DBMaster

OLEDB 用户手册

SYSCOM

SYSCOM Computer Engineering Co./Corporate Headquarters

B1, 2-7F No. 115 Emei Street, Wanhua District,
Taipei City 108, Taiwan (R.O.C.)

www.dbmaker.com

www.dbmaker.com.tw/service

Document No.645049-237491/DBM54CN-M03312017-OLED

发行日期: 2017-03-31

版权所有

对于本手册中没有体现的关于产品最新功能的描述，请在安装完成SYSCOM DBMaster 软件后阅读 README.TXT 文件。

注册商标

SYSCOM，SYSCOM 图标和 DBMaster 是SYSCOM 公司的注册商标。
Microsoft, MS-DOS, Windows 和 Windows NT 是 Microsoft 公司的注册商标。
UNIX 是 The Open Group 的注册商标。
ANSI 是美国国家标准化组织的注册商标。

手册中提到的其他产品名称或许是它们各自持有者的注册商标，仅仅是为提供此信息。
SQL 是行业语言，并不为任何公司或任何组织所有。

注意事项

本手册中有关软件的描述，均以该软件所提供的使用许可为基础。

对于授权许可的详细信息，请与您的经销商联系。关于计算机产品的特殊用途的市场性与适用性，经销商不会给予任何说明和保证。因外界因素如地震、过热、过冷和潮湿而引起产品的任何损坏以及由于使用不正确的电压和不兼容的软硬件而引起的损失和损坏，经销商概不负责。

虽然该手册的内容已经过仔细核对，但错误再所难免。若手册再有改动，不另行通知。还请见谅。

目录

1 简介

OLE DB是一系列组件对象模型（COM）接口。COM接口为应用程序在不同的DBMS以及非DBMS条件下访问存储数据提供了统一的访问路径。另外，为了支持多种信息源，OLE DB也支持数据库服务的执行。应用这些接口，数据用户可以通过相同的方法轻松访问数据。有了OLE DB，用户便无需考虑数据存储的位置、数据格式或数据类型。

DBMaster OLE DB Provider是为了访问DBMaster数据库系统而设计的，它允许DBMaster的OLE DB程序通过多个绑定接口，轻松开发高性能的用户应用程序。DBMaster OLE DB Provider专为使用DBMaster以及访问其它信息源不相容时而设计。

1.1 其它相关文件

除了本书之外，我们还为您提供了其它用户手册和参考文献，帮助您更深地了解DBMaster数据库管理系统。

您可以根据自己的需要，参考相关手册：

- 有关DBMaster性能和功能的介绍，请参考*DBMaster指南*。
- 有关设计、管理和维护DBMaster数据库的信息，请参考*数据库管理员手册*。
- 有关DBMaster SQL语-言的语法和使用的相关信息，请参考*SQL命令与函数参考手册*。
- 有关嵌入式的ESQL/C语言的语法和使用，请参考*ESQL/C程序员参考手册*。
- 有关ODBC API和JDBC API的信息，请参考*ODBC程序员参考手册*和*JDBC程序员参考手册*。
- 有关DCI COBOL接口的详细信息，请参考*DBMasterDCI用户手册*。
- 有关dmSQL命令行工具的使用方法，请参考*dmSQL使用手册*。
- 有关DBMaster的错误信息和警告信息，请参考*错误信息参考手册*。
- 有关使用DBMaster工具来配置和管理数据库的信息，请参考*数据库管理工具用户手册*、*服务器管理工具用户手册*和*配置管理工具用户手册*。
- 有关SQL存储过程的使用方法，请参考*SQL存储过程用户手册*。

1.2 技术支持

在软件试用期间，Syscom Computer Engineering Co.(“Syscom”)将为您提供30天的免费email支持和电话支持。当软件注册后，我们还会再为您提供30天的免费技术支持。如此一来，您就可以获得60天的免费支持。不仅如此，在您购买软件后，Syscom对任何问题都会以email的方式为您提供技术支持。

您除了可以获得免费的技术支持外，还可以以20%的零售价购买其它产品。要想获得更多的详细资料和价格信息，请与sales@dbmaker.com.tw保持联系。

您可以通过任何一种方式(普通信件、电话或email)与Syscom技术支持保持联系，请登录至：http://www.dbmaker.com.tw/service 以获取详细信息。在与Syscom技术支持联系之前，请先查询当前数据库的常见问题解答。

无论您以何种方式与Syscom的技术支持联系时，请务必写上以下有效信息：

- 产品名和版本号
- 注册号
- 注册的用户名和地址
- 供应商/发行商地址
- 操作平台和计算机系统配置
- 错误发生前执行的动作
- 如果可以，请提供错误信息和编号
- 其它相关信息

1.3 文档协定

为方便用户的阅读和使用，本手册使用了一种标准的排版约定，注释、程序、示例和命令行都用缩进排版的方式进行了特别的设置。

协定	说明
斜体字	斜体字表示必须输入的信息占位符，例如用户名和表名。此字符可用实际的名称来替换。有时，文档也会使用斜体字来介绍新的关键字，强调着重点。
黑体字	黑体字表示文件名、数据库名、表名称、字段名、用户名和其它数据库对象。它也用于强调程序执行步骤中的菜单命令。
关键字	文字段落中， SQL语言使用的关键字都是以大写字母出现的。
小符号	文档中出现的小写字符表示键盘上的按键，两个键名之间的加号（+）表示在按住第一个键不放的同时，再按第二个键。两个键名之间的逗号（，）表示释放第一个键以后，再按第二个键。
注意	包含一些重要的信息。
➲ **程序**	表示后面跟随的是程序的执行步骤或连续的项目。很多任务都是通过这种方式描述，给用户提供一个逻辑顺序步骤得以效仿。
➲ **示例**	例子用来阐明描述，通常包括屏幕上出现的文本，用户也可以将这些例子输入到计算机中，通过屏幕看到运行结果。当然，示例还包括一些原型和语法。
命令行	包括文本，这些命令都可以输入计算机中，显示在屏幕上。通常用于显示SQL命令的输入输出或dmconfig.ini中的内容。

表 1-1文档协定

2 支持的数据类型

下表展示了DBMaster Provider数据类型在OLE DB中相对应的数据类型：

DBMaster数据类型	OLE DB类型指示器	SQL类型
integer	DBTYPE_I4	SQL_INTEGER
smallint	DBTYPE_I2	SQL_SMALLINT
float	DBTYPE_R4	SQL_REAL
double	DBTYPE_R8	SQL_DOUBLE
decimal	DBTYPE_NUMERIC	SQL_DECIMAL
serial	DBTYPE_I4	SQL_INTEGER
char [(n)]	DBTYPE_BSTR, DBTYPE_WSTR	SQL_CHAR
varchar [(n)]	DBTYPE_BSTR, DBTYPE_WSTR	SQL_VARCHAR
binary	DBTYPE_BYTES	SQL_BINARY
varbinary	DBTYPE_BYTES	SQL_VARBINARY
Long varchar[(n)]	DBTYPE_WSTR	SQL_LONGVARCHAR
Long varbinary	DBTYPE_BYTES	SQL_LONGVARBINARY

