集，然后选择下列文件。这些文件是构建一个 SMO 应用程序至少需要的文件：

Microsoft.SqlServer.ConnectionInfo.dll

Microsoft.SqlServer.Smo.dll

Microsoft.SqlServer.Management.Sdk.Sfc.dll

Microsoft.SqlServer.SqlEnum.dll

|  |
| --- |
| **ms162129.note(zh-cn,SQL.105).gif注意：** |
| 使用 **Ctrl** 键可选择多个文件。 |

1. 添加需要的任何其他 SMO 程序集。例如，如果您要专门对 Service Broker 进行编程，则可以添加以下程序集：

Microsoft.SqlServer.ServiceBrokerEnum.dll

1. 单击**“打开”**。
2. 在**“视图”**菜单中，单击**“代码”**。或者选择“Program1.cs [设计] 窗口”，然后双击 Windows 窗体以显示代码窗口。
3. 在代码的命名空间语句前，键入以下 **using** 语句，以限定 SMO 命名空间中的类型：
4. using Microsoft.SqlServer.Management.Smo;

using Microsoft.SqlServer.Management.Common;

1. SMO 在 Microsoft.SqlServer.Management.Smo 下具有各种命名空间，如 Microsoft.SqlServer.Management.Smo.Agent。请根据需要添加这些命名空间。
2. 您可以立即添加 SMO 代码。
3. **Overview**
4. In this series of articles, I'll demonstrate how to use SMO (SQL Management Objects) to do a variety of common DBA tasks including Backups, Restores,Index Maintenance, Integrity checks and more. These are some of the operations available in the SMO database maintenance utility available on this site - [**ExpressMaint.**](http://www.sqldbatips.com/showarticle.asp?ID=29) In this article we will concentrate on how to build a SMO application and how to connect to a SQL Server and retrieve some server properties. Example code will use VB.NET and C#. For those of you that don't have access to Visual Studio 2005 I'll include the command line compiler commands.. I'd also highly recommend the SMO sample applications that are part of the SQL Server Engine samples that come with SQL2005 and are also available for download [**here.**](http://www.codeplex.com/MSFTEngProdSamples/Release/ProjectReleases.aspx?ReleaseId=4038)
5. **Building SMO applications**
6. In order to use the SMO objects in a .NET application we need to add a reference to the SMO assemblies. The easiest way to do this in Visual Studio 2005 is to choose Project>Add Reference from the main menu and add select the following assemblies

|  |  |
| --- | --- |
|  Microsoft.SqlServer.ConnectionInfo   Microsoft.SqlServer.Smo   Microsoft.SqlServer.SmoEnum   Microsoft.SqlServer.SqlEnum | http://www.sqldbatips.com/images/smoref.gif |

1. There are a number of other assemblies that need to be referenced for certain specific tasks such as managing Service Broker but these core assemblies contain all the fuctionality required for most basic administrative operations. See the SMO Programming reference in Books On Line for more details. If you don't have access to Visual Studio 2005, simply add references to the required assemblies on the compiler command line when compiling applications. An example commandline for the VB.NET and C# compilers is shown below (these can be found in the C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx folder) assuming a console application.
2. **Note that the entire command should be on one line even though the examples span multiple lines for formatting purposes**

|  |
| --- |
| [Visual Basic]    vbc /t:exe c:\SMOtest.vb /r:  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.ConnectionInfo.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.Smo.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.SmoEnum.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.SqlEnum.dll"  [C#]  csc /t:exe /out:c:\SMOTest.exe c:\SMOtest.vb /r:  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.ConnectionInfo.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.Smo.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.SmoEnum.dll",  "C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.SqlEnum.dll" |

1. **Getting Connected**
2. Getting connected to SQL Server could not be simpler. For a default instance connecting with Windows Authentication, we simply need to create a new SMO Server object as shown in the example below and we can begin obtaining information about the server such as the servername and version string.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Console.Writeline(svr.Name & " " & svr.Information.VersionString)  End Sub  End Module  [C#]  using System;  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Console.WriteLine(svr.Name + " " + svr.Information.VersionString);  }  }  } |

