



DBMaster

OLE DB ユーザガイド

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive
Santa Clara, CA 95050, U.S.A.

Contact Information:

CASEMaker US Division

E-mail : info@casemaker.com

Europe Division

E-mail : casemaker.europe@casemaker.com

Asia Division

E-mail : casemaker.asia@casemaker.com(Taiwan)

E-mail : info@casemaker.co.jp(Japan)

www.casemaker.com

www.casemaker.com/support

©Copyright 1995-2010 by Syscom Computer Engineering Co.

Document No. 645049-234292/DBM52J-M04082010-OLED

発行日 :2010-04-08

ALL RIGHTS RESERVED.

本書の一部または全部を無断で、再出版、情報検索システムへ保存、その他の形式へ転作することは禁止されています。

本文には記されていない新しい機能についての説明は、CASEMakerのDBMasterをインストールしてからREADME.TXTを読んでください。

登録商標

CASEMaker、CASEMakerのロゴは、CASEMaker社の商標または登録商標です。

DBMasterは、Syscom Computer Engineering社の商標または登録商標です。

Microsoft、MS-DOS、Windows、Windows NTは、Microsoft社の商標または登録商標です。

UNIXは、The Open Groupの商標または登録商標です。

ANSIは、American National Standards Institute, Incの商標または登録商標です。

ここで使用されているその他の製品名は、その所有者の商標または登録商標で、情報として記述しているだけです。SQLは、工業用語であって、いかなる企業、企業集団、組織、組織集団の所有物でもありません。

注意事項

本書で記述されるソフトウェアは、ソフトウェアと共に提供される使用許諾書に基づきます。

保証については、ご利用の販売店にお問い合わせ下さい。販売店は、特定用途への本コンピュータ製品の商品性や適合性について、代表または保証しません。販売店は、突然の衝撃、過度の熱、冷氣、湿度等の外的な要因による本コンピュータ製品へ生じたいかなる損害に対しても責任を負いません。不正な電圧や不適合なハードウェアやソフトウェアによってもたらされた損失や損害も同様です。

本書の記載情報は、その内容について十分精査していますが、その誤りについて責任を負うものではありません。本書は、事前の通知無く変更することがあります。

コンテンツ

1	イントロダクション.....	1-1
2	サポートしているデータ型.....	2-1
3	COMオブジェクト定義	3-1
3.1	データソース	3-1
3.2	セッション.....	3-2
3.3	コマンド	3-3
3.4	行セット	3-4
4	インターフェース (OLE DB)	4-1
4.1	DBMaster OLE DB プロバイダのサポートするインターフェース	4-1
5	OLE DBアプリケーション作成方法	5-1
5.1	データソースへの新規コネクション確立	5-1
5.2	OLE DBドライバ経由でのコマンド実行	5-2
5.3	結果返答プロセス	5-3
6	サンプル	6-1
6.1	OLE DB Consumer Application Microsoft Visual C++ Examples	6-1
6.2	ADO Code Examples in Microsoft Visual Basic	6-20
6.3	ADO.NET Code Examples in Visual C#.....	6-22

1 イントロダクション

OLE DBはプロバイダがアプリケーションとドライバDBMSに格納されたデータに対して一貫したアクセスを提供するコンポーネントオブジェクトモデル(COM)インターフェイスセットです。DBMSではないソースとデータベースサービス実行機能も使用可能です。これらのインターフェイスを活用することで、データの格納場所、データフォーマット、データ型を意識することなく、同じ方法にてアクセスすることが可能になります。DBMasterのOLE DBプロバイダはDBMasterのデータベースシステムにアクセスするために設計されています。OLE DBプログラマはDBMasterのOLEプロバイダインターフェイスの高いパフォーマンスによるアプリケーション開発が実現できます。DBMasterのOLE DBはDBMasterに特化したものですので他のソースへのアクセスには適していません。