DBMaster数据类型	OLE DB类型指示器	SQL类型
file	DBTYPE_BYTES	SQL_LONGVARBINARY SQL_FILE
date	DBTYPE_DATE , DBTYPE_DBDATE	SQL_TYPE_DATE
time	DBTYPE_DATE , DBTYPE_DBTIME	SQL_TYPE_TIME
timestamp	DBTYPE_DATE , DBTYPE_DBTIMESTAMP	SQL_TYPE_TIMESTAMP
nchar	DBTYPE_BSTR DBTYPE_WSTR	SQL_WCHAR
nvarchar	DBTYPE_BSTR DBTYPE_WSTR	SQL_WVARCHAR
blob	DBTYPE_BSTR DBTYPE_BYTES	SQL_LONGVARBINARY
clob	DBTYPE_BSTR, DBTYPE_WSTR	SQL_LONGVARCHAR
nclob	DBTYPE_BSTR DBTYPE_WSTR	SQL_WLONGVARCHAR

注意 *DBMaster OLE DB Provider支持十进制数据类型的精度为38。依据不同的模型，DBMaster OLE DB Provider对应的数据类型不同。例如：假设在ADO和ADO.NET模型下，char数据类型在使用ADO模型时与DBTYPE_BSTR相应，而使用ADO.NET模型时，与DBTYPE_WSTR相对应。*

3 COM对象定义

OLE DB使用微软标准通用接口，被称为COM构造。与ODBC系统类似，OLE DB提供了一组API，但是并不像ODBC那样，OLE DB API是完全基于COM组件的。也就是说，在抽象对象上的操作，例如数据源、会话、命令以及Rowset，都可以通过COM来访问。DBMaster OLE DB Provider支持四个对象：数据源对象、会话对象、命令对象以及Rowset对象。这些对象将在以下章节详细描述。

3.1 数据源

数据源对象是一个COM对象，它通过用户连接到驱动程序的基础数据存储区。DBMaster OLE DB Provider 定义了自己的数据源对象类。用户必须创建和初始化这个类的实例才能连接到OLE DB Provider。数据源对象就像会话对象制造厂，通过数据源对象产生一个新的会话对象。

数据源对象标准定义如下：

```
CoType TDataSource {
      [mandatory]   interface IDBCreateSession;
      [mandatory]   interface IDBInitialize;
      [mandatory]   interface IDBProperties;
      [mandatory]   interface IPersist;
      [optional]    interface IConnectionPointContainer;
      [optional]    interface IDBInfo;
      [optional]    interface IPersistFile;
   }
```

3.2 会话

会话对象代表对DBMaster数据库的一个单独连接，允许数据访问和操作。一个单独的数据源对象可以创建多个会话，会话对象是命令与Rowset对象的加工厂，它提供了创建命令对象、Rowset以及修改表和索引的模式。会话对象能够作为其他交易对象的加工厂，交易对象用于控制嵌套的交易。

当所有关系到会话对象的参照被释放后，会话对象会从内存中移除，并且连接都被删除。会话对象模板如下：

```
CoType TSession {
     [mandatory]   interface IGetDataSource;
     [mandatory]   interface IOpenRowset;
     [mandatory]   interface ISessionProperties;
     [optional]    interface IAlterIndex;
     [optional]    interface IAlterTable;
     [optional]    interface IBindResource;
     [optional]    interface ICreateRow;
     [optional]    interface IDBCreateCommand;
     [optional]    interface IDBSchemaRowset;
     [optional]    interface IIndexDefinition;
     [optional]    interface ISupportErrorInfo;
     [optional]    interface ITableCreation;
     [optional]    interface ITableDefinition;
     [optional]    interface ITableDefinitionWithConstraints;
     [optional]    interface ITransaction;
     [optional]    interface ITransactionJoin;
     [optional]    interface ITransactionLocal;
     [optional]    interface ITransactionObject;
  }
```

3.3 命令

命令有四种状态：初始化、无准备的、已经准备的或已经执行的。命令中的参数用于在执行时绑定用户变量，在执行时一条命令会返回一条结果或多条结果。单条结果可能是一个行集对象或者为行数（例如，行数受到命令update、delete或insert的影响）。命令同样可能返回多条结果，如果命令由多条或独立的文本命令组成，例如一批SQL语句或者当多个参数集传给命令时，结果一定会返回到一个多结果对象中。

命令对象被用来执行DBMaster OLE DB Provider文本命令。文本命令通过DBMaster OLE DB Provider语言来传递，通常被用于创建一个行集，例如执行一条SQL SELECT命令。

命令对象标准如下：

```
CoType TCommand {
      [mandatory]   interface IAccessor;
      [mandatory]   interface IColumnsInfo;
      [mandatory]   interface ICommand;
      [mandatory]   interface ICommandProperties;
      [mandatory]   interface ICommandText;
      [mandatory]   interface IConvertType;
      [optional]    interface IColumnsRowset;
      [optional]    interface ICommandPersist;
      [optional]    interface ICommandPrepare;
      [optional]    interface ICommandWithParameters;
      [optional]    interface ISupportErrorInfo;
   }
```

3.4 行集

行集是一个重要的对象，能够使OLE DB组成以表形式传递并且操作数据。一个行集对象是一组行的集合，每一行都拥有字段数据。例如，DBMaster OLE DB Provider以行集的形式传递数据和元数据。通过OLE DB提供的行集使得通过相同的对象聚集读取或输入的数据成为可能。

行集对象的标准定义如下：

```
CoType TRowset {
      [mandatory]   interface IAccessor;
      [mandatory]   interface IColumnsInfo;
      [mandatory]   interface IConvertType;
      [mandatory]   interface IRowset;
      [mandatory]   interface IRowsetInfo;
      [optional]    interface IConnectionPointContainer;
      [optional]    interface IDBAsynchStatus;
      [optional]    interface IRowsetChange;
      [optional]    interface IRowsetFind;
      [optional]    interface IRowsetIndex;
      [optional]    interface IRowsetLocate;
      [optional]    interface IRowsetRefresh;
      [optional]    interface IRowsetScroll;
      [optional]    interface IRowsetUpdate;
      [optional]    interface IRowsetView;
   }
```

4 接口 (OLE DB)

接口是一组语义相关的函数，用于访问COM对象。每一个OLE DB接口定义了一个约束，允许对象根据组件对象模型（COM）互相作用。OLE DB提供了许多接口的实现，大多数接口可以通过开发者设计的OLE DB应用来执行相关操作。本章对当前DBMaster OLE DB Provider支持的接口进行了总结。

4.1 DBMaster OLE DB Provider接口

下表总结了DBMaster OLE DB Provider当前版本支持的接口，更多信息请参考MSDN。

对象	接口	是否支持
Command	**IAccessor**	Yes
	IColumnsInfo	Yes
	IColumnsRowset	Yes
	ICommand	Yes
	ICommandPersist	No
	ICommandPrepare	Yes
	ICommandProperties	Yes
	ICommandText	Yes
	ICommandWithParameters	Yes
	IConvertType	Yes
	IDBInitialize	No
	ISupportErrorInfo	No
Data Source	**IConnectionPointContainer**	No
	IDBAsynchStatus	No
	IDBAsynchNotify	No
	IDBCreateSession	Yes
	IDBInfo	Yes
	IDBInitialize	Yes
	IDBProperties	Yes
	IPersist	Yes
	IPersistFile	No
	ISupportErrorInfo	No
Error	**IErrorInfo**	No