1. The same effect can be achieved by passing the server or server\instance in the Server constructor e.g.

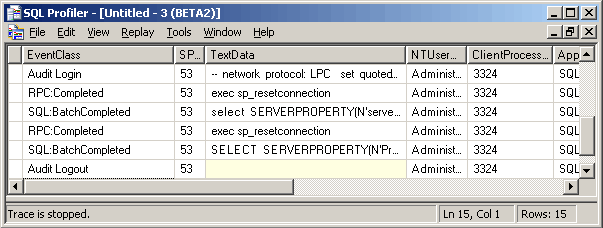
|  |
| --- |
| Dim svr As Server = New Server("(local)")  Server svr = new Server(@"(local)\INSTANCE01") |

1. However, if we want to specify more complex connection options or control the connection pooling behaviour of SMO, we can use the ConnectionContext object. This is a child object property of the Server object so we can specify these options after creating the Server object or by declaring a local variable as a ServerConnection object and passing it to the Server constructor. Using this we can specify such things as the authentication mode and username and password e.g.

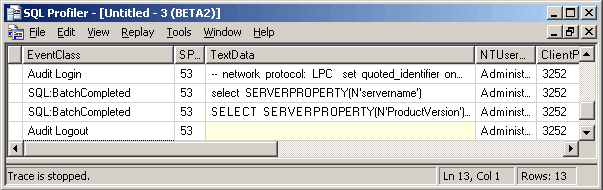
|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Imports Microsoft.SqlServer.Management.Common  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  svr.ConnectionContext.LoginSecure = false  svr.ConnectionContext.Login = "username"  svr.ConnectionContext.Password = "password"  Console.Writeline(svr.Name & " " & svr.Information.VersionString)  End Sub  End Module  [C#]  using System;  using Microsoft.SqlServer.Management.Smo;  using Microsoft.SqlServer.Management.Common;  namespace SMOTest  {  class Program  {  static void Main()  {  ServerConnection conn = new ServerConnection();  conn.LoginSecure = false;  conn.Login = "username";  conn.Password = "password";  Server svr = new Server(conn);  Console.WriteLine(svr.Name + " " + svr.Information.VersionString);  }  }  } |

1. The default behaviour of SMO is to use connection pooling. Connections will be established and released as required. However, this behaviour can be changed by either setting the AutoDisconnectMode property of the ConnectionContext to AutoDisconnectMode.NoAutoDisconnect or by setting the NonPooledConnection property to true. If AutoDisconnectMode is set to NoAutoDisconnect and Connect is explicitly called then the connection is not returned to the pool until Disconnect is called. By setting the NonPooledConnection property to true we are explictily requesting a non pooled connection. The behaviour can be observed by using Profiler to monitor the SMO application as shown in the VB.NET code snippet below.
2. **Default behaviour - pooled connection**

|  |
| --- |
| Dim svr As Server = New Server()  Console.Writeline(svr.Name & " " & svr.Information.VersionString) |

1. 
2. **Non pooled connection**

|  |
| --- |
| Dim svr As Server = New Server()  svr.ConnectionContext.NonPooledConnection = True  svr.ConnectionContext.Connect()  Console.Writeline(svr.Name & " " & svr.Information.VersionString)  svr.ConnectionContext.Disconnect() |

1. 
2. In the next few articles we will dive deeper into SMO and use it to perform some common DBA tasks such as Backup, Restore and Index maintenance. We will also look at how we can use one of the new features of SMO - Partial Instantiation - to optimize SMO applications.

**Overview**

In this series of articles, I'll demonstrate how to use SMO (SQL Management Objects) to do a variety of common DBA tasks including Backups, Restores,Index Maintenance, Integrity checks and more. These are some of the operations available in the SMO database maintenance utility available on this site - [**ExpressMaint.**](http://www.sqldbatips.com/showarticle.asp?ID=29) In this article we will concentrate on how to examine database integrity using SMO in SQL2005. For information on how to build and compile a SMO application please review [**Getting Started with SMO in SQL 2005**](http://www.sqldbatips.com/showarticle.asp?ID=34).

