



DBMaker

OLEDB User's Guide



CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive

Santa Clara, CA 95050, U.S.A.

www.casemaker.com

www.casemaker.com/support

©Copyright 1995-2017 by CASEMaker Inc.

Document No. 645049-237132/DBM541-M02282017-OLED

Publication Date: 2017-02-28

All rights reserved. No part of this manual may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the manufacturer.

For a description of updated functions that do not appear in this manual, read the file named README.TXT after installing the CASEMaker DBMaker software.

Trademarks

CASEMaker, the CASEMaker logo, and DBMaker are registered trademarks of CASEMaker Inc. Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corp. UNIX is a registered trademark of The Open Group. ANSI is a registered trademark of American National Standards Institute, Inc.

Other product names mentioned herein may be trademarks of their respective holders and are mentioned only for information purposes. SQL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

Notices

The software described in this manual is covered by the license agreement supplied with the software.

Contact your dealer for warranty details. Your dealer makes no representations or warranties with respect to the merchantability or fitness of this computer product for any particular purpose. Your dealer is not responsible for any damage caused to this computer product by external forces including sudden shock, excess heat, cold, or humidity, nor for any loss or damage caused by incorrect voltage or incompatible hardware and/or software.

Information in this manual has been carefully checked for reliability; however, no responsibility is assumed for inaccuracies. This manual is subject to change without notice.

Contents

- 1 Introduction 1-1**
 - 1.1 Additional Resources 1-2**
 - 1.2 Technical Support 1-3**
 - 1.3 Document Conventions 1-4**
- 2 Supported Data Types 2-1**
- 3 COM Object Definitions..... 3-1**
 - 3.1 Data Source 3-2**
 - 3.2 Sessions 3-3**
 - 3.3 Commands 3-4**
 - 3.4 Rowsets 3-5**
- 4 Interfaces (OLE DB) 4-1**
 - 4.1 OLE DB Provider for DBMaker Supported Interfaces..... 4-2**
- 5 Using the OLE DB Provider 5-1**
 - 5.1 Setting up the envirement..... 5-2**
 - 5.2 Invoking the OLEDB provider 5-3**
 - Adding a Foreign Key 5-3

5.3	Programming an OLE DB application	5-4
	Establishing a New Connection to a Data Source	5-4
	Executing a Command via OLE DB Driver	5-5
	Processing the Returned Results	5-5
6	Samples	6-1
6.1	OLE DB Consumer Application Microsoft Visual C++ Examples	6-2
6.2	ADO Code Examples in Microsoft Visual Basic	6-21
6.3	ADO.NET Code Examples in Visual C#	6-24

1 Introduction

OLE DB is a set of Component Object Model (COM) interfaces. The COM interfaces provide applications with uniform access to data stored in diverse DBMS and non-DBMS information sources. In addition to supporting many information sources, OLE DB also supports the implementation of database services. Utilizing these interfaces data consumers easily access data through a consistent method. With OLE DB consumers need not consider the storage location of the data, the format of the data, or the type of data.

OLE DB Provider for DBMaker is designed for accessing the DBMaker database system. OLE DB Provider for DBMaker allows OLE DB programmers to easily develop high performance consumer applications using a host of bundled interfaces. OLE DB Provider for DBMaker is specifically designed for use with DBMaker and is incompatible for accessing other information sources.

1.1 Additional Resources

DBMaker provides many other user's guides and reference manuals in addition to this reference.

For more information on a particular subject, consult one of these books:

- ◆ For an introduction to DBMaker's capabilities and functions, refer to the *DBMaker Tutorial*.
- ◆ For more information on designing, administering, and maintaining a DBMaker database, refer to the *Database Administrator's Guide*.
- ◆ The *SQL Command and Function Reference* provides more information about the SQL language implemented by DBMaker.
- ◆ The *ESQL/C Programmer's Guide* is an excellent resource on the ESQL/C language implemented by DBMaker.
- ◆ For more information on the native ODBC API and JDBC API, refer to the *ODBC Programmer's Guide* and *JDBC Programmer's Guide*.
- ◆ For more information on the DCI COBOL Interface, refer to the *DCI User's Guide*.
- ◆ The *dmSQL User's Guide* offers detailed information on using dmSQL.
- ◆ The *Error and Message Reference* provides detailed information about error and warning messages.
- ◆ The *JDBA Tool User's Guide*, *JServer Manager User's Guide*, and *JConfiguration Tool Reference* each offer information on configuring and managing databases using DBMaker's JTools.
- ◆ The *DBMaker SQL Stored Procedure User's Guide* provides detailed information about the SQL stored procedure language implemented in DBMaker.