Rowset	**IAccessor**	Yes
	IColumnsInfo	Yes
	IColumnsRowset	Yes
	IConnectionPointContainer	No
	IConvertType	Yes
	IDBAsynchStatus	No
	IDBAsynchNotify	No
	IDBInitialize	No
	IRowset	Yes
	IRowsetChange	Yes
	IRowsetFind	No
	IRowsetIdentity	No
	IRowsetIndex	No
	IRowsetInfo	Yes
	IRowsetLocate	No
	IRowsetRefresh	No
	IRowsetScroll	No
	IRowsetUpdate	No
	IRowsetView	No
	ISupportErrorInfo	No
Session	**IAlterIndex**	No
	IAlterTable	No
	IBindResource	No
	IConnectionPointContainer	No
	ICreateRow	No
	IDBASynchStatus	No
	IDBCreateCommand	Yes
	IDBInitialize	No

	IDBSchemaRowset	Yes
	IGetDataSource	Yes
	IIndexDefinition	No
	IOpenRowset	Yes
	ISessionProperties	Yes
	ISupportErrorInfo	No
	ITableDefinition	No
	ITransaction	Yes
	ITransactionJoin	Yes
	ITransactionLocal	Yes
	ITransactionObject	No

表4-1 DBMaster OLE DB Provider支持的接口

5 使用OLE DB服务程序

本章将按照以下三部分对使用DBMaster OLE DB服务程序进行详细说明：

1. 搭建环境。
2. 调用OLE DB服务程序。
3. 编写OLE DB应用程序。

5.1 搭建环境

注册表项：使用用户应用程序，启动 DBMaster OLE DB 服务程序。

您需要注册 DBMaster OLE DB 服务程序的 GUID。

对于 DBMaster 和 DBMaker 普通版本，在安装 DBMaster 时，会自动注册 OLE DB 的 GUID。

对于 DBMaster 和 DBMaker 的 BUNDLE 版本，则需要使用如下命令来注册 OLE DB 的 GUID。

```
regsvr32  bundle_path/bin/dmole54.dll
```

成功注册了 OLE DB 的 GUID 之后，您就可以在 DBMaster 的 BUNDLE 版本中使用 OLE DB。

5.2 调用OLE DB服务程序

根据用户的编程需求，可以使用多种方法来调用DBMaster(dmole54.dll) OLE DB。在IDBInitialize上调用CoCreateInstance，是传统上用来在OLE DB中打开数据源对象的一种方法。

添加主键

用户可以使用如下典型的连接字符串，从ADO或ADO.net来调用DBMaster OLE DB服务程序：

```
"Provider=dmole54;Data Source=dbName;User ID=userName; Password=userPassword;"
```

在上述的连接字符串中，dmole54是服务程序的名称。OLE DB服务程序的名称因DBMaster和DBMaker版本不同而各异，具体如下所示：

对于DBMaker普通版本，OLE DB服务程序的名称是dmole54。

对于DBMaster普通版本，OLE DB服务程序的名称是dmole54J。

对于DBMakerBUNDLE版本，OLE DB服务程序的名称是dmole54B。

对于DBMasterBUNDLE版本，OLE DB服务程序的名称是dmole54JB。

从ADO或ADO.NET调用时，OLE DB服务已自动开启。

5.3 编写OLE DB应用程序

编写OLE DB应用程序包括三步：

1. 建立新的数据源连接。
2. 通过OLE DB驱动执行命令。
3. 处理返回的结果。

建立新的数据源连接

对OLE DB用户来说，首要任务是创建一个数据源对象的实例。创建数据源的基本步骤如下：

1. 通过调用**CoInitialize(NULL)**初始化COM库。
2. 通过调用**CoCreateInstance**方法来创建一个数据源对象的实例，语法如下：

```
TDAPI CoCreateInstance(  REFCLSID rclsid,
                          LPUNKNOWN pUnkOuter,
                          DWORD dwClsContext,
                          REFIID riid,
                          LPVOID * ppv);
```

一个唯一的类别标识符(CLSID)识别每一个OLE DB Provider，对于DMOLE54而言类别标识符为CLSID_DMOLE54。

3. 数据源对象揭示了**IDBProperties**接口。用户使用**IDBProperties**来提供基本鉴定信息，如服务器名称、数据库名称、用户ID以及密码。这些办法是通过调用**IDBProperties::SetProperties**来实现的。
4. 数据源对象同样揭示了**IDBInitialize**接口。通过调用**IDBInitialize::Initialize**方法来确立与数据源的连接。

通过OLE DB驱动来执行命令

如果在数据源连接已经确认后，用户通过调用**IDBCreateSession::CreateSession**来创建会话。会话的功能是作为命令、行集或者交易加工厂。

会话对象能够创建命令对象。DBMaster OLE DB Provider中的命令对象支持SQL命令的执行，同时也支持多个参数。

可参考下面执行命令的示例。如果用户想执行SELECT * FROM Authors命令，一开始需要**IDBCreateCommand**接口，用户可以通过执行**IDBCreateCommand::CreateCommand**方法来创建一个命令对象，然后要求**ICommandText**接口。**ICommandText::SetCommandText**方法用于指明执行的命令。最后，使用**Execute**来执行命令，像SELECT * FROM **Authors**就产生了一个结果集对象。

用户可以要求**IOpenRowset**接口通过单独的表和索引来直接工作。通过**IOpenRowset::OpenRowset**方法开启并且返回一个行集，包括从一个单独的基础表或索引返回的所有行。

处理返回的结果

当行集对象通过任意命令执行或者由提供者直接生成一个行集对象时，用户必须在行集中找回并且访问数据。

行集是重要对象，它使得所有OLE DB数据驱动以表的形式展示数据。行集由一系列行组成，每一行都包括字段数据。一个行集对象通过展示不同的接口使访问更加轻松。例如：**IRowset**是包括持续从行集中获取行的方法的接口；**IAccessor**是定义一组字段绑定描述表数据与用户程序是如何绑定的接口；**IColumnInfo**接口提供行集的字段信息；**IRowsetInfo**接口提供行集的信息。

用户可以调用**IRowset**方法从行集中取回一行数据到缓冲区中。用户必须在调用**GetData**之前描述使用一系列DBBINDING 结构的缓冲。在获取数据时，提供者要使用每一个与决定用户缓冲区中在哪里和如何获取数据相绑定的信息。在用户缓冲区设置数据时，提供者使用每一个与决定用户缓冲区中在哪里和如何获取数据相绑定的信息。

DBBINDING结构被指定后，通过调用**IAccessor::CreateAccessor**方法创建了一个访问，该访问是许多绑定的集合并且被用于获得或者设置用户缓冲区的数据。

6 示例

本章提供的示例演示了行集程序和一个OLE DB用户的对象模式。示例中创建了一个数据源、一个会话以及行集对象，允许用户演示并且在行集中操作行以及处理错误。命令行转换用于指明当一个人口调查统计员、级别号码、用户提示或者连接串被用于创建数据源对象，命令被用于创建行集等。

注意 *本章有三段代码的示例。C++示例程序展示的是DBMaster支持的OLE DB基本的执行，Visual Basic示例程序是通过ADO方法来访问DBMaster OLE DB Provider，而C#示例程序是通过ADO.NET方法来访问DBMaster OLE DB Provider。*