**Checking Database Integrity**

Integrity checks in SMO are methods of the Database object. The table below shows the SMO method and equivalent TSQL command.

|  |  |
| --- | --- |
| **SMO Method** | **TSQL Command** |
| CheckAllocations(RepairType.None) | DBCC CHECKALLOC WITH NO\_INFOMSGS |
| CheckAllocationsDataOnly() | DBCC CHECKALLOC(N'databasename', NOINDEX) |
| CheckCatalog() | DBCC CHECKCATALOG |
| CheckTables(RepairType.None) | DBCC CHECKDB WITH NO\_INFOMSGS |
| CheckTablesDataOnly() | DBCC CHECKDB(N'databasename', NOINDEX) |

All the methods in the table above return a StringCollection. We can iterate through this collection to return the results of the integrity check as demonstrated in the following code samples.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Imports System.Collections.Specialized  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim sc As StringCollection  Dim db As Database  db = svr.Databases("AdventureWorks")  sc = db.CheckTables(RepairType.None)  For c As Integer = 0 To (sc.Count - 1)  If sc(c).Length > 0 Then  Console.WriteLine(sc(c))  End If  Next    End Sub  End Module  [C#]  using System;  using Microsoft.SqlServer.Management.Smo;  using System.Collections.Specialized;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Database db;  db = svr.Databases["AdventureWorks"];  StringCollection sc;  sc = db.CheckCatalog();  for (int i = 0; i < sc.Count; i++)  {  if (sc[i].Length > 0)  Console.WriteLine(sc[i]);  }  }  }  } |

**Overview**

In this series of articles, I'll demonstrate how to use SMO (SQL Management Objects) to do a variety of common DBA tasks including Backups, Restores,Index Maintenance, Integrity checks and more. These are some of the operations available in the SMO database maintenance utility available on this site - [**ExpressMaint.**](http://www.sqldbatips.com/showarticle.asp?ID=29) In this article we will concentrate on how to perform backups using SMO in SQL2005. For information on how to build and compile a SMO application please review [**Getting Started with SMO in SQL 2005**](http://www.sqldbatips.com/showarticle.asp?ID=34).

**Database Backups**

In order to perform a backup using SMO we require two objects, a Server object and a Backup object. In its simplest form, a Backup object requires only a few properties to be set before calling the SqlBackup method and passing in the Server object as can be seen in the following example that does a Full database backup of the SMO database to the file c:\SMOTest.bak

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim bkp As Backup = New Backup()  bkp.Devices.AddDevice("C:\SMOTest.bak", DeviceType.File)  bkp.Database = "SMO"  bkp.Action = BackupActionType.Database  bkp.Initialize = True  bkp.PercentCompleteNotification = 10  AddHandler bkp.PercentComplete, AddressOf ProgressEventHandler  bkp.SqlBackup(svr)  End Sub  Private Sub ProgressEventHandler(ByVal sender As Object, \_  ByVal e As PercentCompleteEventArgs)  Console.WriteLine(e.Percent.ToString + "% backed up")  End Sub  End Module  [C#]  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Backup bkp = new Backup();  bkp.Devices.AddDevice(@"C:\SMOTest.bak", DeviceType.File);  bkp.Database = "SMO";  bkp.Action = BackupActionType.Database;  bkp.Initialize = true;  bkp.PercentCompleteNotification = 10;  bkp.PercentComplete += new PercentCompleteEventHandler(bkp\_PercentComplete);  bkp.SqlBackup(svr);  }  static void bkp\_PercentComplete(object sender, PercentCompleteEventArgs e)  {  Console.WriteLine(e.Percent.ToString() + "% backed up");  }  }  } |

There are many other properties that can be set for the Backup object. For the full list see the Backup Object in Books On Line (if you have SQL2005 Books On Line installed [**click this link**](ms-help://MS.SQLCC.v9/MS.SQLSVR.v9.en/smo9mref/html/af02fd7d-8008-bad9-724b-b7f7abe7961b.htm) to be taken to the correct page).

