*Programming the Object Models using VBA & VBScript*

Includes CD-ROM with Enhanced Object Browser

# DAO Object Model

*The Definitive Reference*

O'REILLY®

*Helen Feddema*

# DAO Object Model

*The Definitive Reference*

Helen Feddema

O'REILLY®

*Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo*

***DAO Object Model: The Definitive Reference***
by Helen Feddema

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

***Editor:*** Ron Petrusha

***Production Editor:*** Clairemarie Fisher O'Leary

***Cover Designer:*** Edie Freedman

***Printing History:***

January 2000: First Edition.

ISBN: 1-56592-435-5 [7/00]
[M]

# Preface

Access has an object model, but that isn't where its data is kept. I can recall the first time I tried to work with Access data, from Word VBA code. I set up a reference to the Access type library, then opened the Object Browser with the Access library selected, and looked for objects representing the Access tables and fields that I could use to get at the data I wanted to import into a Word document. I couldn't find anything! There were only a few major components in the Access object model, and none of them had anything to do with data. After some research, I found that what I needed was another object model entirely—DAO (Data Access Objects). Until recently, if you wanted to work with the data stored in Access tables, you had to use DAO. Now there is another object model that can be used to get at data in Access tables—ADO (ActiveX Data Objects)—but DAO is still the primary object model used for manipulating Access data.

## What This Book Is About

This book explains all the objects in the DAO object model, down to the fields that contain the data, including their methods and properties. I will let you know where a property or method doesn't work the way Help says it should. Since you might be working with DAO from a variety of applications, I don't just cover using it in Access and Excel VBA, like the examples in Help; my examples include Access, Word, Outlook, Excel VBA, Outlook VBS (used for code behind Outlook forms), and even a few WSH (Windows Scripting Host) examples, so you can see the syntax needed to work with DAO in these dialects of VB (sometimes it is surprisingly different).

## What This Book Covers

This book covers the DAO object model that represents Access data; it applies to DAO versions 3.5 to 3.6 (Access 97 to Access 2000), since there are minimal differences between these versions. Basically, DAO 3.6 added support for Jet 4.0 and made some changes in the way non-Access data is handled.

## Who This Book Is For

If you need to work with data stored in Access tables from another Office application, in VBA or VBS, this book is for you. Even if you work with DAO exclusively within Access, you will find it handy, especially if you are working with Access 2000, because of a very unfortunate change in the way Help is handled between Access 97 and Access 2000.

## Why Not Just Use Online Help?

In Access 97, when you are working with code in a module and you open Help, there is a DAO book in the Help Contents tab. In addition, DAO objects, methods, and properties are listed in the Help index, so you can open these while working in Access VBA code, which is where you need information on these objects.

In Access 2000, however, opening Help from a code module will get you Microsoft Visual Basic Help, which has lots of well-organized information about VBA (as you would expect), but DAO is a different matter. There is no Help book for DAO in the Help index, but there is some DAO information buried in Help, along with information related to the new ADO object model, some of whose objects have the same name as objects in the DAO object model, to make it even more confusing! As is the case with HTML Help in general, if you follow the links from topic to topic, you will find yourself viewing topics belonging to many different applications, since there is no structure in HTML Help to ensure that you remain in a thread of topics appropriate for the language or object model with which you are working.

For example, if you look up "recordset" in the Help index, you will get a list of topics, some of which apply to DAO and some to ADO, and it isn't always easy to tell which object model a topic belongs to. There is no way of examining DAO objects in an organized manner because of the lack of a DAO Help book in the Contents list. HTML Help suffers from the same deficiency as many web search engines, which blindly pull up all topics containing a particular word, regardless of whether they are relevant or not.

To get at the DAO Help topics in Access 2000, you have to open Help from the main Access window and open the DAO book. If you then switch from Help back to Access, open a module, and select Help from the VBE menu, you will get an "Unable to display Help" error message, and Help will switch to the VB Help topics, with a "The page cannot be displayed" message where the Help topic should be. Since Access Help is now less helpful than ever, a DAO reference work which is available when you are working in Access code is essential. Word is, Microsoft is working on an improved interface for Access 2000 Help, but for the time being, Help is more of a hindrance than a useful tool.

## *How This Book Is Organized*

Each chapter in this book covers a DAO collection together with its singular object (or an individual object) and provides a complete reference to that object and its properties and methods. (There are no events in the DAO object model.) The treatment of each collection or object follows a standard format. The collection (or object) is described, summary tables list its properties and methods, and (where appropriate) code samples are given. In addition, the end of the section containing the general description of the collection or object includes an "Access" section that provides you with the following items of information:

- Whether the object is creatable. That is, whether a new instance of a class can be instantiated in VBA by using the `New` keyword or the *CreateObject* function, or in a scripted environment by using the VBScript *CreateObject* function or an object creation method provided by the scripted environment's object model (such as `Server.CreateObject` in ASP, for instance). To retrieve a reference to an object that is not creatable (such as a Recordset object), you must either navigate to it, access it through another object's property, or handle it as the return value of another object's method.
- The properties and methods of particular objects that return that type of collection or object.

Each object in the DAO object model is covered, with the exception of the Connection object, which is used only with ODBCDirect.

The treatment of collection and object properties also follows a standardized format, which consists of the following:

- An icon that indicates whether a property is read-only—that is, that a property's value can be retrieved but cannot be set. All other properties are either read-write (which is true of the vast majority of properties) or have an unusual behavior that is explained in the property description.

- The syntax needed to set to retrieve a property for those cases, such as collections and property arrays, where simple assignment does not work.
- The property's data type.
- A description of the property
- Where they are appropriate, tips and suggestions for using the property effectively or for avoiding some pitfall associated with the property.
- Where relevant, a code fragment that uses the property either in VBA code, in VBScript code, or in both.

Collection and object methods also are treated consistently throughout the book. Each entry for the method of a DAO collection or a DAO object includes the following:

- The method's syntax
- A description of the method's parameters
- A general description of the method
- Where they are appropriate, tips and suggestions for using the method effectively or for avoiding some pitfall associated with the method
- Where relevant, a code fragment that uses the method either in VBA code, in VBScript code, or in both

## About the CD-ROM

The CD-ROM accompanying *DAO Object Model:The Definitive Reference* includes all the sample code from the book, along with the Object Model Browser, an enhanced object browser for DAO. Unlike the Object Browser, which offers a flat, one-dimensional view of an object browser, Object Model Browser graphically depicts an object model hierarchically, so that you can easily determine how to navigate the object hierarchy or what methods and/or properties return objects of a particular type. For documentation on the Object Model Browser, see the appendixes.

For details on the organization of the CD-ROM as well as any other last-minute changes, see the *ReadMe.txt* file in the root directory of the CD-ROM.

## Conventions in This Book

Throughout this book, we've used the following typographic conventions:

`Constant width`
: Constant width in body text indicates a language construct such as a VBA statement (like `For` or `Set`), an intrinsic or user-defined constant, a user-

defined type, or an expression (like `DBEngine.Workspaces.Count - 1`). Code fragments and code examples appear exclusively in constant width text. In syntax statements and prototypes, text in constant width indicates such language elements as the function, procedure, or method's name and any other invariable elements required by the syntax.

*`Constant width italic`*
Constant width italic in body text indicates argument and variable names. In syntax statements or prototypes, it indicates replaceable parameters.

*Italic*
Italicized words in the text indicate intrinsic or user-defined functions and procedure names. Many system elements like paths and filenames are also italicized, as are new items.

This symbol indicates a tip.

This symbol indicates a warning

$\boxed{\text{RO}}$
Indicates a property that is read-only at runtime. If neither the $\boxed{\text{RO}}$ nor the $\boxed{\text{WO}}$ (write-only) icon appears beside a property name, that property is either read-write or its precise read-write status is explained in the description.

## How to Contact Us

We have tested and verified all the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
1-800-998-9938 (in the U.S. or Canada)
1-707-829-0515 (international/local)
1-707-829-0104 (fax)

You can also send messages electronically. To be put on our mailing list or to request a catalog, send email to:

> *info@oreilly.com*

To ask technical questions or comment on the book, send email to:

> *bookquestions@oreilly.com*

For our Visual Basic-related web site that includes a discussion forum and featured articles on assorted programming topics, see:

> *http://vb.oreilly.com*

## Call for Additions and Amendments

I would be the first to acknowledge that I don't know everything about every component of the DAO object model. If you know how to do something that I said can't be done or have some tips or warnings that might be useful to others, please submit them on the discussion forum at our web site, *http://forums.oreilly.com/~vb/*.