1.2 Technical Support

CASEMaker provides thirty days of complimentary email and phone support during the evaluation period. When software is registered, the support period is extending an additional thirty days for a total of sixty days. However, CASEMaker will continue to provide email support (free of charges) for bugs reported after the complimentary support or registered support expires.

For most products, support is available beyond sixty days and may be purchased for twenty percent of the retail price of the product. Please contact sales@casemaker.com for details and prices.

CASEMaker support contact information, by post mail, phone, or email, for your area is at: www.casemaker.com/support. We recommend searching the most current database of FAQ's before contacting CASEMaker support staff.

Please have the following information available when phoning support for a troubleshooting enquiry or include this information in your correspondence:

- ◆ Product name and version number
- ◆ Registration number
- ◆ Registered customer name and address
- ◆ Supplier/distributor where product was purchased
- ◆ Platform and computer system configuration
- ◆ Specific action(s) performed before error(s) occurred
- ◆ Error message and number, if any
- ◆ Any additional information deemed pertinent

1.3 Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and CommandLine conventions also have a second setting used with indentation.



CONVENTION	DESCRIPTION
<i>Italics</i>	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but is replaced by the actual name. Italics also introduce new words, and are occasionally used for emphasis in text.
Boldface	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
SMALL CAPS	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.
 Procedure	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow.
 Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
CommandLine	Indicates text, as it should appear on a text-delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file.

Table 1-1 Document Conventions

2 Supported Data Types

The following table shows one way that a DBMaker Provider might map its data types to OLE DB data types.

DBMaker data type	OLE DB type indicator	SQL type
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
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_DBTIMES TAMP	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

NOTE *OLE DB Provider for DBMaker supports 38 levels of precision for the decimal data type. The data type mapping varies in OLE DB Provider for DBMaker according to the method. For example, consider the ADO and the ADO.NET methods. The char data type is mapped to DBTYPE_BSTR using the ADO method, however, using the ADO.NET method, the same data type is mapped to DBTYPE_WSTR.*

3 COM Object Definitions

OLE DB uses Microsoft Corporation's standard for Universal Data Access. This is called the COM infrastructure. Similar to the ODBC system, OLE DB provides a set of APIs, however, unlike ODBC, OLE DB APIs are based entirely on COM. In other words, operations on abstract objects, such as Data source, Session, Command, and Rowset, can be accessed via COM. OLE DB Provider for DBMaker supports four objects: Data source Object, Session Object, Command Object, and Rowset Object. These objects are described in the following section of this chapter.

3.1 Data Source

A Data source Object is a COM object through which a consumer connects to a provider's underlying data store. OLE DB Provider for DBMaker defines its own Data source Object class. To connect to the provider, a consumer must create and initialize an instance of this class. Data source Objects are like factories for session objects.

The Data Source Object cotype is defined following table.

```
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 Sessions

A Session object represents a single connection to a DBMaker database. The Session object exposes the interfaces that allow data access and data manipulation. A single Data source Object may be able to create multiple sessions. Session objects are factories for Command and Rowset objects, which provide methods for creating Command objects and rowsets and modifying tables and indexes. Session objects can also function as factories for transaction objects. Transaction objects are used for controlling nested transactions.

The Session object is removed from memory and the connection is dropped after all references to the Session object are released. The Session object cotype is defined below.

```
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 Commands

Commands exist in one of four states: Initial, Unprepared, Prepared, or Executed. Parameters are used with commands to bind to consumer variables at execution time. A command returns either a single result or multiple results when executed. The single result can be either a rowset object or a row count (i.e., the number of rows affected by a command that updates, deletes, or inserts rows). The command can also return multiple results. If the command text comprises multiple, separate text commands, such as a batch of SQL statements, or if more than one set of parameters is passed to a command, then the results must be returned in a multiple results object.