2 サポートしているデータ型

この表では DBMaster のプロバイダがマッピング対応する OLE DB のデータ型の一覧で表示しています。

DBMaster 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

DBMaster data type	OLE DB type indicator	SQL type
file	DBTYPE_BYTES	SQL_LONGVARBI NARY 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_TIMES TAMP
nchar	DBTYPE_BSTR DBTYPE_WSTR	SQL_WCHAR
nvarchar	DBTYPE_BSTR DBTYPE_WSTR	SQL_WVARCHAR
blob	DBTYPE_BSTR DBTYPE_BYTES	SQL_LONGVARBI NARY
clob	DBTYPE_BSTR, DBTYPE_WSTR	SQL_LONGVARC HAR
nclob	DBTYPE_BSTR DBTYPE_WSTR	SQL_WLONGVAR CHAR

NOTE:

1. DBMaster の OLE DB プロバイダは精度が 38 桁の decimal データに対応しています。
2. DBMaster の OLE DB プロバイダにおけるデータマッピングはメソッドによって変わります。

例) ADO と ADO.NET のメソッドを決定するとき、ADO メソッドでは DBTYPE_BSTR にマップし、ADO.NET メソッドで同じデータでも DBTYPE_WSTR を使用します。

3 COM オブジェクト定義

OLE DBはMicrosoft社の標準Universal Data Accessを使用します。COMインフラストラクチャと呼ばれるものです。ODBCに似たシステムでOLE DBはAPIのセットを提供します。しかしながらOLE DBのAPIは完全にCOMに基づいたものになっています。通信はCOMに基づき、データソース、セッション、コマンド、行セットといったような理論的なオペレーションとして扱われます。DBMasterのOLE DBプロバイダは4つのオブジェクトをサポートします。(データソースオブジェクト、セッションオブジェクト、コマンドオブジェクト、行セットオブジェクト)これらについては以下の章にて記述があります。

3.1 データソース

OLE DBではデータソースオブジェクトは使用者がプロバイダの基本データにアクセスするCOMオブジェクトです。DBMasterのOLE DBは独自のデータソースオブジェクトクラスを定義しています。プロバイダに接続する場合使用者はこのクラスのインスタンスを作成、初期化する必要があります。データソースオブジェクトはセッションオブジェクトを生成します。

データソースオブジェクトのcotypeは以下のように定義されます。

```
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 セッション

セッションオブジェクトはDBMasterのデータベースに接続するシングルコネクションに代表されます。セッションオブジェクトはデータソースへのアクセスと操作を許可するインターフェイスと接する部分です。シングルデータソースオブジェクトであればマルチセッションを確立することも可能と考えられます。セッションオブジェクトはコマンドオブジェクト、行セットの生成方法を提供するコマンドと行セットを生成し、表や索引を変更します。セッションオブジェクトはネステッドトランザクションをコントロールするために使用されるトランザクションオブジェクトの働きもします。

最終的にセッションオブジェクトの参照は解放され、セッションオブジェクトはメモリーから消去され接続も切断されます。セッションオブジェクトのcotypeは以下のように定義されています。

```
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 コマンド

コマンドは4つの状態のうちの1つになります。Initial, Unprepared, Prepared, Executedです。実行時、パラメータはコマンドと同時に使用され、変数をバインドします。実行されるとコマンド結果として行セット、もしくは数値によるUPDATE、DELETE、INSERTされた行カウントを返します。コマンドはマルチ結果オブジェクトとして複数の結果を返すことも可能です。コマンドテキストが複数で構成されるとき、バッチSQLのように分割されるか複数のパラメータセットが1つのコマンドとして通ります。コマンドオブジェクトはDBMasterのテキストコマンドプロバイダを実行するために使用されます。テキストコマンドはDBMasterのプロバイダ言語にて表現され、一般的にSQL SELECT文を実行するような行セットを生成するために使われます。