6.1 OLE DB用户应用程序微软Visual C++示例

下例演示了如何初始化数据源并且用C++语言编写如何通过OLE DB Provider来访问DBMaster数据库。

```
#include "stdafx.h"
#define UNICODE
#define _UNICODE
#define DBINITCONSTANTS
#define INITGUID
#define BLOCK_SIZE 512

#define DMOLE54

#include <windows.h>
#include <stdio.h>     // Input and output functions
#include <stddef.h>   // for macro offset
#include <oledb.h>   // OLE DB include files
#include <oledberr.h>  // OLE DB Errors
#include <Ks.h>
#include <Guiddef.h>
#include <comsvcs.h>
#include <atlbase.h>
#include "dmdasql.h"

static IMalloc* g_pIMalloc = NULL;

typedef struct {
                   LONG    bookmark;
                   char    id[9];
                   char    fname[20];
                   DBDATE   hire_date;
} Employee;
typedef struct {
          char    id[10];
          char    fname[20];
          char    lname[20];
} EEmployee;

typedef struct tagemployee1{
                                                  short  szjob_id;
}employee1;

HRESULT SetInitProps(IDBInitialize *pIDBInitialize)
{
         const ULONG   nProps = 4;
         IDBProperties* pIDBProperties = NULL;
         DBPROP InitProperties[nProps] = {0};
```

```
        DBPROPSET rgInitPropSet = {0};
        HRESULT hr = S_OK;

        // Initialize common property options
        for (ULONG i = 0; i < nProps; i++ )
        {
                VariantInit(&InitProperties[i].vValue);
                InitProperties[i].dwOptions = DBPROPOPTIONS_REQUIRED;
                InitProperties[i].colid = DB_NULLID;
        }

        // Level of prompting that will accompany the
        // connection process
        InitProperties[0].dwPropertyID = DBPROP_INIT_PROMPT;
        InitProperties[0].vValue.vt = VT_I2;
        InitProperties[0].vValue.iVal = DBPROMPT_NOPROMPT;

        // Data source name (please refer to the sample source included with the
OLE
        // DB SDK)
        InitProperties[1].dwPropertyID = DBPROP_INIT_DATASOURCE;
        InitProperties[1].vValue.vt = VT_BSTR;
        InitProperties[1].vValue.bstrVal = SysAllocString(OLESTR("oledbtest"));

        // User ID
        InitProperties[2].dwPropertyID = DBPROP_AUTH_USERID;
        InitProperties[2].vValue.vt = VT_BSTR;
        InitProperties[2].vValue.bstrVal = SysAllocString(OLESTR("sysadm"));

        // Password
        InitProperties[3].dwPropertyID = DBPROP_AUTH_PASSWORD;
        InitProperties[3].vValue.vt = VT_BSTR;
        InitProperties[3].vValue.bstrVal = SysAllocString(OLESTR(""));

        rgInitPropSet.guidPropertySet = DBPROPSET_DBINIT;
        rgInitPropSet.cProperties = nProps;
        rgInitPropSet.rgProperties = InitProperties;

// Set initialization properties
        hr = pIDBInitialize->QueryInterface(IID_IDBProperties, (void**)
                &pIDBProperties);
        hr = pIDBProperties->SetProperties(1, &rgInitPropSet);

        SysFreeString(InitProperties[1].vValue.bstrVal);
        SysFreeString(InitProperties[2].vValue.bstrVal);
        SysFreeString(InitProperties[3].vValue.bstrVal);

        pIDBProperties->Release();
        return (hr);
}

             // Initialize a data source
```

```
HRESULT InitDSO(IDBInitialize **ppIDBInitialize)
{
CoCreateInstance(CLSID_DMOLE54, NULL,
CLSCTX_INPROC_SERVER, IID_IDBInitialize, (void**)ppIDBInitialize);

        if (ppIDBInitialize == NULL)
                return E_FAIL;

        if (FAILED(SetInitProps(*ppIDBInitialize)))
                return (E_FAIL);

        if (FAILED((*ppIDBInitialize)->Initialize()))
                return (E_FAIL);

        return S_OK;
}

             // Test property and return its property values in the Data Source
HRESULT TestProperty(IDBInitialize *pIDBInitialize)
{
        IDBProperties *pIDBProperties = NULL;
        IRowset *pIRowset = NULL;

        DBPROPSET *rgPropSet = NULL;
        DBPROPIDSET rgPropIDSet[1] = {0};
        DBPROPID rgPropID = {0};
        HRESULT hr        = S_OK;
        ULONG cPropSets = 0;

        pIDBInitialize->QueryInterface(IID_IDBProperties,
                (void**)&pIDBProperties);

        rgPropID = DBPROP_CANSCROLLBACKWARDS;
        rgPropIDSet->cPropertyIDs = 1;
        rgPropIDSet->rgPropertyIDs = &rgPropID;
        rgPropIDSet->guidPropertySet = DBPROPSET_ROWSET;

        if((hr = pIDBProperties->GetProperties(1, rgPropIDSet,
                &cPropSets, &rgPropSet)) != S_OK)
        {
                printf("DBPROP_CANSCROLLBACKWARDS -- failed\n");
return hr;
        }
        printf("DBPROP_CANSCROLLBACKWARDS -- OK\n");
        return hr;
}

// Test rowset and open and return a rowset that includes all rows from a single
base table
HRESULT DisplayRowset(IDBInitialize *pIDBInitialize)
{
        IDBCreateSession *pIDBCreateSession = NULL;
```

```
        IOpenRowset *pIOpenRowset = NULL;
        HRESULT hr = S_OK;
        DBID  TableID = {0};
        WCHAR wszTableName[] = L"employee";
        DBPROPSET rgPropSets[1] = {0};
        const ULONG cProperties = 7;
        DBPROP rgProp[cProperties] = {0};
        IRowset *pIRowset = NULL;

// Create the TableID
    TableID.eKind            = DBKIND_NAME;
    TableID.uName.pwszName   = wszTableName;

        rgProp[0].colid = DB_NULLID;
        rgProp[0].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[0].dwStatus = 0;
        rgProp[0].dwPropertyID = DBPROP_CANHOLDROWS;
        rgProp[0].vValue.vt       = VT_BOOL;
        rgProp[0].vValue.boolVal = VARIANT_TRUE;

        rgProp[1].colid = DB_NULLID;
        rgProp[1].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[1].dwStatus = 0;
        rgProp[1].dwPropertyID = DBPROP_CANSCROLLBACKWARDS;
        rgProp[1].vValue.vt       = VT_BOOL;
        rgProp[1].vValue.boolVal = VARIANT_TRUE;

        rgProp[2].colid = DB_NULLID;
        rgProp[2].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[2].dwStatus = 0;
        rgProp[2].dwPropertyID = DBPROP_CANFETCHBACKWARDS;
        rgProp[2].vValue.vt       = VT_BOOL;
        rgProp[2].vValue.boolVal = VARIANT_TRUE;

    rgProp[3].colid = DB_NULLID;
        rgProp[3].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[3].dwStatus = 0;
        rgProp[3].dwPropertyID = DBPROP_IRowsetChange;
        rgProp[3].vValue.vt       = VT_BOOL;
        rgProp[3].vValue.boolVal = VARIANT_TRUE;

        rgProp[4].colid = DB_NULLID;
        rgProp[4].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[4].dwStatus = 0;
        rgProp[4].dwPropertyID = DBPROP_UPDATABILITY;
        rgProp[4].vValue.vt       = VT_I4;
        rgProp[4].vValue.lVal = DBPROPVAL_UP_CHANGE | DBPROPVAL_UP_INSERT |
DBPROPVAL_UP_DELETE;

        rgProp[5].colid = DB_NULLID;
        rgProp[5].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[5].dwStatus = 0;
```

```
        rgProp[5].dwPropertyID = DBPROP_ACCESSORDER;
        rgProp[5].vValue.vt       = VT_I4;
        rgProp[5].vValue.lVal = DBPROPVAL_AO_RANDOM;

        rgProp[6].colid = DB_NULLID;
        rgProp[6].dwOptions = DBPROPOPTIONS_REQUIRED;
        rgProp[6].dwStatus = 0;
        rgProp[6].dwPropertyID = DBPROP_IConnectionPointContainer;
        rgProp[6].vValue.vt       = VT_BOOL;
        rgProp[6].vValue.boolVal = VARIANT_TRUE;

        hr = pIDBInitialize->QueryInterface(IID_IDBCreateSession,
                (void**)&pIDBCreateSession);

        hr = pIDBCreateSession->CreateSession(NULL, IID_IOpenRowset,
(IUnknown**)&pIOpenRowset);
        pIDBCreateSession->Release();

        rgPropSets->rgProperties = rgProp;
        rgPropSets->cProperties   = cProperties;
        rgPropSets->guidPropertySet = DBPROPSET_ROWSET;

        hr = pIOpenRowset->OpenRowset(
                NULL,
                &TableID,
                NULL,
                IID_IRowset,
                1,
                rgPropSets,
                (IUnknown**)&pIRowset);
        pIOpenRowset->Release();

        if(!pIRowset)
        {
                return hr;
        }

        IColumnsInfo     *pIColumnsInfo = NULL;
        DBORDINAL                cColumns = 0;
        DBCOLUMNINFO     *prgInfo = NULL;
        OLECHAR                  *pstrBuf = NULL;
        ULONG i = 0;

        pIRowset->QueryInterface(IID_IColumnsInfo, (void **)&pIColumnsInfo);