**Filegroup Backups**

In the example below, just the WorkOrderGroup filegroup of the Adventureworks database is backed up (the script to convert the AdventureWorks OLTP demo database into a multi filegroup database can be found in the SQL Server Engine samples and by default is located in C:\Program Files\Microsoft SQL Server\90\Samples\Engine\Administration\FileGroups\Scripts). Filegroups to be backed up are specified by adding them to the Backup objects DatabaseFileGroups collection. In order to backup specific files, they can be added to the DatabaseFiles collection.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim bkp As Backup = New Backup()  bkp.Database = "AdventureWorks"  bkp.Action = BackupActionType.Files  bkp.DatabaseFileGroups.Add("WorkOrderGroup")  bkp.Devices.AddDevice("C:\AWFGTest.bak", DeviceType.File)  bkp.SqlBackup(svr)  End Sub  End Module  [C#]  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Backup bkp = new Backup();  bkp.Database = "AdventureWorks";  bkp.Action = BackupActionType.Files;  bkp.DatabaseFileGroups.Add("WorkOrderGroup");  bkp.Devices.AddDevice(@"C:\AWFGTest.bak", DeviceType.File);  bkp.SqlBackup(svr);  }  }  } |

**Differential Backups**

Performing a differential backup is very straightforward, simply set the Incremental property of the Backup object to True

|  |
| --- |
| bkp.Incremental = True |

**Log Backups**

Log backups are equally straightforward, simply set the Action property of the Backup object to BackupActionType.Log

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim bkp As Backup = New Backup()  bkp.Action = BackupActionType.Log  bkp.Database = "SMO"  bkp.Devices.AddDevice("c:\SMOTest.trn", DeviceType.File)  bkp.SqlBackup(svr)  End Sub  End Module  [C#]  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Backup bkp = new Backup();  bkp.Action = BackupActionType.Log;  bkp.Database = "SMO";  bkp.Devices.AddDevice(@"c:\SMOTest.trn", DeviceType.File);  bkp.SqlBackup(svr);  }  }  } |

**Overview**

In this series of articles, I'll demonstrate how to use SMO (SQL Management Objects) to do a variety of common DBA tasks including Backups, Restores,Index Maintenance, Integrity checks and more. These are some of the operations available in the SMO database maintenance utility available on this site - [**ExpressMaint.**](http://www.sqldbatips.com/showarticle.asp?ID=29) In this article we will concentrate on how to perform restores using SMO in SQL2005. For information on how to build and compile a SMO application please review [**Getting Started with SMO in SQL 2005**](http://www.sqldbatips.com/showarticle.asp?ID=34).

**Database Restores**

In order to perform a restore using SMO we require two objects, a Server object and a Restore object. In its simplest form, a Restore object requires only a few properties to be set before calling the SqlRestore method and passing in the Server object as can be seen in the following example that does a Full database restore of the SMO database from the file c:\SMOTest.bak, replacing the database it it already exists. This example also uses an Event Handler for the PercentComplete event to display the restore progress.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim res As Restore = New Restore()  res.Devices.AddDevice("C:\SMOTest.bak", DeviceType.File)  res.Database = "SMO"  res.ReplaceDatabase = True  res.PercentCompleteNotification = 10  AddHandler res.PercentComplete, AddressOf ProgressEventHandler  res.SqlRestore(svr)  End Sub  Private Sub ProgressEventHandler(ByVal sender As Object, \_  ByVal e As PercentCompleteEventArgs)  Console.Writeline(e.Percent.ToString + "% restored")  End Sub  End Module  [C#]  using System;  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Restore res = new Restore();  res.Database = "SMO";  res.Action = RestoreActionType.Database;  res.Devices.AddDevice(@"C:\SMOTest.bak", DeviceType.File);  res.PercentCompleteNotification = 10;  res.ReplaceDatabase = true;  res.PercentComplete += new PercentCompleteEventHandler(ProgressEventHandler);  res.SqlRestore(svr);  }    static void ProgressEventHandler(object sender, PercentCompleteEventArgs e)  {  Console.WriteLine(e.Percent.ToString() + "% restored");  }  }  } |

There are other properties that can be set for the Restore object. For the full list see the Restore Object in Books On Line (if you have SQL2005 Books On Line installed [**click this link**](ms-help://MS.SQLCC.v9/MS.SQLSVR.v9.en/smo9mref/html/0516d21c-2910-eaba-eeba-28c8bd7e001b.htm) to be taken to the correct page).