コマンドオブジェクトのcotypeは以下のものとなります。

```

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 行セット

行セットはOLE DBコンポーネントがフォーマットのデータ操作を可能にするための中心的なオブジェクトです。行セットとは複数の行から成るセットで各行がカラムにデータを持ちます。例えばDBMasterのプロバイダはコンシューマーにメタデータ形式でデータを提供します。クエリプロセッサはクエリ結果をフォームに返します。OLE DBを通して行セット使用することで同じオブジェクトを通過したデータをの統合を可能にします。

行セットオブジェクトのcotypeは以下のものとなります。

```
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 インターフェース (OLE DB)

インターフェースはCOMオブジェクトにアクセスする機能の意味上のグループです。それぞれOLE DBインターフェースはコンポーネントオブジェクト (COM)が相互作用できるように規約を定義します。OLE DBには複数の実行インターフェースがあります。大抵のインターフェースは開発者が独自に設計したOLE DBアプリケーションにより実行できます。このチャプターは現行のDBMasterのOLE DBプロバイダが提供するインターフェースの概要を紹介します。

4.1 DBMaster OLE DB プロバイダのサポートするインターフェース

以下は現行の DBMaster OLE DB プロバイダがサポートしている OLE DB インターフェースの一覧です。各インターフェースの詳しい内容についてはマイクロソフト社が提供する技術情報 MSDN をご覧ください。

Object	Interface	Supported
コマンド	Iaccessor	Yes
	IColumnsInfo	Yes
	IColumnsRowset	Yes

Object	Interface	Supported
	ICommand	Yes
	ICommandPersist	No
	ICommandPrepare	Yes
	ICommandProperties	Yes
	ICommandText	Yes
	ICommandWithParameters	Yes
	IConvertType	Yes
	IDBInitialize	No
	ISupportErrorInfo	No
データソース	IConnectionPointContainer	No
	IDBAsynchStatus	No
	IDBAsynchNotify	No
	IDBCreateSession	Yes
	IDBInfo	Yes
	IDBInitialize	Yes
	IDBProperties	Yes
	IPersist	Yes
	IPersistFile	No
	ISupportErrorInfo	No

Object	Interface	Supported
エラー	IErrorInfo	No
行セット	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

Object	Interface	Supported
セッション	IAAlterIndex	No
	IAAlterTable	No
	IBindResource	No
	IConnectionPointContainer	No
	ICreateRow	No
	IDBASynchStatus	No
	IDBCreateCommand	Yes
	IDBInitialize	No
	IDBSchemaRowset	Yes
	IGetDataSource	Yes
セッション	IIndexDefinition	No
	IOpenRowset	Yes
	ISessionProperties	Yes
	ISupportErrorInfo	No
	ITableDefinition	No
	ITransaction	Yes
	ITransactionJoin	Yes
	ITransactionLocal	Yes
	ITransactionObject	No

5 OLE DB アプリケーション作成方法

OLE DB アプリケーションのプログラミングには下記手順が含まれます。

1. データソースへの新規コネクションを確立
2. OLE DB ドライバ経由でコマンド実行
3. 結果返答プロセス

5.1 データソースへの新規コネクション確立

プロバイダのデータソースオブジェクトへのインスタンス作成は OLE DB コンシューマーが最初に行うタスクです。

基本的なデータソース作成手順はこちら：

1. **ColInitialize(NULL)** 呼び出しによるCOMライブラリの初期化
2. **CoCreateInstance** メソッド呼び出しによるデータソースオブジェクトのインスタンス作成。構文は以下のものになります。：

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

一意のクラス識別子(CLSID)は各 OLE DB プロバイダを識別します。
DMOLE43 ではクラス識別子は CLSID_DMOLE43 です。

3. データソースオブジェクトは **IDBProperty** インターフェースを使用します。コンシューマーは **IDBProperty** をサーバー名やデータベース名、ユーザ ID/パスワードをといった基本認証を行うために **IDBProperty** を使用します。これらのプロパティは **IDBProperties::SetProperties** メソッドにより設定されます。
4. データソースオブジェクトは **IDBInitialize** インターフェースによって使用され、**IDBInitialize::Initialize** メソッドを呼び出すことでデータソースへのコネクションを確立します。