## Acknowledgments

My editor at O'Reilly, Ron Petrusha, has been of great help to me in guiding this book through a long process. Originally, the book was to cover both the Access and DAO object models, but when we saw that the number of major objects in the Access 2000 object model had jumped from 6 to 21, we realized that the book couldn't do justice to the Access object model without growing to enormous size. So we decided to split it into two books: the present book on the DAO object model and an upcoming book on the Access object model.

In addition to Ron, Katie Gardner, Tara McGoldrick, and Cheryl Smith at O'Reilly have assisted ably during the process of putting the book together. My thanks also go to my agent, Claire Horne, who was the one who first put together the book project. Thanks to tech editor Russ Darroch for many helpful suggestions, and to Matt Childs for carefully testing the code. Many people associated with *Woody's Office Watch* have been helpful in answering questions and prying information out of Microsoft, particularly Woody Leonhard and Peter Deegan. The readers of *Woody's Access Watch* (I am the editor of this e-zine) have also been helpful in suggesting workarounds for various Access and Office programming problems.

# 1 Introduction

As Windows versions have progressed, the techniques available for transferring data among Windows applications have improved, from the simple cut and paste available through the Windows 3.0 clipboard, to Dynamic Data Exchange (DDE) and Open Database Connectivity (ODBC), to the presently dominant technique, currently called Automation (it was originally called Object Linking and Embedding, then OLE Automation). With Automation code you can work directly with the components and functionality of applications that support Automation, using their object models.

### OLE Servers and Clients

Microsoft used to make a distinction between applications that were OLE servers, which exposed their objects for manipulation by code running from other applications, and OLE clients, which hosted a VB dialect with functions and methods used to manipulate objects in OLE server applications' object models.

In previous versions of Office, some applications were OLE servers only, some were OLE clients only, and some were both OLE clients and servers. With Office 2000, however, all the major Office applications (Access, Word, Excel, Outlook, and PowerPoint) support Automation both as clients and servers, so Microsoft has stopped making this distinction and simply notes that an application supports Automation.

An object model represents the components of an application (or a subset of its components) as a set of objects (usually arranged hierarchically), each of which has properties, methods, and events (though not necessarily all three) that developers can reference in code. Theoretically, any language can access the DAO object model (for example, JScript or Perl running in IIS/ASP, or C/C++). But in real life, Visual Basic for Applications (VBA) or VBScript (VBS) are the most common languages used to work with object models, probably by several orders of magnitude. An *object* generally represents something you work with in the application's interface—for example, Access tables, forms, and reports; Word documents, tables, and words; Excel worksheets, charts, and ranges; and so forth. Additionally, some of the objects in an object model may be collections of other objects, such as the Reports collection in Access (which is a collection of all open Report objects) or the Worksheets collection in Excel (which is a collection of all the Worksheet objects in a particular workbook).

When you write code in a dialect of Visual Basic to work with other applications that support Automation, you need to understand the server application's object model, so that you will be able to reference the appropriate application components as represented in its object model, and use their methods and properties to achieve the desired results. While the names of objects in an object model may be familiar to you from working with the corresponding objects in the interface (for example, Access tables, forms, and reports), in other cases the object names may not be familiar from working in the application's interface (such as the Access Screen and DoCmd objects, the DAO Container objects, the Outlook NameSpace, Explorer and Inspector objects, or the Word Range object.)

An application's object model may not represent all of the application's functionality. For example, the Outlook object model omits Views, and the Access object model omits Import/Export specifications. The DAO object model avoids such discrepancies, as it has no interface.

## *Early and Late Binding*

Visual Basic for Applications allows you to access an object model using either *early binding* or *late binding*. In contrast, VBScript, because its code is interpreted rather than compiled and because it supports only the Variant data type and does not allow strong typing, supports only late binding.

Late binding means that references to objects in the object model are resolved at runtime; this is because those references cannot be resolved at design time either because precise object types are unknown or because the language does not support early resolution of object references. Typically, in VBA, which supports explicit data typing, this means that object variables are declared to be of the generic Object data type, rather than of more specific object types (like a Database object, a TableDef object, or a Workspace object, for instance).

The VBA code shown in Example 1-1 uses late binding to display the number of records in the Customers table of the Northwind database. The equivalent VBScript code (which is nearly identical) is shown in Example 1-2. Note the use of a named constant for the recordset type argument in the VBA code, while the VBS code uses its numeric equivalent.

The code in Example 1-1 can use a String variable because String is not a component of an object model. This is possible in VBA code, but not in VBS code, which doesn't permit data typing of any variables.

*Example 1-1. VBA Code Using Late Binding*

```
Private Sub cmdRecordCountLB_Click()

   Dim objDBEng As Object
   Dim objRS As Object
   Dim objDB As Object
   Dim strDBName As String

   strDBName = "D:\Documents\Northwind.mdb"
   'Use DBEngine.35 for Access 97, DBEngine.36 for Access 2000.
   Set objDBEng = CreateObject("DAO.DBEngine.35")
   Set objDB = objDBEng.OpenDatabase(strDBName)
   Set objRS = objDB.OpenRecordset("Customers", dbOpenTable)

   Debug.Print objRS.RecordCount & " records in Customers table"

   objRS.Close
   objDB.Close

End Sub
```

*Example 1-2. VBScript Code Using Late Binding*

```
Sub cmdRecordCount_Click()

   Dim dbe
   Dim wks
   Dim dbs
   Dim rst
   Dim strDBName
   strDBName = "D:\Documents\Northwind.mdb"

   Set dbe = Application.CreateObject("DAO.DBEngine.35")
   Set wks = dbe.Workspaces(0)
```

*Example 1-2. VBScript Code Using Late Binding (continued)*

```
   Set dbs = wks.OpenDatabase(strDBName)
   Set rst = dbs.OpenRecordset("Customers", 1)
   Msgbox rst.RecordCount & " records in Customers"
   rst.Close
   dbs.Close

End Sub
```

*Early binding*, on the other hand, involves accessing an automation object's type library (a library that contains information about an object model) to resolve references at design time rather than at runtime. This in turn requires that the development environment produce compiled code; hence, scripting languages such as VBScript cannot support early binding. Taking advantage of early binding requires exact typing of object variables; variables can't be declared as generic objects of the Object data type, but must instead be declared as specific objects defined by the object model.

The VBA code in Example 1-3 is the equivalent of that in Example 1-1, except that it uses early binding. Note that, rather than declaring generic object variables of type Object, the code declares specific object types (the DBEngine, Recordset, and Database objects) defined by the DAO object model. Also note that the code takes advantage of a symbolic constant, `dbOpenTable`, that is defined in the type library.

*Example 1-3. VBA Code Using Early Binding*

```
Private Sub cmdRecordCountEB_Click()

   Dim dbe As DAO.DBEngine
   Dim dbs As DAO.Database
   Dim rst As DAO.Recordset
   Dim strDBName As String

   strDBName = "D:\Documents\Northwind.mdb"
   'Use DBEngine.35 for Access 97, DBEngine.36 for Access 2000.
   Set dbe = CreateObject("DAO.DBEngine.35")
   Set dbs = dbe.OpenDatabase(strDBName)
   Set rst = dbs.OpenRecordset("Customers", dbOpenTable)

   Debug.Print rst.RecordCount & " records in Customers table"

   rst.Close
   dbs.Close

End Sub
```

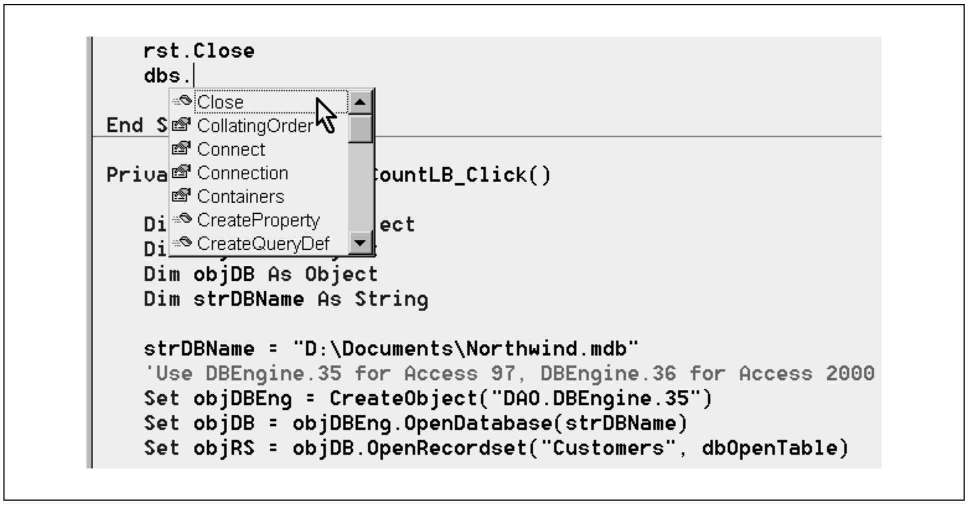
Late binding is vastly inferior to early binding in two major respects:

*Performance*

Because all object references must be resolved at runtime rather than at design time, in most cases late binding involves an enormous performance penalty. It is not uncommon for a particular operation in late-bound code to consume anywhere from 25% to 1000% more time than the same operation in early-bound code.