        if(pIColumnsInfo)
        {
                hr = pIColumnsInfo->GetColumnInfo(&cColumns, &prgInfo,
&pstrBuf);
                if(SUCCEEDED(hr))
                {
                   printf("GetColumnInfo -- OK\n");
                }
```

```
                pIColumnsInfo->Release();
        }
        IAccessor         *pIAccessor = NULL;
        HACCESSOR hAccessor = 0;
        DBBINDSTATUS rgStatus[3] = {0};
        DBBINDING Bindings[3] = {0};
        ULONG acbLengths[] = {9, 20, 6};

        for (i=0; i<3; i++)
        {
                Bindings[i].iOrdinal = i + 1;
                Bindings[i].obLength = 0;
                Bindings[i].obStatus = 0;
                Bindings[i].pTypeInfo = NULL;
                Bindings[i].pObject = NULL;
                Bindings[i].pBindExt = NULL;
                Bindings[i].dwPart = DBPART_VALUE;
                Bindings[i].dwMemOwner = DBMEMOWNER_CLIENTOWNED;
                Bindings[i].eParamIO = DBPARAMIO_OUTPUT;
                Bindings[i].cbMaxLen = acbLengths[i];
                Bindings[i].dwFlags = 0;
                Bindings[i].wType = DBTYPE_STR;
                if(i==2){Bindings[i].wType = DBTYPE_DBDATE;}
                Bindings[i].bPrecision = 0;
                Bindings[i].bScale = 0;
        }
        Bindings[0].obValue = offsetof(Employee, id);
        Bindings[1].obValue = offsetof(Employee, fname);
        Bindings[2].obValue = offsetof(Employee, hire_date);

        pIRowset->QueryInterface(IID_IAccessor, (void**)&pIAccessor);
        hr = pIAccessor->CreateAccessor(
                DBACCESSOR_ROWDATA,
                3,
                Bindings,
                0,
                &hAccessor,
                rgStatus);

        pIAccessor->Release();

        Employee emp = {0};
        ULONG    cRowsObtained = 0;
        HROW     rghRows[100] = {0};
        HROW*    phRows = rghRows;

        hr = pIRowset->GetNextRows(NULL, 0, 21, &cRowsObtained, &phRows);
        for(i=0; i<cRowsObtained; i++)
        {
                hr = pIRowset->GetData(rghRows[i], hAccessor, &emp);
                if(hr != S_OK)
                        break;
```

```
                printf("%s\t %s\n", emp.id, emp.fname);
        }

        pIAccessor->ReleaseAccessor(hAccessor, NULL);
pIRowset->Release();
return S_OK;
}

              // Manipulate a command object and execute the select command
HRESULT My_Sel_Command(IDBInitialize *pIDBInitialize)
{
        IDBCreateSession* pIDBCreateSession = NULL;
        IDBCreateCommand* pIDBCreateCommand = NULL;
        ICommandText* pICommandText = NULL;
        WCHAR wSQLSelect[] = L"select * from employee";
        long cRowsAffected = 0;
        IAccessor* pIAccessor = NULL;
        IRowset *pIRowset = NULL;
        HACCESSOR hAccessor = {0};
        ULONG I = 0;
        HRESULT hr = S_OK;
        DBBINDSTATUS rgStatus[3] = {0};
        DBBINDING Bindings[3] = {0};
        ULONG acbLengths[] = {9, 20, 6};

        // Get the session
        pIDBInitialize->QueryInterface(IID_IDBCreateSession,
(void**)&pIDBCreateSession);
        pIDBCreateSession->CreateSession(NULL, IID_IDBCreateCommand,
(IUnknown**)&pIDBCreateCommand);
        pIDBCreateSession->Release();

        // Create the command
        pIDBCreateCommand->CreateCommand(NULL, IID_ICommandText, (IUnknown**)
&pICommandText);
        pIDBCreateCommand->Release();

        // Set the command text for the first delete statement then execute the
command.

        pICommandText->SetCommandText(DBGUID_DBSQL, wSQLSelect);
        pICommandText->Execute(NULL, IID_IRowset, NULL, &cRowsAffected, (IUnknown
**)&pIRowset);

        for (i=0; i<3; i++)
        {
                Bindings[i].iOrdinal = i + 1;
                Bindings[i].obLength = 0;
                Bindings[i].obStatus = 0;
                Bindings[i].pTypeInfo = NULL;
                Bindings[i].pObject = NULL;
                Bindings[i].pBindExt = NULL;
```

```
                Bindings[i].dwPart = DBPART_VALUE;
                Bindings[i].dwMemOwner = DBMEMOWNER_CLIENTOWNED;
                Bindings[i].eParamIO = DBPARAMIO_OUTPUT;
                Bindings[i].cbMaxLen = acbLengths[i];
                Bindings[i].dwFlags = 0;
                Bindings[i].wType = DBTYPE_STR;
                if(i==2){Bindings[i].wType = DBTYPE_DBDATE;}
                Bindings[i].bPrecision = 0;
                Bindings[i].bScale = 0;
        }
Bindings[0].obValue = offsetof(Employee, id);
        Bindings[1].obValue = offsetof(Employee, fname);
        Bindings[2].obValue = offsetof(Employee, hire_date);

        pIRowset->QueryInterface(IID_IAccessor, (void**)&pIAccessor);
        hr = pIAccessor->CreateAccessor(
                DBACCESSOR_ROWDATA,
                3,
                Bindings,
                0,
                &hAccessor,
                rgStatus);

        Employee emp = {0};
        ULONG    cRowsObtained = 0;
        HROW     rghRows[100] = {0};
        HROW*    phRows = rghRows;

pIRowset->GetNextRows(DB_NULL_HCHAPTER, 1, 1, &cRowsObtained, &phRows);

        for(i=0; i<cRowsObtained; i++)
        {
                pIRowset->GetNextRows(DB_NULL_HCHAPTER, 0, i+2, &cRowsObtained,
&phRows);
                hr = pIRowset->GetData(rghRows[i], hAccessor, &emp);
                if(hr != S_OK)
                        break;
                printf("%s\n", emp.id);
        }
        pIAccessor->ReleaseAccessor(hAccessor, NULL);
        pIAccessor->Release();
        pIRowset->Release();
        pICommandText->Release();

        return S_OK;
}

// Create accessor
HRESULT CreateParamAccessor(
        ICommand* pICmd,   // [in]
        HACCESSOR*  phAccessor,   // [out]
        IAccessor** ppIAccessor   // [out]
```

```
        )
{
        IAccessor* pIAccessor = NULL;
        HACCESSOR hAccessor = NULL;
        const ULONG nParams = 3;
        DBBINDING Bindings[nParams] = {0};
        DBBINDSTATUS rgStatus[nParams] = {0};
        HRESULT hr = S_OK;

        ULONG acbLengths[] = {10,20,20};

        for (ULONG i = 0; i < nParams; i++)
        {
                Bindings[i].iOrdinal = i + 1;
                Bindings[i].obLength = 0;
                Bindings[i].obStatus = 0;
                Bindings[i].pTypeInfo = NULL;
                Bindings[i].pObject = NULL;
                Bindings[i].pBindExt = NULL;
                Bindings[i].dwPart = DBPART_VALUE;
                Bindings[i].dwMemOwner = DBMEMOWNER_CLIENTOWNED;
                Bindings[i].eParamIO = DBPARAMIO_INPUT;
                Bindings[i].cbMaxLen = acbLengths[i];
                Bindings[i].dwFlags = 0;
                Bindings[i].wType = DBTYPE_STR;
                Bindings[i].bPrecision = 0;
                Bindings[i].bScale = 0;
        }

        Bindings[0].obValue = offsetof(EEmployee, id);
        Bindings[1].obValue = offsetof(EEmployee, fname);
        Bindings[2].obValue = offsetof(EEmployee, lname);

        pICmd->QueryInterface(IID_IAccessor, (void**)&pIAccessor);

        hr = pIAccessor->CreateAccessor(
DBACCESSOR_PARAMETERDATA,   // Accessor used to specify parameter data
                nParams,   // Number of parameters being bound
                Bindings,   // Structure containing bind information
                sizeof(EEmployee),   // Size of parameter structure
                &hAccessor,   // Returned accessor handle
                rgStatus   // Information about binding validity
                );

        *ppIAccessor = pIAccessor ;
        *phAccessor = hAccessor ;

        return (hr);
}

// Execute an insert command with parameter
HRESULT InsertWithParameters(IDBInitialize *pIDBInitialize)
```

```
{
        IDBCreateSession* pIDBCreateSession = NULL;
        IDBCreateCommand* pIDBCreateCommand = NULL;
        ICommandText* pICommandText = NULL;
        ICommandPrepare* pICommandPrepare = NULL;
        ICommandWithParameters* pICmdWithParams = NULL;
        IAccessor* pIAccessor = NULL;
        WCHAR wSQLString[] = TEXT("insert into eemployee values(?,?,?)");
        DBPARAMS Params = 0;
        HRESULT hr = S_OK;
        long cRowsAffected = 0;
        HACCESSOR hParamAccessor = {0};