5.2 OLE DB ドライバ経由でのコマンド実行

データソースへのコネクション確立後に **IDBCreateSession::CreateSession** メソッドを呼び出してセッションを作成します。コマンド、行セット、トランザクション生成といったものはセッションファンクションとして扱われます。

セッションオブジェクトはコマンドオブジェクトを作成することができます。OLE DB プロバイダのコマンドオブジェクトは **DBMaster** の SQL コマンド実行をサポートしており、加えて、**DBMaster** の OLE DB プロバイダのコマンドオブジェクトは複数のパラメータをサポートしています。

下記のコマンド実行例を考えてみます。コンシューマーはコマンド **SELECT * FROM Authors** を実行しようとしています。始めに、**IDBCreateCommand** を要求します。コマンドオブジェクト作成、 **ICommandText** インターフェースを要求する **IDBCreateCommand::CreateCommand** メソッドを実行可能です。 **ICommandText::SetCommandText** メソッドはコマンド実行の設定に使用されます。最後に **Execute** コマンドを使用してコマンドが実行されます。**SELECT * FROM Authors** のようなコマンドは結果セット(行セット)を生成します。

ワーキングディレクトリ、各表、索引に対して IOpenRowset インターフェースを要求します。IOpenRowset::OpenRowset メソッドは単一の表もしくは索引のすべての行を含む行セットをオープンし、返答します。

5.3 結果返答プロセス

コンシューマーは行セットオブジェクトがコマンドもしくは行セット生成をプロバイダが直接行ったとき、行セット内のデータ検索/アクセスします。

行セットは全ての OLE DB データプロバイダがデータを表形式で表示するために中核となるオブジェクトです。行セットは複数の行のセットから構成されます。各行はカラムデータを含みます。行セットは多くのインターフェースからのアクセスを容易にします。例えば、 IRowset は行セットからシーケンシャルに行をフェッチするメソッドを含んだインターフェースです。IAccessor は表データがどのようにプログラム変数にバインドされているか記述されているカラムグループを定義するインターフェースです。IColumnInfo は行セットのカラム情報を提供するインターフェースです。IRowsetInfo インターフェースは行セットの情報を提供します。

コンシューマーは IRowset::GetData メソッドを呼び出し、行とデータを行セットから検索しバッファに置きます。GetData が呼び出される前にコンシューマーは DBBINDING 構造のセットのバッファを記述する必要があります。コンシューマーバッファからデータをどこから、どのように検索するかを決定するためにデータ検索中にプロバイダは各バインドの情報を使用します。

DBBINDING 構造が定義された後、IAccessor::CreateAccessor メソッドによってアクセサーが作成されます。アクセサーはバインディングの集合でコンシューマーバッファでのデータ検索/設定に使用されます。

6 サンプル

以下のサンプルは行セットプログラミングとOLE DBコンシューマーのオブジェクトのデモです。サンプルではデータソース、セッション、行セットオブジェクトを作成;行セット上で行の表示と操作;エラー操作を行えます。コマンドラインスイッチは計数、クラスID、ユーザプロンプト、接続文字列がデータソースオブジェクト作成に使われる場合、コマンドが行セット作成に使用される場合などに使用されます。

注: このチャプターでは3つのコードを例示します。C++サンプルプログラムはDBMasterのOLE DBプロバイダ基本実行を例示。Visual BasicサンプルプログラムはADO経由でのDBMaster OLE DBプロバイダへのアクセスを例示。C#サンプルプログラムはADO.NET経由でのDBMaster OLE DBプロバイダへのアクセスを例示しています。