The Command object is used to execute an OLE DB Provider for DBMaker text command. Text commands are expressed in the OLE DB Provider for DBMaker language, and are generally used for creating a rowset, for example, executing a SQL SELECT statement.

The Command object cotype is defined as follows.

```
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 Rowsets

Rowsets are the central objects that enable OLE DB components to expose and manipulate data in tabular form. A Rowset object is a set of rows each having columns of data. For example, OLE DB Provider for DBMaker presents data and metadata to consumers in the form of rowsets. The use of rowsets throughout OLE DB makes it possible to aggregate components that consume or produce data through the same object.

The Rowset object cotype is defined as follows.

```
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 Interfaces (OLE DB)

Interfaces are a group of semantically related functions that provide access to a COM object. Each OLE DB interface defines a contract that allows objects to interact according to the Component Object Model (COM). OLE DB provides many interface implementations. Most interfaces can also be implemented by developers designing OLE DB applications. This chapter summarizes the OLE DB interfaces that are supported by the current version of the OLE DB Provider for DBMaker.

4.1 OLE DB Provider for DBMaker Supported Interfaces

The following table summarizes the OLE DB interfaces that are supported by the current version of the OLE DB Provider for DBMaker. For more information about the interfaces, please refer to MSDN.

OBJECT	INTERFACE	SUPPORTED
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

Interfaces (OLE DB) 4

	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

Interfaces (OLE DB) 4

	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

Table 4-1 OLE DB Provider for DBMaker Supported Interfaces

5 Using the OLE DB Provider

This section contains detailed information about using the OLE DB Provider for DBMaker. It is divided into three parts, as follows:

- 1.** Setting up the environment
- 2.** Invoking the OLE DB provider
- 3.** Programming an OLE DB application

5.1 Setting up the environment

Registry entries: To enable the OLE DB Provider for DBMaker to work with users' applications.

You should register GUID of OLE DB Provider for DBMaker.

For DBMaker and DBMaster normal version, the GUID of OLE DB Provider will be automatically registered when DBMaker is installed.

For DBMaker and DBMaster bundle version, users should register it in the command line with the following command:

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

After the GUID of OLE DB Provider is successfully registered, users can use OLEDB in DBMaker bundle version.

5.2 Invoking the OLEDB provider

According to users' programming needs, the OLE DB Provider for DBMaker (dmole54.dll) can be invoked with a variety of methods. Calling CoCreateInstance on IDBInitialize is traditionally used in OLE DB to open a data source object.

Adding a Foreign Key

The OLE DB Provider for DBMaker is invoked from ADO or ADO.net with a typical connection string, as follow:

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

In this connection string, dmole54 is oledb provider name. The oledb provider name is different for different version of DBMaker and DBMaster. As follows:

For DBMaker normal version, the oledb provider name is dmole54

For DBMaster normal version, the oledb provider name is dmole54J

For DBMaker bundle version, the oledb provider name is dmole54B

For DBMaster bundle version, the oledb provider name is dmole54JB

When called from ADO or ADO.NET, the OLE DB services are automatically enabled.

5.3 Programming an OLE DB application

Programming an OLE DB application involves three steps:

1. Establishing a new connection to a data source.
2. Executing a command via OLE DB driver
3. Processing the returned results.

Establishing a New Connection to a Data Source

Creating an instance of the data source object of the provider is the first task of an OLE DB consumer. The basic steps for creating a data source are:

1. Initialize the COM library by calling **CoInitialize(NULL)**.
2. Create an instance of a data source object by calling the **CoCreateInstance** method. The syntax is:

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

A unique class identifier (CLSID) identifies each OLE DB provider. For DMOLE54, the class identifier is DBMAKER_DMOLE54.

3. The data source object exposes the **IDBProperties** interface. The consumer uses the **IDBProperties** to provide basic authentication information such as server name, database name, user ID, and password. These properties are set by calling the **IDBProperties::SetProperties** method.
4. The data source object also exposes the **IDBInitialize** interface. Establish a connection to the data source by calling the **IDBInitialize::Initialize** method.

Executing a Command via OLE DB Driver

The consumer calls the `IDBCreateSession::CreateSession` method to create a session after the connection to a data source is established. The session functions as a command, rowset, or transaction factory.