*Assistance when coding*

Both the Visual Basic and VBA development environments (though not the VBS Script Editor) offer a feature called Auto List Members that allows you to select an item that completes an expression from a popup list box. For example, Figure 1-1 shows the list box that appears when you type `dbs` (the name of a Database object variable) followed by a period, indicating that you want to use a method or property of a Database object. Support for auto list members is not available for late-bound objects.

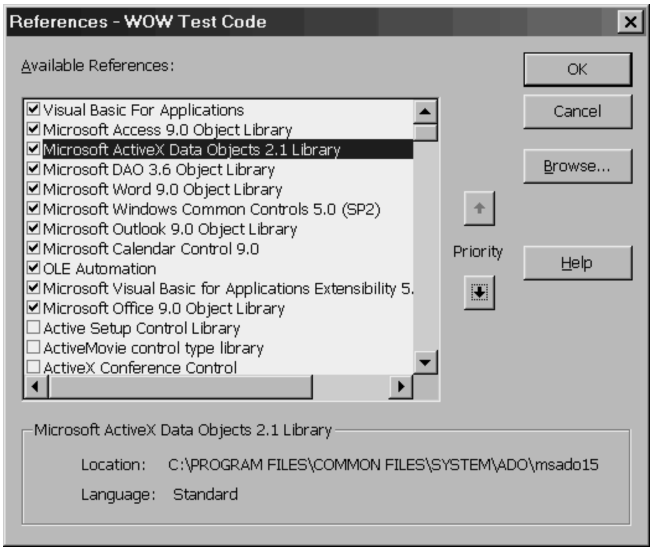


```
    rst.Close
    dbs.
End S
Priva                       ountLB_Click()

    Di                ect
    Di
    Dim objDB As Object
    Dim strDBName As String

    strDBName = "D:\Documents\Northwind.mdb"
    'Use DBEngine.35 for Access 97, DBEngine.36 for Access 2000
    Set objDBEng = CreateObject("DAO.DBEngine.35")
    Set objDB = objDBEng.OpenDatabase(strDBName)
    Set objRS = objDB.OpenRecordset("Customers", dbOpenTable)
```

*Figure 1-1. Auto list members for a database variable*

In other words, unless there's a compelling reason to do otherwise, it's best to take advantage of early-bound object references.

Typically, early binding to a particular object model is made available to VB or VBA by adding a reference to the object library to a project. In VB this is done by selecting the References option from the Project menu; in hosted VBA environments, by selecting the References option from the Tools menu. The result is the References dialog, shown in Figure 1-2, which allows you to select object model references from a list box.



☑ Visual Basic For Applications
☑ Microsoft Access 9.0 Object Library
☑ Microsoft ActiveX Data Objects 2.1 Library
☑ Microsoft DAO 3.6 Object Library
☑ Microsoft Word 9.0 Object Library
☑ Microsoft Windows Common Controls 5.0 (SP2)
☑ Microsoft Outlook 9.0 Object Library
☑ Microsoft Calendar Control 9.0
☑ OLE Automation
☑ Microsoft Visual Basic for Applications Extensibility 5.
☑ Microsoft Office 9.0 Object Library
☐ Active Setup Control Library
☐ ActiveMovie control type library
☐ ActiveX Conference Control

Microsoft ActiveX Data Objects 2.1 Library

Location: C:\PROGRAM FILES\COMMON FILES\SYSTEM\ADO\msado15

Language: Standard

*Figure 1-2. The References dialog with references set to the ADO and DAO object libraries*

## *The Object Models for Working with Access*

While most applications have only one object model, Access has three: one for the interface, one for data stored in Access databases, and one for data stored in either Access databases or external data sources. (One of them—ADO—is a recent addition to Microsoft's stable of data access technologies and new to Access 2000.)

### *The Access Object Model*

The Access object model (shown in Figure 1-3) represents the Access interface elements (forms, reports, and modules) and a good deal of the application's functionality via the Screen and DoCmd objects. Although there is some overlap of functionality between the Access and DAO object models (via the DBEngine object, located under the Application object), the Access object model is primarily used to work with Access objects, such as forms and reports.

### *The DAO Object Model*

The DAO (Data Access Objects) object model represents the data stored in Access tables. You need to work with the DAO object model both within Access VBA and when working with data stored in Access databases from other applications, using

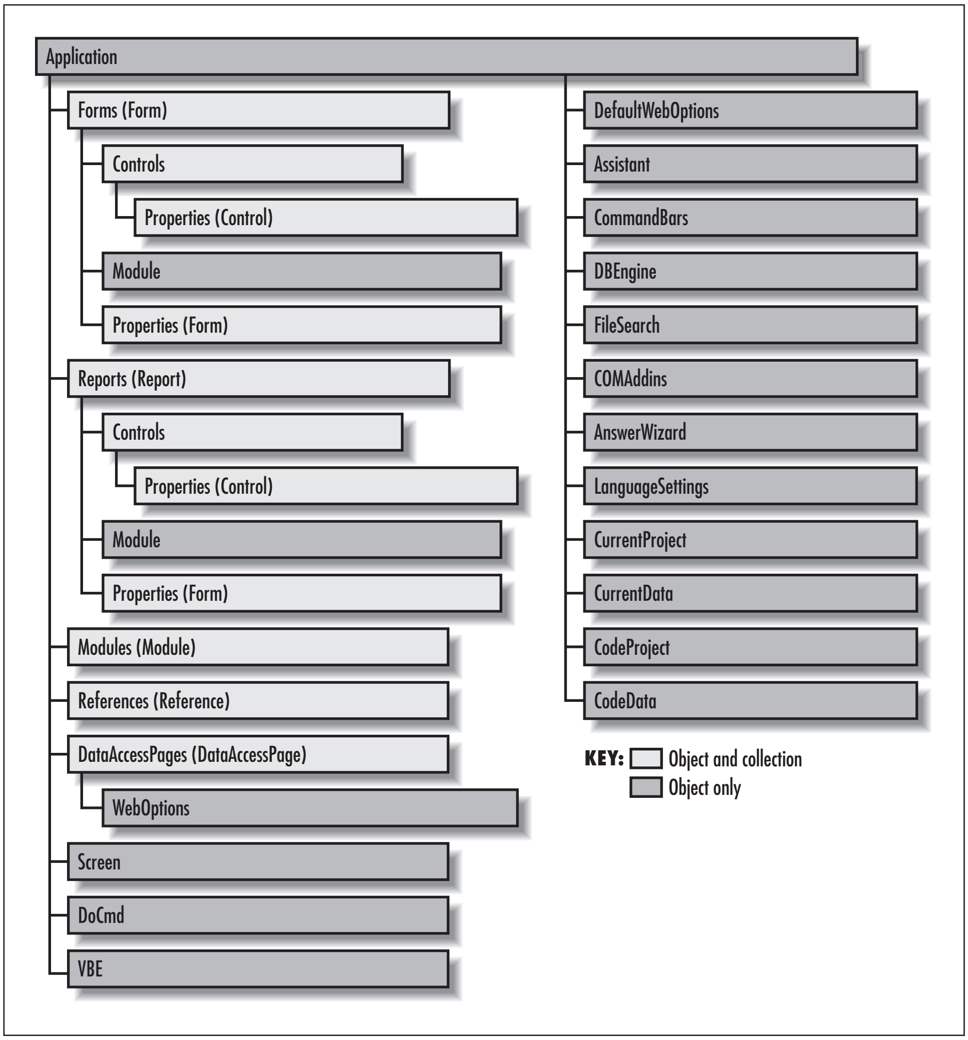


*Figure 1-3. The Access 2000 object model*

VBA and VBS. The DAO object model is available to anyone who has the Jet engine library installed (this library is fairly easy to redistribute), while in order to use the Access object model, the user must have Access installed on his machine. There are two versions of the DAO object model: one for Jet workspaces (shown in Figure 1-4) and one for ODBCDirect workspaces (shown in Figure 1-5).