**Restoring a Database to a New Location**

Using SMO, the equivalent of the T-SQL WITH MOVE syntax for restores is to use the RelocateFiles property of the Restore Object and the RelocateFile object. In the example below, we will restore a copy of the SMO database to a database called SMO2 with the data and log files on the C: drive. The RelocateFile constructor can take two parameters, the first is the logical filename and the second is the physical filename. This provides the mapping of where to move the files during the restore.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim res As Restore = New Restore()  res.Devices.AddDevice("C:\SMOTest.bak", DeviceType.File)  res.Database = "SMO2"  res.NoRecovery = False  res.ReplaceDatabase = True  res.RelocateFiles.Add(New RelocateFile("SMO", "c:\SMO2.mdf"))  res.RelocateFiles.Add(New RelocateFile("SMO\_Log", "c:\SMO2.ldf"))  res.SqlRestore(svr)  End Sub  End Module  [C#]  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Restore res = new Restore();  res.Database = "SMO2";  res.Action = RestoreActionType.Database;  res.Devices.AddDevice(@"C:\SMOTest.bak", DeviceType.File);  res.ReplaceDatabase = true;  res.RelocateFiles.Add(new RelocateFile("SMO", @"c:\SMO2.mdf"));  res.RelocateFiles.Add(new RelocateFile("SMO\_Log", @"c:\SMO2.ldf"));  res.SqlRestore(svr);  }  }  } |

**Reading Backup File Information**

There are a number of methods of the Restore object that can be used to obtain information about a backup device and the files it contains including ReadBackupHeader, ReadFileList and ReadMediaHeader. In the example below, we will use the ReadFileList method to obtain the list of Logical filenames on the disk device c:\SMOTest.bak and display them on the console.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Imports System.Data  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim res As Restore = New Restore()  Dim dt As DataTable  Dim foundrows As DataRow()    res.Devices.AddDevice("C:\SMOTest.bak", DeviceType.File)  dt = res.ReadFileList(svr)  foundrows = dt.Select(Nothing)  For Each dr As DataRow In foundrows  Console.WriteLine(dr("LogicalName").ToString())  Next  End Sub  End Module  [C#]  using System;  using System.Data;  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Restore res = new Restore();  DataTable dt;  DataRow[] foundrows;    res.Devices.AddDevice(@"C:\SMOTest.bak", DeviceType.File);  dt = res.ReadFileList(svr);    foundrows = dt.Select();    foreach (DataRow r in foundrows)  {  Console.WriteLine(r["LogicalName"].ToString());  }  }  }  } |

Since all these methods return a DataTable, we can eaily extract all the columns returned by each method by enumerating the Columns collection of the returned DataTable object as in the VB.NET example below.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Imports System.Data  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim res As Restore = New Restore()  Dim dt As DataTable  Dim foundrows As DataRow()  res.Devices.AddDevice("C:\SMOTest.bak", DeviceType.File)  dt = res.ReadFileList(svr)  foundrows = dt.Select(Nothing)  For Each col As DataColumn In dt.Columns  Console.Write(col.ColumnName + vbTab)  Next  Console.WriteLine()  For Each dr As DataRow In foundrows  For Each col As DataColumn In dt.Columns  Console.Write(dr(col.ColumnName).ToString() + vbTab)  Next  Console.WriteLine()  Next  End Sub  End Module |

**Log Restores**

Log restores are equally straightforward, simply set the Action property of the Restore object to RestoreActionType.Log. Additional properties can be set for Log backups such as ToPointInTime which allows recovery to a specific point in time.

|  |
| --- |
| [Visual Basic]  Imports Microsoft.SqlServer.Management.Smo  Module SMOTest  Sub Main()  Dim svr As Server = New Server()  Dim res As Restore = New Restore()  res.Devices.AddDevice("C:\SMOTest.trn", DeviceType.File)  res.Database = "SMO"  res.NoRecovery = False  res.Action = RestoreActionType.Log  res.SqlRestore(svr)  End Sub  End Module  [C#]  using System;  using Microsoft.SqlServer.Management.Smo;  namespace SMOTest  {  class Program  {  static void Main()  {  Server svr = new Server();  Restore res = new Restore();  res.Database = "SMO";  res.Action = RestoreActionType.Log;  res.Devices.AddDevice(@"C:\SMOTest.trn", DeviceType.File);  res.NoRecovery = false;  res.SqlRestore(svr);  }  }  } |