Session objects can create Command objects. The command object of OLE DB Provider for DBMaker supports the execution of SQL commands. Additionally, the Command object of OLE DB Provider for DBMaker supports multiple parameters.

Consider the following example of executing a command. A consumer wants to execute the command: `SELECT * FROM Authors`. To begin, the consumer requests the `IDBCreateCommand` interface. The consumer can execute the `IDBCreateCommand::CreateCommand` method to create a command object and then request the `ICommandText` interface. The `ICommandText::SetCommandText` method is used for specifying the command to be executed. Lastly, the command is executed using the `Execute` command. Commands like `SELECT * FROM Authors` produce a result set (rowset) object.

The consumer requests the `IOpenRowset` interface for working directly with individual tables or indexes. The `IOpenRowset::OpenRowset` method opens and returns a rowset that includes all rows from a single base table or index.

Processing the Returned Results

The consumer must retrieve and access data in a rowset when the rowset object is produced by either the execution of a command or the generation of a rowset object directly by the provider.

Rowsets are central objects enabling all OLE DB data providers to expose data in tabular form. The rowset comprises a set of rows. Each row contains column data. A rowset object facilitates access by exposing various interfaces. For example, `IRowset` is an interface containing methods for sequentially fetching rows from the rowset. `IAccessor` is an interface for defining a group of column bindings describing how tabular data is bound to consumer program variables. The `IColumnInfo` interface

provides information about columns in the rowset. The **IRowsetInfo** interface provides information about rowset.

The consumer can call the **IRowset::GetData** method to retrieve a row of data from the rowset into a buffer. The consumer must describe the buffer using a set of **DBBINDING** structures before **GetData** is called. During data retrieval, the provider uses information in each binding to determine where and how to retrieve data from the consumer buffer. When setting data in the consumer buffer, the provider uses information in each binding to determine where and how to return data in the consumer buffer.

After the **DBBINDING** structures are specified, an accessor is created by calling the **IAccessor::CreateAccessor** method. An accessor is a collection of bindings and is used to retrieve or set the data in the consumer buffer.

6 Samples

The sample provided here demonstrates rowset programming and an object model for an OLE DB consumer. The sample creates a data source, a session, and rowset objects; allows the user to display and navigate the rows in the rowset; and handles errors. Command line switches are used to specify when an enumerator, class ID, user prompt, or connection string is used to create the data source object, a command is used to create the rowset, and so on.

NOTE *There are three code examples in this chapter. The C++ sample program shows a basic implementation of the OLE DB Provider for DBMaker. The Visual Basic sample program accesses OLE DB Provider for DBMaker through ADO methods. The C# sample program accesses OLE DB Provider for DBMaker through ADO.NET methods.*

6.1 OLE DB Consumer Application Microsoft Visual C++ Examples

This example demonstrates how to initialize a data source and how to access a database of DBMaker by OLE DB provider for DBMaker in C++.

```
#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_DBMAKER_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_CANSROLLBACKWARDS;
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 = {};
    ULONG    cRowsObtained = 0;
    HROW     rghRows[100] = {};
    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(rgRows[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].pszDataSourceType = (unsigned short *) L"DBTYPE_I2";
rgParamBindInfo[0].pszName = 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 ADO Code Examples in Microsoft Visual Basic

Use the following code example to learn how to create a connection via DBMaker OLE DB driver when writing in Visual Basic.

```
'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

ErrorHandler:
'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 ADO.NET Code Examples in Visual C#

This example demonstrates how to access DBMaker via OLE DB provider for DBMaker when writing in C#.

NOTE *This example uses OleDbCommand method to show insert ordinary type data into the database of DBMaker.*

```

/*****
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 = "1414141414142424242'x";
byte[]  c_binary2 = new byte[10];
for(int i=0;i<10;i++) c_binary2[i]=(byte) 'A';
string  c_char = "AAAAABBBBBCCCCDDDD";
string  c_varchar = "AAAAABBBBBCCCCDDDD";
string  c_nchar = "AAAAABBBBBCCCCDDDD";
string  c_nvarchar = "AAAAABBBBBCCCCDDDD";
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();
    }
}

```