When you write code to work with Access, you need the DAO object model to retrieve data from tables, append records to tables, or update data in tables, and you need the Access object model to display forms, print reports, or run macros. Often you will need to use both the Access and DAO object models in the same

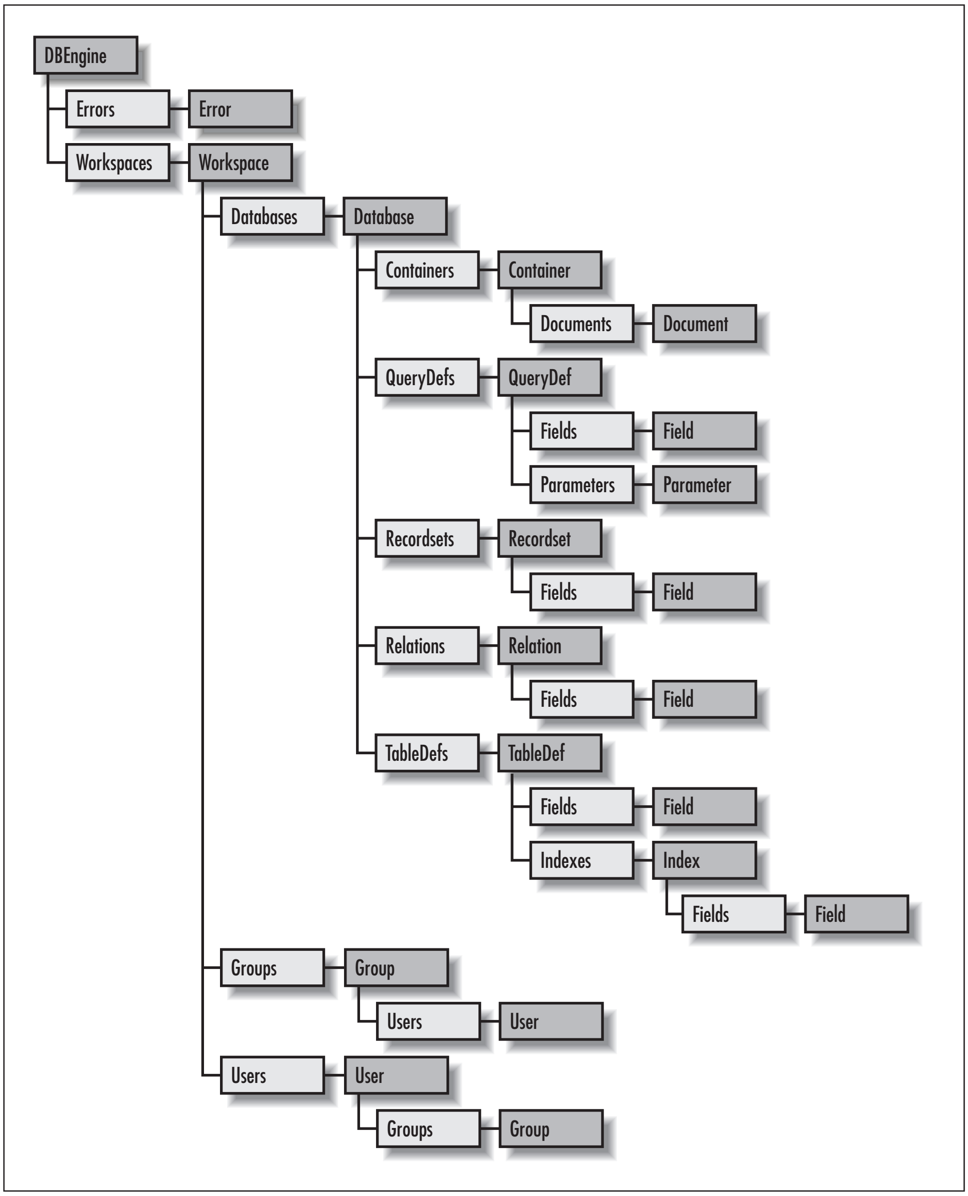


*Figure 1-4. The DAO object model for Jet workspaces*

procedure. The DAO object model (primarily the more extensive Jet version) is covered in this book; an upcoming book will cover the Access 2000 object model.

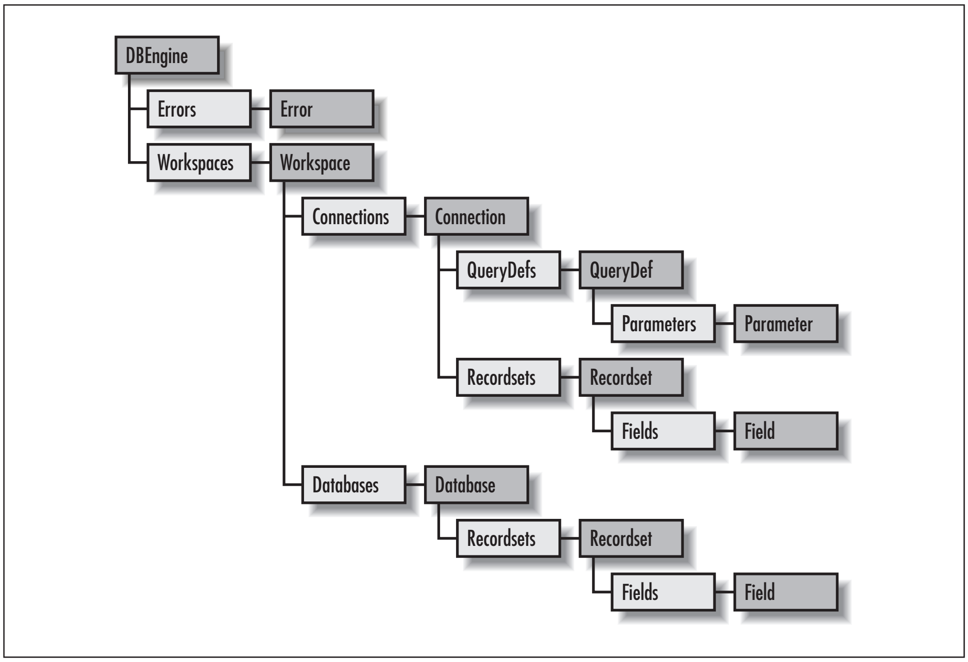


*Figure 1-5. The DAO object model for ODBCDirect workspaces*

## *ActiveX Data Objects*

The ADO (ActiveX Data Objects, with a silent "X") object model (shown in Figure 1-6) can represent data in Access tables, but it is also used to work with data in external (non-Access) data sources. As a rule of thumb, if you are working exclusively with Access data, you can stick with the DAO object model; if you need to work with data in non-Access data sources, you need the ADO object model. However, ADO is a new (think version 1.0) technology, which many developers consider to be lacking in robustness; many are waiting until the next version to rely on ADO for real-world applications.

Each of the Connection, Command, Recordset, and Field objects also has its own Properties collection, as shown in Figure 1-7.

See the forthcoming *ActiveX Data Objects: The Definitive Guide*, written by Jason T. Roff and published by O'Reilly & Associates, for more details on the ADO object model.

However, despite concerns about ADO's reliability, there are some circumstances in which ADO code is more efficient for working with Access data, such as using

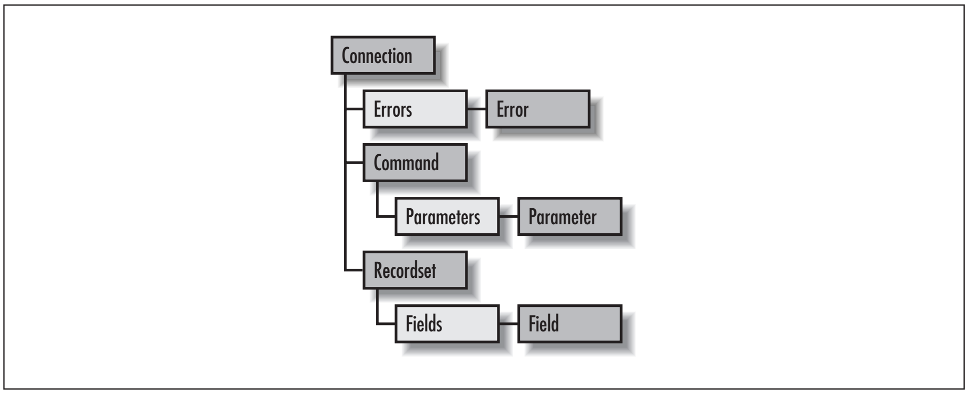


*Figure 1-6. The ADO object model*

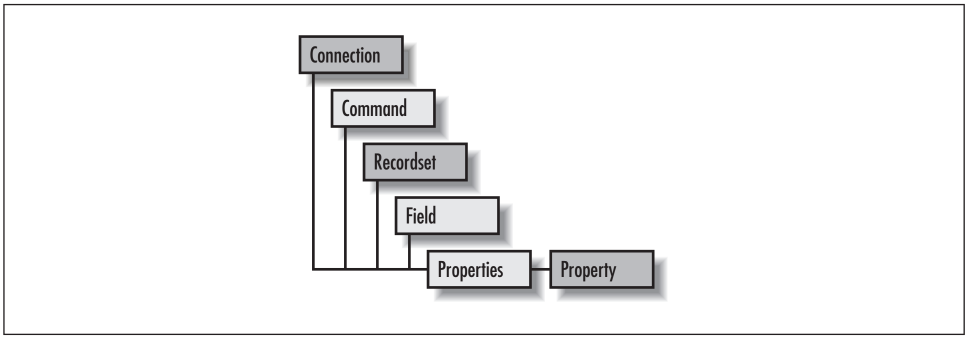


*Figure 1-7. The Properties collection in the ADO object model*

an ADO recordset to fill the drop-down list of a combo box in place of a cumbersome fill function, or saving a recordset to a file on your computer, which is possible with ADO recordsets but not with DAO recordsets.