[**C#连接SMO及数据库操作（一）**](http://www.cnblogs.com/coolsoft/archive/2010/08/18/1802690.html)

using System;  
using System.Collections.Generic;  
using System.Text;  
using Microsoft.SqlServer.Management.Common;  
using Microsoft.SqlServer.Management;  
using Microsoft.SqlServer.Management.Smo;  
  
  
namespace ConsoleApplication1  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 //连接本地数据库  
 Microsoft.SqlServer.Management.Common.ServerConnection conn = new ServerConnection("localhost", "sa", "pass");  
 //连接远程数据库  
 //Microsoft.SqlServer.Management.Common.ServerConnection conn = new ServerConnection("tcp:192.168.18.10,pass", "sa", "pass");  
 Server s = new Server(conn);  
 Console.WriteLine("DataBaseCount:" + s.Databases.Count);  
  
 //创建数据库  
 Database db1 = new Database(s, "NewDataBaseName");  
 //db1.Create();  
 Console.WriteLine("DataBaseCount:" + s.Databases.Count);  
  
 //创建表  
 Database db = s.Databases["NewDataBaseName"];//引用数据库  
 Table tb = new Table(db, "NewTableName");//表名  
 Column c = new Column(tb, "CustomerId");//字段CustomerId  
 c.Identity = true;  
 c.IdentitySeed = 1;  
 c.DataType = DataType.Int;  
 c.Nullable = false;  
 tb.Columns.Add(c);  
 c = new Column(tb, "CustomerName");//字段CustomerName  
 c.DataType = DataType.VarChar(20);  
 c.Nullable = true;  
 tb.Columns.Add(c);  
 //tb.Create();//创建表  
   
 //创建储存过程  
 StoredProcedure sp = new StoredProcedure(db, "NewProcedure");  
 StoredProcedureParameter spp1 = new StoredProcedureParameter(sp, "@addId", DataType.Int);  
 sp.TextMode = false;  
 sp.Parameters.Add(spp1);  
 sp.TextBody = "select \* from newtableName where CustomerId=@addId";  
 //sp.Create();  
  
 //执行储存过程  
 db.ExecuteNonQuery("exec NewProcedure addId=1");  
  
 //删除数据库  
 Database db2 = s.Databases["NewDataBaseName"];  
 //db2.Drop();  
 Console.WriteLine("DataBaseCount:" + s.Databases.Count);  
  
 //删除表  
 Table tb1 = db.Tables["NewTableName"];  
 //tb1.Drop();  
  
 //删除储存过程  
 StoredProcedure sp2 = db.StoredProcedures["NewProcedure"];  
 //sp2.Drop();  
 }  
 }  
}

# Starting SQL Trace using C#

Everybody uses SQL Profiler tool to capture useful information against SQL Server. But is it possible to capture SQL Traces programmatically? Well, by and large there are two possible ways that I can think of. Firstly we can call TSQL stored procedues such as sp\_trace\_create in the program. Secondly, we can utilize Microsoft.SqlServer.Management.Trace classes (aka Trace Management Object (TMO)).

  Microsoft.SqlServer.Management.Trace is implemented in Microsoft.SqlServer.ConnectionInfoExtended.dll which is in the GAC. Typically you will needMicrosoft.SqlServer.ConnectionInfo.dll in order to establish SQL connection. So add references to two DLLs in your C# project. Now, here is a small sample that starts SQL Trace and saves the result to a trace file.

using System;

using System.IO;

using System.Reflection;

using Microsoft.SqlServer.Management.Common;

using Microsoft.SqlServer.Management.Trace;

namespace StartTrace

{

    class Program

    {

        static void Main(string[] args)

        {

            if (args.Length != 2)

            {

                string asmName = Assembly.GetExecutingAssembly().GetName().Name;