6.1 OLE DB Consumer Application Microsoft Visual C++ Examples

このサンプルではデータソースの初期化方法とC++のDBMaster OLE DBプロバイダによるデータベースへのアクセス方法を例示します。

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

#define DMOLE43
```

```
#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_DMOLE43, 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");
    }
    plColumnInfo->Release();
}
IAccessor *plAccessor = 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);

plRowset->QueryInterface(IID_IAccessor, (void**)&plAccessor);
hr = plAccessor->CreateAccessor(
    DBACCESSOR_ROWDATA,
    3,
    Bindings,
```

```

        0,
        &hAccessor,
        rgStatus);

plAccessor->Release();

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

hr = plRowset->GetNextRows(NULL,0,21,&cRowsObtained,
&phRows);
for(i=0; i<cRowsObtained; i++)
{
    hr = plRowset->GetData(rgRows[i], hAccessor, &emp);
    if(hr != S_OK)
        break;
    printf("%s\t %s\n", emp.id, emp.fname);
}

plAccessor->ReleaseAccessor(hAccessor, NULL);
plRowset->Release();
return S_OK;
}

// Manipulate a command object and execute the select command
HRESULT My_Sel_Command(IDBInitialize *plDBInitialize)
{
    IDBCreateSession* plDBCreateSession = NULL;
    IDBCreateCommand* plDBCreateCommand = NULL;
    ICommandText* plCommandText = NULL;
    WCHAR wSQLSelect[] = L"select * from employee";
    long cRowsAffected = 0;
    IAccessor* plAccessor = NULL;
    IRowset *plRowset = 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);

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

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

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

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

    return S_OK;
}

// Create accessor

```

```
HRESULT CreateParamAccessor(
    ICommand* pICmd,    // [in]
    HACCESSOR* phAccessor, // [out]
    IAccessor** pplAccessor // [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
    );

    *pplAccessor = plAccessor ;
    *phAccessor = hAccessor ;

    return (hr);
}

// Execute an insert command with parameter
HRESULT InsertWithParameters(IDBInitialize *pIDBInitialize)
{
    IDBCreateSession* pIDBCreateSession = NULL;
    IDBCreateCommand* pIDBCreateCommand = NULL;
    ICommandText* plCommandText = NULL;
    ICommandPrepare* plCommandPrepare = NULL;
    ICommandWithParameters* plCmdWithParams = NULL;
    IAccessor* plAccessor = 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 = plCommandText->Execute(NULL, IID_NULL, &Params,
    &cRowsAffected, NULL);
    printf("%ld rows inserted.\n", cRowsAffected);
}

plAccessor->ReleaseAccessor(hParamAccessor, NULL);
plAccessor->Release();
plCommandText->Release();

return S_OK;
}

// Create accessor
HRESULT myCreateParamAccessor
(
    ICommand* plCmd,    // [in]
    HACCESSOR* phAccessor,    // [out]
    IAccessor** pplAccessor    // [out]
)
{
    IAccessor* plAccessor = 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
    );

    *pplAccessor = 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 ADO Code Examples in Microsoft Visual Basic

以下のコード例ではVisual Basic使用時のDBMaster OLE DBドライバ経由でのコネクション作成を例示。

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

'establish the connection
    sCnStr = "Provider=DMOLE43; 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#

以下の例では C# 使用時の OLE DB ドライバ経由での DBMaster アクセスを例示。

注：この例示は OleDbCommand の使用時、普通タイプデータを

DBMaster データベースに挿入することを表示します。

/******

表スキーマ以下の例に使用されます :

```
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 = "AAAAABBBBBCCCCCDDDDDD";
string    c_varchar = "AAAAABBBBBCCCCCDDDDDD";
string    c_nchar = "AAAAABBBBBCCCCCDDDDDD";
string    c_nvarchar = "AAAAABBBBBCCCCCDDDDDD";
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=DMOLE43;Data
Source=DBSAMPLE4;";
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();
}
}
```