If you need to use both the DAO and ADO objects models in your code, bear in mind that they have some identically named objects, recordsets in particular. Even though an object name may be the same in both object models, its functionality may differ in ADO and DAO, which can result in strange bugs in your code. If you have references set to both the DAO and ADO object models (see Figure 1-2) and you don't use prefixes in your code to indicate which object model your database variables belong to, when the code is compiled, variables for objects such as recordsets will be assumed to belong to whichever object model is highest in the list of available object libraries, possibly with unfortunate results. By default, new Access 2000 databases have a reference set to the ADO object library, but not to the DAO object library, while converted databases will usually have a reference set to the DAO object library.

If you get compile errors after converting an Access 97 database to Access 2000, this may be because the ADO reference is located above the DAO reference in the References list. If you don't anticipate using ADO in the database, just uncheck its reference; otherwise, move the DAO reference above the ADO reference in the list, so any ambiguous objects will be interpreted as belonging to the DAO object model.

To prevent confusion (particularly likely with recordsets, which are the most commonly used objects in both object models), use the prefix DAO when declaring a DAO database or recordset and the prefix ADODB when declaring a recordset, as in the following code samples (and Example 1-3):

```
Dim dbs as DAO.Database
Dim rst as DAO.Recordset
Dim rst as ADODB.Recordset
```

(You have to work down to recordsets from databases in DAO, while in ADO you can create a recordset variable directly.)

If you only need to work with a single object model (DAO or ADO) in your code, you can eliminate the need to use prefixes by checking only the appropriate library in the References dialog so as to remove ambiguity in your code. However, it may be prudent to use prefixes even in that case, as you may need to use the other object model at a later time. If you use prefixes right from the start, you won't have to rewrite your code later on when you add the second reference.

## *The Visual Basic Dialects*

You can work with object models in any programming language that supports Automation. For working with Office applications using their object models, generally you will use a dialect of Visual Basic—either Visual Basic (VB) itself, Visual Basic for Applications (VBA), or Visual Basic Scripting Edition (VBS). VBA is well documented in the Help files provided with Office, but VBS is another matter. The documentation for the dialect of VBS used in Outlook is quite inadequate: covering only properties, methods, and events, and omitting functions, operators, keywords, and other language components. Microsoft provides online documentation for VBS at *http://msdn.microsoft.com/scripting/default.htm* (and you can download the documentation to your computer for offline access).

To work with object models representing other applications, you can use the *CreateObject* and *GetObject* functions (or, in some dialects, methods). In some cases, the `New` keyword can be used when declaring an object using early binding, thus avoiding the need to use *CreateObject* or *GetObject*:

```
Dim appWord As New Word.Application
```

However, different dialects of VB and different Office applications are inconsistent in their support of these features, so you may need to experiment with the `New` keyword, *CreateObject*, and *GetObject* in a particular procedure to see which is most reliable. Depending on the dialect of VB you are using and the object model you are manipulating, you may get different results. In Outlook VBS, for example, *GetObject* doesn't work at all, while when working with the Outlook object model from another application, there appears to be no difference between using the `New` keyword and the *CreateObject* method to create an instance of Outlook.

In my experience, sometimes *GetObject* works with Excel worksheets, and sometimes it doesn't. When you use the `New` keyword to create an instance of Word 2000, you can create multiple Word instances by repeated use of the keyword (with different application variables). When you try the same thing with Outlook 2000, however, you won't get any extra instances of Outlook. These differences depend on whether the target application is an SDI (Single Document Interface) application (like Word 2000) or an MDI (Multiple Document Interface) application (like Excel 2000) and perhaps other factors as well.

While working with various dialects of VBA and VBS on several Office applications, I have found the *CreateObject* method to be the most reliable way of creating an instance of an application; it works with all dialects and all target applications. The `New` keyword works in many cases, and the *GetObject* method works in some cases. Regardless, once you have created an instance of an application, you can always work down through its object model to the particular component you need to work with. So *GetObject* is not needed, though if it works in a particular case, it can be a handy shortcut.

There is a difference between Access and other hosted apps (as well as VB). Since the database engine is always active in Access, you don't have to instantiate a new DBEngine object; with VB and hosted VBA in Word, etc., you do.

VBA and VBS have been upgraded a number of times since their introduction; the VBA versions are listed in Table 1-1, and VBS versions are listed in Table 1-2.

*Table 1-1. VBA Versions*

| Product | VBA Version | Year |
|---|---|---|
| Excel 5.0 | 1.0 | 1993 |
| Project 4.0 | 1.0 | 1994 |
| Excel 95 | 1.0 | 1995 |
| Visual Basic 4.0 | 2.0 | 1994 |
| Access 95 | 2.0 | 1995 |
| Office 97 | 5.0 | 1997 |
| Office 2000 | 6.0 | 1999 |

*Table 1-2. VBS Versions*

| Product | VBS Version | Year |
|---|---|---|
| Internet Explorer 3.0 | 1.0 | 1996 |
| Internet Information Server 3.0 | 2.0 | 1997 |
| Internet Explorer 4.0 | 3.0 | 1998 |
| Windows Scripting Host 1.0 | 3.0 | 1998 |
| Outlook 98 | 3.0 | 1998 |
| Visual Studio 6.0 | 4.0 | 1998 |
| Internet Explorer 5.0 | 5.0 | early 1999 |
| Internet Information Server 5.0 | 5.0 | early 1999 |
| Preliminary releases of Windows 2000 | 5.1 | late 1999 |

The main differences between VB, VBA, and VBS are listed in Table 1-3.

*Table 1-3. The Visual Basic Dialects*

| VB | VBA | VBS |
|---|---|---|
| Allows you to create a standalone executable | Doesn't support creating a standalone executable; runs from a module or macro | Doesn't support creating a standalone executable; runs from code attached to a form (Outlook), a web page (IE), or a standalone script (WSH) |
| Must be purchased separately | Included with most Office and other Microsoft applications (substantially the same dialect) and many third-party applications (e.g., AutoCAD) | Included with Internet Explorer, Outlook, and Windows 98 (significantly different dialects) |
| Has a rich developer's environment | Has a rich developer's environment | Has a limited developer's environment |
| Has powerful debugging tools | Has powerful debugging tools | Has limited or nonexistent debugging tools |
| Has a full-featured Object Browser | Has a full-featured Object Browser | Has a limited Object Browser |

*Table 1-3. The Visual Basic Dialects (continued)*

| VB | VBA | VBS |
|---|---|---|
| Supports data typing of variables | Supports data typing of variables | Does not support data typing of variables (all variables are Variants) |
| Supports named constants and arguments | Supports named constants and arguments | Offers very limited support for named constants (developers must use numeric values or declare their own constants) and no support for named arguments |
| Produces compiled code stored in the "document" | Produces compiled code stored in the "document" | Runs from script that is interpreted by Outlook, IE, IIE, or WSH |

Given a choice, VBA is generally preferable since it is provided with most Microsoft Office applications. With Office 2000, Outlook finally hosts VBA as well as VBS. In Office 97, Word, Access, Excel, and PowerPoint host VBA, while Outlook only hosts VBS. Internet Explorer 4.x and 5.x host VBS, and Windows 98 includes VBS as well in the form of the Windows Script Host. Additionally, some non-Office Microsoft applications (such as Project) host VBA, as do some non-Microsoft applications, such as Visio (which was recently purchased by Microsoft). VBA also provides the developer with a sophisticated work environment (VBE), unlike VBS with its Notepad-like Script Editor.

If you need to run a procedure directly from an Outlook form, from an IE web page, from an Active Server page or from the Windows command line, you need to write the code in VBS; if the code can be run from Access, Word, Excel, Outlook 2000 (for application-wide code), or PowerPoint, you can use the more powerful VBA dialect. On the other hand, if you need to create a standalone application, you must use VB.

The samples in Example 1-4 and Example 1-5 perform the same actions (running a make-table query and then listing the TableDefs in the Northwind database) in Access VBA and Outlook VBS code.