        EEmployee aEmployee[] =
        {
                "1001", "Terrible", "Fang",
                "1002", "David", "Chen",
                "1003", "Alen", "Wu"
        };
        EEmployee Temp = {0};

        ULONG nParams = 3;

        pIDBInitialize->QueryInterface(IID_IDBCreateSession,
                (void**)&pIDBCreateSession);
        pIDBCreateSession->CreateSession(NULL, IID_IDBCreateCommand,
                (IUnknown**) &pIDBCreateCommand);
        pIDBCreateSession->Release();

        // Create the command
        pIDBCreateCommand->CreateCommand(NULL, IID_ICommandText,
                (IUnknown**) &pICommandText);
        pIDBCreateCommand->Release();

        // The command requires the actual text and a language indicator
        pICommandText->SetCommandText(DBGUID_DBSQL, wSQLString);

        // Prepare the command
        hr = pICommandText->QueryInterface(IID_ICommandPrepare,
(void**)&pICommandPrepare);
        if (FAILED(pICommandPrepare->Prepare(0)))
        {
                pICommandPrepare->Release();
                pICommandText->Release();
                return (E_FAIL);
        }
        pICommandPrepare->Release();

        // Create parameter accessors
        if (FAILED(CreateParamAccessor(pICommandText, &hParamAccessor,
&pIAccessor)))
        {
                pICommandText->Release();
```

```
                return (E_FAIL);
        }

        Params.pData = &Temp;   // pData is the buffer pointer
        Params.cParamSets = 1;   // Number of sets of parameters
        Params.hAccessor = hParamAccessor;   // Accessor to the parameters

        // Specify the parameter information
        for (UINT nCust = 0; nCust < 3; nCust++)
        {
                strcpy(Temp.id, aEmployee[nCust].id);
                strcpy(Temp.fname, aEmployee[nCust].fname);
                strcpy(Temp.lname, aEmployee[nCust].lname);
                // Execute the command
                hr = pICommandText->Execute(NULL, IID_NULL, &Params,
&cRowsAffected, NULL);
                printf("%ld rows inserted.\n", cRowsAffected);
        }

        pIAccessor->ReleaseAccessor(hParamAccessor, NULL);
        pIAccessor->Release();
        pICommandText->Release();

        return S_OK;
}

// Create accessor
HRESULT myCreateParamAccessor
   (
   ICommand* pICmd,   // [in]
   HACCESSOR*  phAccessor,    // [out]
   IAccessor** ppIAccessor   // [out]
   )
   {
   IAccessor* pIAccessor = NULL;
   HACCESSOR hAccessor = {0};
   const ULONG nParams = 1;
   DBBINDING Bindings[nParams] = {0};
   DBBINDSTATUS rgStatus[nParams] = {0};    // Return information for
                                       // individual binding validity
   HRESULT hr = S_OK;
   ULONG acbLengths[] = {2};

   for (ULONG i = 0; i < nParams; i++)
      {
      Bindings[i].iOrdinal = i + 1;
      Bindings[i].obLength = 0;
      Bindings[i].obStatus = 0;
      Bindings[i].pTypeInfo = NULL;
      Bindings[i].pObject = NULL;
      Bindings[i].pBindExt = NULL;
      Bindings[i].dwPart = DBPART_VALUE;
```

```
      Bindings[i].dwMemOwner = DBMEMOWNER_CLIENTOWNED;
      Bindings[i].eParamIO = DBPARAMIO_INPUT;
      Bindings[i].cbMaxLen = acbLengths[i];
      Bindings[i].dwFlags = 0;
      Bindings[i].wType = DBTYPE_I2;
      Bindings[i].bPrecision = 0;
      Bindings[i].bScale = 0;
      }

   Bindings[0].obValue = offsetof(employee, ajob_id);

   pICmd->QueryInterface(IID_IAccessor, (void**)&pIAccessor);

   hr = pIAccessor->CreateAccessor(
      DBACCESSOR_PARAMETERDATA, //Accessor for specifying parameter data
      nParams,   // Number of parameters being bound
      Bindings,   // Structure containing bind information
      sizeof(employee),    // Size of parameter structure
      &hAccessor,   // Returned accessor handle
      rgStatus   // Information about binding validity
      );

      *ppIAccessor = pIAccessor;
      *phAccessor = hAccessor;
       return (hr);
   }

// Execute a command with a parameter
HRESULT My_Command_Para(IDBInitialize *pIDBInitialize)
{
   IDBCreateSession* pIDBCreateSession = NULL;
   IDBCreateCommand* pIDBCreateCommand = NULL;
   ICommandText* pICommandText = NULL;
   ICommandPrepare* pICommandPrepare = NULL;
   ICommandWithParameters* pICmdWithParams = NULL;
   IAccessor* pIAccessor = NULL;
   // WCHAR wSQLString[] = L"delete from employee where job_id=?";
   // WCHAR wSQLString[] = L"select * from employee where job_id=?";
   WCHAR wSQLString[] = L"update employee set fname='LingAn' where job_id=?";
   DBPARAMS Params = 0;
   HRESULT hr = S_OK;
   long cRowsAffected = 0;
   HACCESSOR hParamAccessor = {0 };
   IRowset *pIRowset = NULL;
    DBORDINAL               rgParamOrdinals[1] = {0};
    DBPARAMBINDINFO      rgParamBindInfo[1] = {0};

    employee1 aEmployee[] =
        {
                   5,6,7
        };
    employee Temp = {0};
```

```
   ULONG nParams = 1;

   rgParamOrdinals[0]              = 1;
   rgParamBindInfo[0].bPrecision    = 0;
   rgParamBindInfo[0].bScale       = 0;
   rgParamBindInfo[0].dwFlags     = DBPARAMFLAGS_ISINPUT;
   rgParamBindInfo[0].pwszDataSourceType = (unsigned short *) L"DBTYPE_I2";
   rgParamBindInfo[0].pwszName     = NULL;
   rgParamBindInfo[0].ulParamSize  = sizeof(SHORT);

    // Get the session
hr = pIDBInitialize->QueryInterface(IID_IDBCreateSession,
    (void**)&pIDBCreateSession);
hr = pIDBCreateSession->CreateSession(NULL, IID_IDBCreateCommand,
    (IUnknown**) &pIDBCreateCommand);
 pIDBCreateSession->Release();

 // Create the command
hr = pIDBCreateCommand->CreateCommand(NULL, IID_ICommandText,
    (IUnknown**) &pICommandText);
 pIDBCreateCommand->Release();

 // The command requires the actual text and a language indicator

hr = pICommandText->SetCommandText(DBGUID_DBSQL, wSQLString);

 // Set parameter information
hr = pICommandText->QueryInterface(IID_ICommandWithParameters,
    (void**)&pICmdWithParams);
hr = pICmdWithParams->SetParameterInfo(nParams, rgParamOrdinals,
    rgParamBindInfo);
 pICmdWithParams->Release();

 // Prepare the command
hr = pICommandText->QueryInterface(IID_ICommandPrepare,
    (void**)&pICommandPrepare);
 if (FAILED(pICommandPrepare->Prepare(0)))
    {
    pICommandPrepare->Release();
    pICommandText->Release();
    return (E_FAIL);
    }
 pICommandPrepare->Release();

 // Create parameter accessors
 if(FAILED(myCreateParamAccessor(pICommandText, &hParamAccessor,
    &pIAccessor)))
    {
    pICommandText->Release();
    return (E_FAIL);
    }
```

```
   Params.pData = &Temp;    // pData is the buffer pointer
   Params.cParamSets = 1;    // Number of sets of parameters
   Params.hAccessor = hParamAccessor;    // Accessor to the parameters

   // Specify the parameter information
   for (UINT nCust = 0; nCust < 3; nCust++)
      {
        Temp.ajob_id = aEmployee[nCust].szjob_id;
        // Execute the command
        hr = pICommandText->Execute(NULL, IID_NULL, &Params, &cRowsAffected,
NULL);
        printf("%ld rows updated.\n", cRowsAffected);
      }

   pIAccessor->ReleaseAccessor(hParamAccessor, NULL);
   pIAccessor->Release();
   pICommandText->Release();

   return (NOERROR);
}
int main(int argc, char *argv[])
{
         IDBInitialize *pIDBInitialize = NULL;
         HRESULT hr = S_OK;
         static LCID lcid = GetSystemDefaultLCID();

         CoInitialize(NULL);

         if(FAILED(CoGetMalloc(MEMCTX_TASK, &g_pIMalloc)))
                  goto EXIT;

         if(FAILED(InitDSO(&pIDBInitialize)))
                  goto EXIT;

         if(FAILED(TestProperty(pIDBInitialize)))
                  goto EXIT;

         if(FAILED(DispalyRowset(pIDBInitialize)))
                  goto EXIT;

         if(FAILED(My_Sel_Command(pIDBInitialize)))
                  goto EXIT;

         if(FAILED(InsertWithParameters(pIDBInitialize)))
                  goto EXIT;

EXIT:    // Clean up and disconnect
         if (pIDBInitialize != NULL)
         {
                  hr = pIDBInitialize->Uninitialize();
                  pIDBInitialize->Release();
         }
```

```
        if (g_pIMalloc != NULL)
                g_pIMalloc->Release();

        CoUninitialize();

        return 0;
}
```