                Console.WriteLine("Usage: {0}.exe <serverName> <traceFile>", asmName);

                Console.WriteLine(@"   ex) {0}.exe TestServer test.trc", asmName);

                return;

            }

            string serverName = args[0];

            string traceFile = args[1];

            (new Program()).StartTrace(serverName, traceFile);

        }

        void StartTrace(string serverName, string traceFile)

        {

            // Sanity check

            if (string.IsNullOrEmpty(serverName))

                throw new ArgumentException("serverName is null or empty");

            if (string.IsNullOrEmpty(traceFile))

                throw new ArgumentException("traceFile is null or empty");

            if (File.Exists(traceFile))

            {

                Console.WriteLine("### WARNING ### '{0}' already exists. Press Q if you don't want to overwrite.", traceFile);

            }

            // Start TraceServer

            TraceServer trcSvr = new TraceServer();

            SqlConnectionInfo ci = new SqlConnectionInfo(serverName);

            string tdf = @"%ProgramFiles%\Microsoft SQL Server\100\Tools\Profiler\Templates\Microsoft SQL Server\100\Standard.tdf";

            tdf = Environment.ExpandEnvironmentVariables(tdf);

            trcSvr.InitializeAsReader(ci, tdf);

            // Press S to stop tracing, Q for quit.

            Console.WriteLine();

            Console.WriteLine("TRACE started at {0} : Server={1}, TraceFile={2}", DateTime.Now, serverName, traceFile);

            Console.WriteLine("Press 'S' to stop trace... Press 'Q' to quit...");

            while (true)

            {

                ConsoleKeyInfo key = Console.ReadKey();

                if (key.Key == ConsoleKey.Q)  // Quit

                {

                    trcSvr.Stop();

                    return;

                }

                if (key.Key == ConsoleKey.S)  // Stop trace

                {

                    trcSvr.Stop();

                    break;

                }

            }

            Console.WriteLine();

            // Write trace to .trc file

            TraceFile writer = new TraceFile();

            writer.InitializeAsWriter(trcSvr, traceFile);

            writer.WriteNotify += new WriteNotifyEventHandler(writer\_WriteNotify);

            while (writer.Write()) ;

            writer.Close();

        }

        // Optional Notify handler

        void writer\_WriteNotify(object sender, TraceEventArgs args)

        {

            Console.WriteLine("Writing: {0}" , args.CurrentRecord[0]);

            args.SkipRecord = false;

        }

    }

}

 In order to start SQL Trace, I created TraceServer object and called InitializeAsReader() method with SQL connection info and profiler template file (in this case, I used stanard.tdf. If you want to have replayable trace file, you have to choose tsql\_replay.tdf template). Once InitializeAsReader is successful, it automatically starts SQL trace again the SQL Server. So now you can send some TSQL statements to the SQL server. When everything is done and you want to stop trace, you press S as source code tells you. Then, next it will write SQL trace data to the output .trc file. I on purpose added WriteNotify event handler just to show you that you can check each trace. If you want to omit unncessary trace data during trace write, this is the place you can control by setting SkipRecord = true.

You can also save the trace results to SQL Table by using TraceTable class and also can replay trace by using TraceReplay class.

# Reading a Trace File using C# in SQL Server 2005

By [**Bill Graziano**](http://www.sqlteam.com/author/bill-graziano) on **25 October 2004** | [0 Comments](http://www.sqlteam.com/forums/topic.asp?TOPIC_ID=41718) | Tags: [.NET](http://www.sqlteam.com/tag/dot-net), [SMO](http://www.sqlteam.com/tag/smo), [Profiler & Trace](http://www.sqlteam.com/tag/profiler-trace)

SQL Server 2005 includes Server Managed Objects (SMO) which is a managed API to SQL Server. These are primarily used to administrative tasks and replaces DMO. One of the features of this is the ability to programmatically read trace files. This article walks through the code needed to read a trace file.

This article uses SQL Server 2005 Beta 2 and Visual Studio 2005 Beta 1. The first step is to create a new C# console project. I called mine TraceFileReader. Next I added reference to oneof the SMO DLLs. Right click the project and choose Add Reference. The file is located in D:\Program Files\Microsoft SQL Server\90\SDK\Assemblies and it's called Microsoft.SqlServer.ConnectionInfo.dll. This DLL contains the code we'll need to read the trace file. It should now be listed in the References section of the project.