*Example 1-4. Access VBA Code*

```
Private Sub cmdExecute_Click()

   Dim wks As Workspace
   Dim dbs As Database
   Dim qdf As QueryDef
   Dim strSQL As String
   Dim tdf As TableDef

   Set wks = Workspaces(0)
   Set dbs = wks.OpenDatabase("D:\Documents\Northwind.mdb")
```

*Example 1-4. Access VBA Code (continued)*

```
   strSQL = _
     "SELECT Orders.*, * INTO tmakNewOrders FROM Orders WHERE OrderDate>#1/6/96#;"
   Set qdf = dbs.CreateQueryDef("qmakTestQuery", strSQL)

   'Execute a make-table query to produce the tmakRecentOrders table.
   qdf.Execute

   For Each tdf In dbs.TableDefs
      Debug.Print "Table Name: " & tdf.Name & vbTab & vbTab & _
         "Attributes: " & tdf.Attributes
   Next tdf
   dbs.Close

End Sub
```

*Example 1-5. VBScript Code*

```
Sub cmdExecuteQDF_Click()

   Dim dao
   Dim wks
   Dim dbs
   Dim strSQL
   Dim qdf

   Set dao = Application.CreateObject("DAO.DBEngine.35")
   Set wks = dao.Workspaces(0)
   Set dbs = wks.OpenDatabase("D:\Documents\Northwind.mdb")
   Set qdf = dbs.QueryDefs("qmakRecentOrders")

   'Execute a SQL statement to produce the tmakRecentOrders table.
   qdf.Execute

   For Each tdf In dbs.TableDefs
      MsgBox "Table Name: " & tdf.Name
   Next
   dbs.Close

End Sub
```

Since VBA has a richer developer's environment than Outlook VBS, it is often easier to write and debug a procedure in VBA, then convert it to Outlook VBS.

The following syntactical differences between the VBA and VBS code are typical of the changes you need to make when converting code from VBA to VBS. This would be necessary, for example, if you need to retrieve Access data using the DAO object model for display on an Outlook form:

- There are no data types in VBS, so you can't declare variables as any specific data type.
- In VBA, the bang (!) operator is used to indicate a member of a collection, such as a field in a recordset, while the dot (.) operator is used to indicate methods or properties. In VBS, you must use the dot operator both for members of collections and for properties and methods.
- In VBA, you can use named constants (such as `dbHiddenObject`) as function arguments or as free-standing expression components (such as `vbTab`). In VBS (with a very few exceptions, such as `True` and `False`), you must use the argument's numeric value or define the constant yourself using the `Const` keyword.
- VBS lacks the Debug window, so you need to replace `Debug.Print` calls with *MsgBox* functions.
- The `With...End With` statement is not supported by VBS, so you have to use the full syntax every time you reference a variable.
- When using a looping code structure (such as `For Each...Next`), you can't specify the looping variable—just use `Next` instead of `Next tdf`.
- When you call an object model method from VBA, you have a choice between using positional arguments or named arguments. For example, some developers prefer to write code like the following:

  ```
  Set rst = qdf.OpenRecordset(dbOpenForwardOnly)
  ```

  in which the meaning of an argument is determined by its function call. Others find it easier to write code like the following:

  ```
  Set rst = qdf.OpenRecordset(Type:=dbOpenForwardOnly)
  ```

  in which arguments are identified by their names. In VBScript, you don't have this choice, since VBScript supports only positional arguments.

The following chapters will contain code samples for Access VBA and (where appropriate) one of the other Office VBA dialects, Outlook VBS or WSH (Windows Script Host) VBS, as well, so you can see the syntactical differences in usage between VBA and VBS code.

# 2
# *DBEngine Object*

The DBEngine object is the highest-level object in the Jet/DAO object model, representing the entire hierarchy of data objects you can manipulate from code. There is only one DBEngine object, and you can't create additional ones. This object corresponds to the Application object that is at the top of most of the Microsoft Office object hierarchies.

DAO has two flavors: Jet and ODBCDirect. The Jet version of DAO lets you access data in Jet databases (basically, Access databases or *.mdb* databases used by other Microsoft applications), Jet-connected ODBC databases, and installable ISAM data sources such as Paradox or Lotus (although ISAM data sources have become much less important in Office 2000). The object model for the Jet version of DAO is shown in Figure 2-1.

The ODBCDirect version of DAO is used to access data sources through ODBC without use of the Jet engine. Its object model is shown in Figure 2-2. This object model lacks some of the components that are needed for working with data in Jet databases. This chapter will cover the more extensive Jet version of the DAO object model.

The DBEngine object contains two collections, Errors and Workspaces, which will be discussed in the next two chapters. The VBA code behind an Access form, shown in Example 2-1, lists the DBEngine object's properties if Access has been started by the user, rather than through automation. The resulting message box is shown in Figure 2-3.

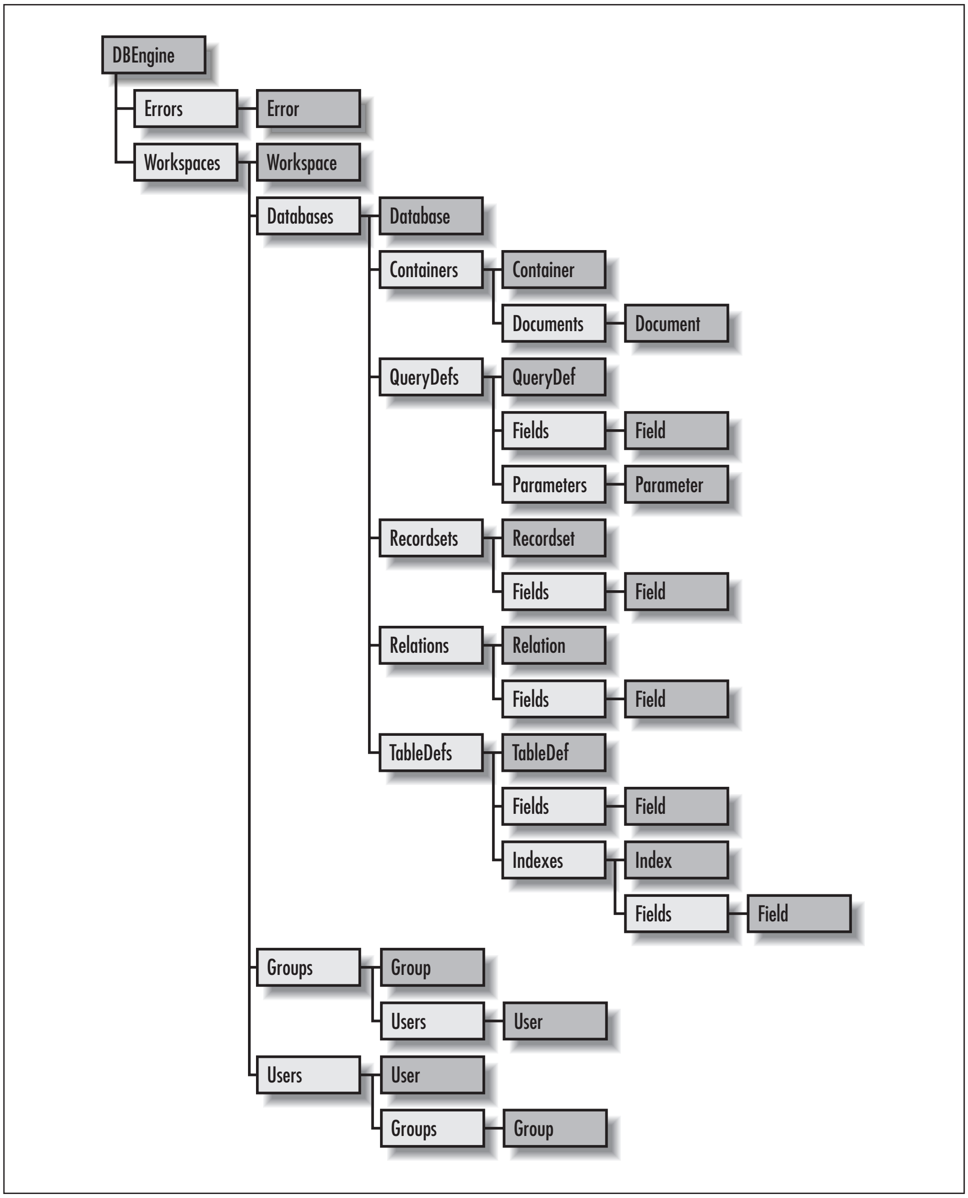


*Figure 2-1. The DAO object model for Jet workspaces*

*Example 2-1. Access Code to Display DBEngine Properties*

```
Private Sub cmdDBEngine_Click()

   Call AppProperties(Me)

End Sub
```