6.2 微软Visual Basic ADO代码示例

通过下列示例来获知当使用Visual Basic时，如何通过DBMaster OLE DB驱动创建一个连接。

```
'BeginNewConnection
Private Function GetNewConnection() As ADODB.Connection
                  Dim oCn As New ADODB.Connection
                  Dim sCnStr As String

               establish the connection
                  sCnStr = "Provider=DMOLE54; Data Source=oledbtest;;User
Id=SYSADM;Pwd=;"
                  oCn.Open sCnStr

                  If oCn.State = adStateOpen Then
                        Set GetNewConnection = oCn
                  End If

End Function
'EndNewConnection

Private Sub Sel_Para()
    On Error GoTo ErrHandler:

    Dim objConn As New ADODB.Connection
    Dim objCmd As New ADODB.Command
    Dim objParam As New ADODB.Parameter
    Dim objRs As New ADODB.Recordset

 ' Connect to the data source.
 'objConn.CursorLocation = adOpenDynamic

    Set objConn = GetNewConnection
    objCmd.ActiveConnection = objConn
    objCmd.Prepared = False

 ' Set the CommandText as a parameterized SQL query.
    objCmd.CommandText = "SELECT test_char " & _
                         "FROM test_datatype " & _
                         "WHERE test_char= ? "

 ' ----Char---- Create new parameter for Test_Char. Initial value is Test0.
    Set objParam = objCmd.CreateParameter("Test_Char", adChar, _
                    adParamInput, 5, "test0")
    objCmd.Parameters.Append objParam

 ' Execute once and display...
```

```
    Set objRs = objCmd.Execute

    Txt_Rst.Text = Txt_Rst.Text & vbCrLf & "Char Para=" & objParam.Value
    Do While Not objRs.EOF
        Txt_Rst.Text = Txt_Rst.Text & vbTab & "Result=" & objRs(0)
        objRs.MoveNext
    Loop

 'clean up
    objRs.Close
    Set objCmd = Nothing

    objConn.Close
    Set objRs = Nothing
    Set objConn = Nothing

    Set objParam = Nothing
Exit Sub

ErrHandler:
    'clean up
    If objRs.State = adStateOpen Then
        objRs.Close
    End If

    If objConn.State = adStateOpen Then
        objConn.Close
    End If

    Set objRs = Nothing
    Set objConn = Nothing
    'Set objCmd = Nothing

    If Err <> 0 Then
        MsgBox Err.Source & "-->" & Err.Description, , "Error"
    End If
End Sub
```