Now we need a using reference to the namespace we're going to need. The code for reading trace files is in Microsoft.SqlServer.Management.Trace. So add the following line with the other using statements:

using Microsoft.SqlServer.Management.Trace;

This tells the application we plan to use the objects defined within this area of the DLL.

Before we can open the trace file we'll need two variables to hold the values we're going to read. The following code declares two string variables for this.

string textData;

string eventClass;

Now we need to add code to open the trace file. First we'll declare a variable to hold the trace file. That code looks like this:

TraceFile myTraceFile = new TraceFile();

TraceFile is the type of object we're declaring and I named the variable myTraceFile. Next we'll open the actual trace file.

myTraceFile.InitializeAsReader(@"D:\TraceFile.trc");

We're calling the InitializeAsReader method and passing it the name of our trace file. The "@" means to treat the string as a literal so the backslash isn't an escape character. After calling this method ourTraceFile object (myTraceFile) is going to act like a TraceReader object. The TraceReader object has it's own methods and properties that I'll discuss in a second. At this point the code should run and open the trace file.

The TraceFile object also supports two other methods. These are InitializeAsReplayOutputWriter and InitializeAsWriter. The second method make it possible to write a trace file yourself and the first I haven't explored yet.

The TraceReader object implements a couple of interfaces that make our life much easier. The two we care about are IDataReader and IDataRecord. These make the TraceReader act just like a record set from the database. We can use the Read method to read each trace event just like we would read records from a database. We'll put this code inside a while loop to read each record. That code looks like this:

while (myTraceFile.Read())

{

}

myTraceFile.Close();

The Read method will return true as long as there are records remaining to read. I also added a line after the while loop to close the trace file. Now add the following code inside the while loop:

eventClass = myTraceFile.GetString(myTraceFile.GetOrdinal("EventClass"));

textData = myTraceFile.GetString(myTraceFile.GetOrdinal("TextData"));

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

Console.WriteLine("EventClass: " + eventClass);

Console.WriteLine();

Console.WriteLine(textData);

Console.WriteLine();

We call the GetString method on myTraceFile to get the contents of a particular column in this record. The GetString method accepts a single integer which determines which column is returned. In my case I'm passing it the result of the GetOrdinal method of myTraceFile. This method accepts a column name as a parameter and returns the number of the column. Next I write out the results and are shown below. The complete program is shown following the output. Please be aware that this runs very slowly when run inside Visual Studio so experiment with a small trace file.

Starting...

===========================================

EventClass: Trace Start

===========================================

EventClass: ExistingConnection

-- network protocol: LPC

set quoted\_identifier on

set implicit\_transactions off

set cursor\_close\_on\_commit off

set ansi\_warnings on

set ansi\_padding on

set ansi\_nulls on

set concat\_null\_yields\_null on

set language us\_english

set dateformat mdy

set datefirst 7

set transaction isolation level read committed

===========================================

EventClass: SQL:BatchCompleted

SELECT \*

FROM dbo.titles

===========================================

EventClass: SQL:BatchCompleted

SELECT \*

FROM dbo.authors

===========================================

EventClass: Trace Stop

===========================================

Done.

#### Program Source

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.SqlServer.Management.Trace;

namespace TraceFileReader

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Starting...");

string textData;

string eventClass;

// Create a TraceFile object

TraceFile myTraceFile = new TraceFile();

// Open the trace file as a reader

myTraceFile.InitializeAsReader(@"D:\TraceFile.trc");

while (myTraceFile.Read())

{

// textData = String.Empty;

eventClass = myTraceFile.GetString(myTraceFile.GetOrdinal("EventClass"));

textData = myTraceFile.GetString(myTraceFile.GetOrdinal("TextData"));

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

Console.WriteLine("EventClass: " + eventClass);

Console.WriteLine();

Console.WriteLine(textData);

Console.WriteLine();

}

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

myTraceFile.Close();

Console.WriteLine("Done.");

}

}

}