*Example 2-1. Access Code to Display DBEngine Properties (continued)*

```
Private Function AppProperties(obj As Object) As Integer

   Dim objAccess As Access.Application
   Dim i As Integer
   Dim strProperties As String

   On Error Resume Next
   Set objAccess = obj.Application

   If objAccess.UserControl = True Then
      For i = 0 To objAccess.DBEngine.Properties.Count - 1
         strProperties = strProperties & _
            objAccess.DBEngine.Properties(i).Name & vbCrLf
      Next i
   End If

   MsgBox left(strProperties, Len(strProperties) - 2), _
      vbOKCancel, "DBEngine properties"

End Function
```

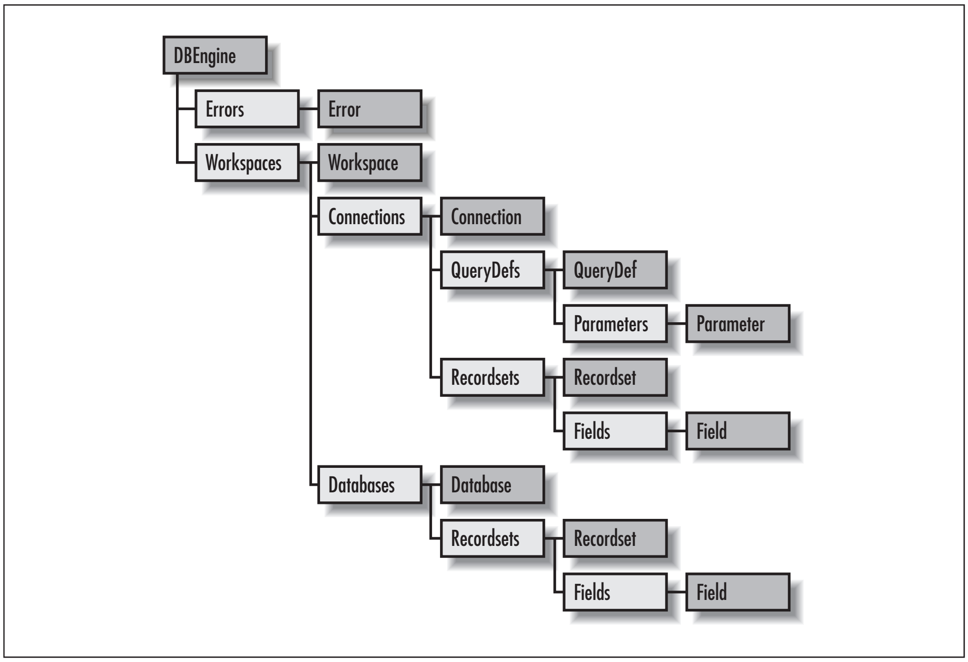
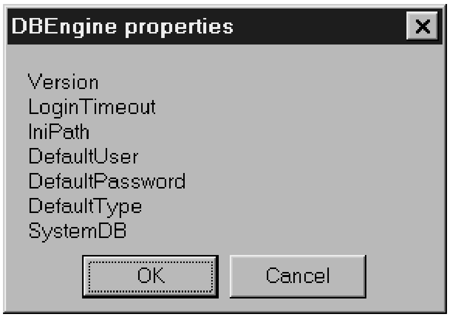


*Figure 2-2. The DAO object model for ODBC workspaces*


DBEngine properties

Version
LoginTimeout
IniPath
DefaultUser
DefaultPassword
DefaultType
SystemDB

*Figure 2-3. DBEngine object properties when Access is started by the user*

Although the example in Access Help implies that you will get a different set of properties when Access is started from Automation code rather than by the user, the code in Example 2-2, an Excel VBA function that opens Access using automation code, produces the same listing of properties.

*Example 2-2. Excel VBA Code to Display DBEngine Properties*

```
Function ListDBEngineProps()

   Dim objAccess As New Access.Application
   Dim i As Integer
   Dim strProperties As String

   If objAccess.UserControl = False Then
      For i = 0 To objAccess.DBEngine.Properties.Count - 1
         strProperties = strProperties & _
            objAccess.DBEngine.Properties(i).Name & vbCrLf
      Next i
   End If

   MsgBox Left(strProperties, Len(strProperties) - 2), _
      vbOKCancel, "DBEngine properties"

End Function
```

Since it is more than likely that there will be several different versions of DAO on any computer (installed by various versions of Microsoft applications), in order to avoid an "ActiveX component can't create object" error, you should append the version number after the DBEngine object reference in your call to the *CreateObject* function, as shown in the following Word VBA and Outlook VBScript code samples. For Access 97, the DAO version ranges from 3.0 (referenced in code as 30) to 3.51 (referenced as 35) depending on whether you have the original release, SR-1, or SR-2; for Access 2000, it is 3.6 (referenced as 36).

The first parameter passed to the VBA *CreateObject* function is a programmatic identifier. When it includes only an object reference (as in `DAO.DBEngine`), it is a version-independent programmatic identifier. When it includes a version in the object reference (as in `DAO.DBEngine.36`), it is a version-dependent programmatic identifier. Typically, version-independent programmatic identifiers in the registry are updated to include information about the most recent installed version of that object. DAO, however, does not appear to do this, making it more important that a version-dependent programmatic identifier be used when calling the *CreateObject* function.

In order to use the DAO object model from other applications, such as Word or Outlook, you need to first define a reference to the DBEngine object. Example 2-3 and Example 2-4, both of which use the *CreateObject* function to create an instance of the DBEngine object, show how to do this using VBA for Word and VBScript for Outlook, respectively. The Word VBA code in Example 2-3 imports three fields from the sample Northwind Products table into a Word table that's also created from code (see Table 2-1 for a portion of the table of imported data). The Outlook VBScript code in Example 2-4 imports three fields of data from the Northwind Customers table into a list box on an Outlook form. The Word VBA code in Example 2-3 uses early binding (after setting references to the Access and DAO object libraries) and declares its variables as specific data types; the Outlook VBScript code in Example 2-4 uses late binding and no data types because VBScript does not support early binding or data typing.

*Table 2-1. Data Imported into a Word Table from the Northwind Products Table*

| ID | Product Name | Units in Stock |
|---|---|---|
| 1 | Chai | 39 |
| 2 | Chang | 17 |
| 3 | Aniseed Syrup | 13 |
| 4 | Chef Anton's Cajun Seasoning | 53 |
| 5 | Chef Anton's Gumbo Mix | 0 |
| 6 | Grandma's Boysenberry Spread | 120 |
| 7 | Uncle Bob's Organic Dried Pears | 15 |
| 8 | Northwoods Cranberry Sauce | 6 |
| 9 | Mishi Kobe Niku | 29 |
| 10 | Ikura | 31 |

*Example 2-3. Word VBA Code to Access the DBEngine Object*

```
Sub FillTableFromAccess()

    Dim dao As DAO.DBEngine
    Dim wks As Workspace
    Dim dbs As Database
    Dim rst As Recordset
    Dim strAccessDir As String
    Dim strDBName As String
    Dim objAccess As New Access.Application

    'Get path to Access database directory from Access SysCmd function.
    Set objAccess = CreateObject("Access.Application")
    strAccessDir = objAccess.SysCmd(9)
    strDBName = strAccessDir & "Samples\Northwind.mdb"
    Debug.Print "DBName: " & strDBName
    objAccess.Quit

    'Set up reference to Access database.
    Set dao = CreateObject("DAO.DBEngine.35")
    Set wks = dao.Workspaces(0)
    Set dbs = wks.OpenDatabase(strDBName)

    'Set reference to Products table.
    Set rst = dbs.OpenRecordset("Products")

    'Create 3-column Word table to fill with Access data.
    ActiveDocument.Tables.Add Range:=Selection.Range, _
        NumRows:=2, NumColumns:=3
    With Selection
        .TypeText Text:="ID"
        .MoveRight Unit:=wdCell
        .TypeText Text:="Product Name"
        .MoveRight Unit:=wdCell
        .TypeText Text:="Units in Stock"
        .MoveRight Unit:=wdCell
    End With

    'Write info from a record in Products to a row of the Word table;
    'loop through recordset until all data has been written to the table.
    Do Until rst.EOF
       With Selection
           .TypeText Text:=rst![ProductID]
           .MoveRight Unit:=wdCell
           .TypeText Text:=rst![ProductName]
           .MoveRight Unit:=wdCell
           .TypeText Text:=rst![UnitsInStock]
           .MoveRight Unit:=wdCell
       End With
        rst.MoveNext
    Loop

End Sub
```

*Example 2-4. VBS Outlook Code to Import Data into a Listbox on an Outlook Form*

```
Function FillListBox()

    Dim rst
    Dim dao
    Dim wks
    Dim dbs
    Dim ctl
    Dim strAccessDir
    Dim objAccess
    Dim CustomerArray(99, 2)

    'Get path to Access database directory from Access SysCmd function.
    Set objAccess = Item.Application.CreateObject("Access.Application")
    strAccessDir = objAccess.SysCmd(9)
    strDBName = strAccessDir & "Samples\Northwind.mdb"
    'MsgBox "DBName: " & strDBName
    objAccess.Quit

    'Set up reference to Access database.
    Set dao = Application.CreateObject("DAO.DBEngine.35")
    Set wks = dao.Workspaces(0)
    Set dbs = wks.OpenDatabase(strDBName)

    'Retrieve Customer info from table.
    Set rst = dbs.OpenRecordset("Customers")
    Set ctl = Item.GetInspector.ModifiedFormPages( _
              "Filling Combo & List Boxes").Controls("lstCustomers")

    ctl.ColumnCount = 3
    ctl.ColumnWidths = "50; 150 pt; 75 pt"

    'Assign Access data to an array of 3 columns and 100 rows.
    CategoryArray(99, 2) = rst.GetRows(100)

    'Display array data in list box.
    ctl.Column() = CategoryArray(99, 2)

End Function
```

In addition to the 2 collections we've briefly discussed (the Errors and Workspaces collections), the DBEngine object supports 8 properties (shown in Table 2-2) and 12 methods (shown in Table 2-3).