6.3 Visual C# ADO.NET示例

本例演示了如何使用C#语言并利用DBMaster OLE DB Provider来访问DBMaster。

注意 *该示例通过OleDbCommand方式来展示如何将普通类型数据插入到DBMaster数据库中。*

```
/*****************************************************************
The table schema used in this sample as following shows:
create table SYSADM.OrdinaryType (
   C00_ID          SERIAL(1),
   C01_INT16       SMALLINT         default null ,
   C02_INT32       INTEGER          default null ,
   C03_FLOAT       FLOAT            default null ,
   C04_DOUBLE      DOUBLE           default null ,
   C05_DECIMAL     DECIMAL(20, 4)   default null ,
   C06_BINARY      BINARY(10)       default null ,
   C07_CHAR        CHAR(20)         default null ,
   C08_VARCHAR     VARCHAR(20)      default null ,
   C09_NCHAR       NCHAR(20)        default null ,
   C10_NVARCHAR    NVARCHAR(20)     default null ,
   C11_DATE        DATE             default null ,
   C12_TIME        TIME             default null ,
   C13_TIMESTAMP   TIMESTAMP        default null )
in DEFTABLESPACE  lock mode page  fillfactor 100 ;
*****************************************************************/

using System;
using System.Data;
using System.Data.OleDb;  //This namespaces declarations OLE DB Provider

public class InsOrdinaryType_1
{
        public static void Main()
        {
                string           myCNString;
                string           myCMString;
                OleDbConnection  myCN;
                OleDbCommand     myCM;
                short    c_int16 = 12345;
                int      c_int32 = 123456;
                float    c_float = 12345678.9012F;
                double   c_double = 1234567890.1234567;
                decimal  c_decimal = 1234567890123.4567M;
                string   c_binary = "AAAAABBBBB";
                string   c_binary1 = "'41414141414242424242'x";
                byte[]   c_binary2 = new byte[10];
```

```
                for(int i=0;i<10;i++) c_binary2[i]=(byte)'A';
                string   c_char = "AAAAABBBBBCCCCCDDDDD";
                string   c_varchar = "AAAAABBBBBCCCCCDDDDD";
                string   c_nchar = "AAAAABBBBBCCCCCDDDDD";
                string   c_nvarchar = "AAAAABBBBBCCCCCDDDDD";
                DateTime c_date = new DateTime(2006,5,22);
                string   c_date1 = "2006/5/22";
                TimeSpan c_time = new TimeSpan(0,16,35,00,000);
                string   c_time1 = "16:35:00";
                DateTime c_timestamp = new DateTime(2006,5,22,16,35,00,000);
                string   c_timestamp1 = "2006/5/22 16:35:00.000";

//insert data by static SQL command string
    //create a connection string
                myCNString =  "Provider=DMOLE54;Data Source=DBNAME;";
                myCNString += "User Id=SYSADM;Password=;";
                myCMString =  "insert into OrdinaryType(";
                myCMString += "c01_int16,c02_int32,c03_float,c04_double";
                myCMString += ",c05_decimal,c06_binary,c07_char";
                myCMString += ",c08_varchar,c09_nchar,c10_nvarchar";
                myCMString += ",c11_date,c12_time,c13_timestamp) ";
                myCMString += " values(" + c_int16 + "," + c_int32 ;
                myCMString += "," + c_float + "," + c_double ;
                myCMString += "," + c_decimal + ",'" + c_binary ;
                myCMString += "','" + c_char + "','" + c_varchar ;
                myCMString += "','" + c_nchar + "','" + c_nvarchar ;
                myCMString += "','" + c_date1 + "','" + c_time1 ;
                myCMString += "','" + c_timestamp1 ;
                myCMString += "' );";
    //establish and open a new connection
                myCN = new OleDbConnection(myCNString);
                myCM = new OleDbCommand(myCMString,myCN);

                try{
                myCN.Open();
                Console.WriteLine("-----------Connection opened-------------");
                        Console.WriteLine(myCMString);
                        int inserted = myCM.ExecuteNonQuery();
                        Console.WriteLine("{0} rows inserted.",inserted);
                        myCN.Close();
                }catch(Exception ex){
                        Console.WriteLine(ex.Message);
                }finally{
                        if(myCN !=null) myCN.Close();
                Console.WriteLine("--------------------------------");
                        Console.WriteLine("connection closed");
                }
                Console.WriteLine("press ENTER to continue...");
                Console.Read();

//insert data by SQL command with parameter
                myCMString = "insert into OrdinaryType(";
```

```
                myCMString += "c01_int16,c02_int32,c03_float,c04_double";
                myCMString += ",c05_decimal,c06_binary,c07_char";
                myCMString += ",c08_varchar,c09_nchar,c10_nvarchar";
                myCMString += ",c11_date,c12_time,c13_timestamp) ";
                myCMString += " values(?,?,?,?,?,?,?,?,?,?,?,?,?);" ;

                myCM = new OleDbCommand(myCMString,myCN);
                myCM.Parameters.Add("@int16",OleDbType.SmallInt).Value =
c_int16;
                myCM.Parameters.Add("@int32",OleDbType.Integer).Value = c_int32;
                myCM.Parameters.Add("@float",OleDbType.Single).Value = c_float;
                myCM.Parameters.Add("@double",OleDbType.Double).Value =
c_double;
                myCM.Parameters.Add("@decimal",OleDbType.Decimal).Value =
c_decimal;
                myCM.Parameters.Add("@binary",OleDbType.Binary,10).Value =
c_binary2;
                myCM.Parameters.Add("@char",OleDbType.Char,20).Value = c_char;
                myCM.Parameters.Add("@varchar",OleDbType.VarChar,20).Value =
c_varchar;
                myCM.Parameters.Add("@nchar",OleDbType.WChar,20).Value =
c_nchar;
                myCM.Parameters.Add("@nvarchar",OleDbType.VarWChar,20).Value =
c_nvarchar;
                myCM.Parameters.Add("@date",OleDbType.DBDate).Value = c_date;
                myCM.Parameters.Add("@int16",OleDbType.DBTime).Value = c_time;
                myCM.Parameters.Add("@int16",OleDbType.DBTimeStamp).Value =
c_timestamp;

                foreach(OleDbParameter para in myCM.Parameters)
                {
                        Console.WriteLine(para.Value);
                }
                try{
                        myCN.Open();
                        Console.WriteLine("-----------Connection opened--------
-----");
                        int inserted = myCM.ExecuteNonQuery();
                        Console.WriteLine("{0} rows inserted.",inserted);
                        myCN.Close();
                }catch(Exception ex){
                        Console.WriteLine(ex.Message);
                }finally{
                        if(myCN !=null) myCN.Close();
                        Console.WriteLine("---------------------------------
");
                        Console.WriteLine("connection closed");
                }
                Console.WriteLine("press ENTER to exit...");
                Console.Read();
        }
}
```