*Table 2-2. DBEngine Properties*

| Property | Description |
|---|---|
| DefaultPassword | Defines the type of the next Workspace object to be created |
| DefaultUser | Defines the user name used to create the default workspace whenever it is initialized |
| DefaultType | Defines the password used to create the default workspace whenever it is initialized |

*Table 2-2. DBEngine Properties (continued)*

| Property | Description |
|---|---|
| IniPath | Indicates the registry key containing information about Jet engine settings |
| LoginTimeout | Determines the number of seconds to wait before an attempt to log onto an ODBC database is considered unsuccessful |
| Properties | Returns a reference to the DBEngine object's Properties collection |
| SystemDB | Defines the Jet engine's workgroup information file |
| Version | Indicates the version of the Jet engine |

*Table 2-3. DBEngine Methods*

| Method | Description |
|---|---|
| BeginTrans | Begins a new transaction |
| CommitTrans | Ends a transaction and saves the changes |
| CompactDatabase | Compacts a closed database |
| CreateDatabase | Creates a new database |
| CreateWorkspace | Creates a Workspace object |
| Idle | Suspends processing to allow the database engine to complete any pending tasks |
| OpenConnection | Opens a database connection |
| OpenDatabase | Opens a database |
| RegisterDatabase | Stores connection information for an ODBC data source in the system registry |
| RepairDatabase | Repairs a corrupted database |
| Rollback | Ends a transaction and discards its changes |
| SetOption | Temporarily overrides default configuration settings |

It's worth emphasizing that all of the members of the DBEngine object are globally available to any VB or VBA application (although not to VBS host applications) that has a reference to the DAO object library. In other words, although you can reference each member of the DBEngine object by explicitly including a reference to the DBEngine object, you can also reference the member without referencing the DBEngine object. For example, the following two statements are identical:

```
    Set db = DBEngine.Workspaces(0).Databases(0)
    Set db = Workspaces(0).Databases(0)
```

Similarly, the following two method calls are identical:

```
    Set db = DBEngine.OpenDatabase(NORTHWIND)
    Set db = OpenDatabase(NORTHWIND)
```

The global members of the DBEngine Object are listed in Table 2-4.

*Table 2-4. Global Members of the DBEngine Object*

| DBEngine Member | Type | Global Availability |
|---|---|---|
| BeginTrans | Method | Yes |
| CommitTrans | Method | Yes |
| CompactDatabase | Method | Yes |
| CreateDatabase | Method | Yes |
| CreateWorkspace | Method | Yes |
| DefaultPassword | Property | Yes |
| DefaultType | Property | Yes |
| DefaultUser | Property | Yes |
| Errors | Property/Collection | Yes |
| Idle | Method | Yes |
| IniPath | Property | Yes |
| LoginTimeout | Property | Yes |
| OpenConnection | Method | Yes |
| OpenDatabase | Method | Yes |
| Properties | Property/Collection | Yes |
| RegisterDatabase | Method | Yes |
| RepairDatabase | Method | Yes |
| Rollback | Method | Yes |
| SetOption | Method | Yes |
| SystemDB | Property | Yes |
| Version | Property | Yes |
| Workspaces | Property/Collection | Yes |

The following sections document the DBEngine object's methods and properties, with the exception of the three properties (Errors, Properties, and Workspaces) that return collections.

### *Access to the DBEngine Object*

*Creatable*

Yes

*Returned by*

The DBEngine object is the top-level object in the DAO object model.

## *DBEngine Properties*

### *DefaultPassword* WO

#### *Data Type*

String

#### *Description*

When it is initialized, uses a case-sensitive string to set the password used to create the default workspace. The password string can be 1–20 characters in length for Jet workspaces or any length for ODBCDirect workspaces. Any character is permitted except ASCII 0. By default, DefaultPassword is a zero-length string, which means that the database is not password protected. DefaultPassword must be set before the default workspace is used in order to have any effect. Use this method if you want to assign a certain password to all new databases. See the DefaultUser entry for a code sample using this method.

### *DefaultType* WO

#### *Data Type*

Long

#### *Description*

Sets or returns a value dictating what type of workspace (Jet or ODBCDirect) the next Workspace object created will be. The property can be set to the values listed in Table 2-5.

*Table 2-5. The Values of the DefaultType Property*

| Named Constant | Value | Description |
|---|---|---|
| `dbUseJet` | 2 | Creates Workspace objects connected to the Jet engine |
| `dbUseODBC` | 1 | Creates Workspace objects connected to an ODBC data source |

### *DefaultUser*

#### *Data Type*

String

### *Description*

When it is initialized uses a string 1–20 characters in length to set the user name used to create the default workspace. Alphabetic characters, accented characters, numbers, spaces, and symbols are permitted, except for the characters listed in Table 2-6, leading spaces, and control characters (ASCII 00 to ASCII 31). By default, DefaultUser is set to "admin."

Normally, user names aren't case sensitive, except when you are recreating a user account created or deleted in another workgroup. In that case, the user name must be a case-sensitive match to the original name.

*Table 2-6. Characters Not Permitted in DefaultUser Strings*

| Character | ASCII Number | Description |
|---|---|---|
| " | 34 | Double quotes |
| / | 47 | Forward slash |
| \ | 92 | Backslash |
| [ | 91 | Left bracket |
| ] | 93 | Right bracket |
| : | 58 | Colon |
| \| | 124 | Pipe |
| < | 60 | Less than |
| > | 62 | Greater than |
| + | 43 | Plus |
| = | 61 | Equal sign |
| ; | 59 | Semicolon |
| , | 44 | Comma |
| ? | 63 | Question mark |
| * | 42 | Asterisk |

## *IniPath*

### *Data Type*

String

### *Description*

Sets or returns a value indicating the Windows Registry key containing information about Microsoft Jet database engine settings or parameters needed for installable ISAM (Indexed Sequential Access Method) databases such as Excel, dBASE, and Paradox. This property must be set *before* invoking any other DAO code, or the change has no effect.

You can use either the HKEY_LOCAL_MACHINE or the HKEY_LOCAL_USER key (as a String) with this property. If you don't supply a root key, HKEY_LOCAL_MACHINE will be used as the default. When setting the property, note that the DAO engine does not test for the existence of the registry key; it simply assigns the string you specify to the IniPath property.

The code sample from Access 97 and Access 2000 Help does not work if run from Microsoft Access, presumably because of some "under the hood" initialization of DAO by Access itself. (Recall that the IniPath property must be set before any other DAO code is invoked.) It does work properly if run from Visual Basic or any other VBA-hosted environment.

### *Word VBA Code*

This example shows how to use the IniPath property to retrieve and set the value of the registry key containing information about the Jet database engine or installable ISAM driver settings, as shown in Figure 2-4.

```
Private Sub cmdIniPath_Click()

On Error GoTo cmdIniPath_ClickError

   Debug.Print "Original IniPath setting = " & _
         IIf(DBEngine.IniPath = "", "[Empty]", DBEngine.IniPath)
      DBEngine.IniPath = _
         "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\" & _
         "Jet\3.5\ISAM Formats\Excel 3.0"
   Debug.Print "New IniPath setting = " & _
      IIf(DBEngine.IniPath = "", "[Empty]", _
          DBEngine.IniPath)

cmdIniPath_ClickExit:
   Exit Sub

cmdIniPath_ClickError:
   MsgBox "Error No: " & Err.Number & "; Description: " & Err.Description
   Resume cmdIniPath_ClickExit
End Sub
```

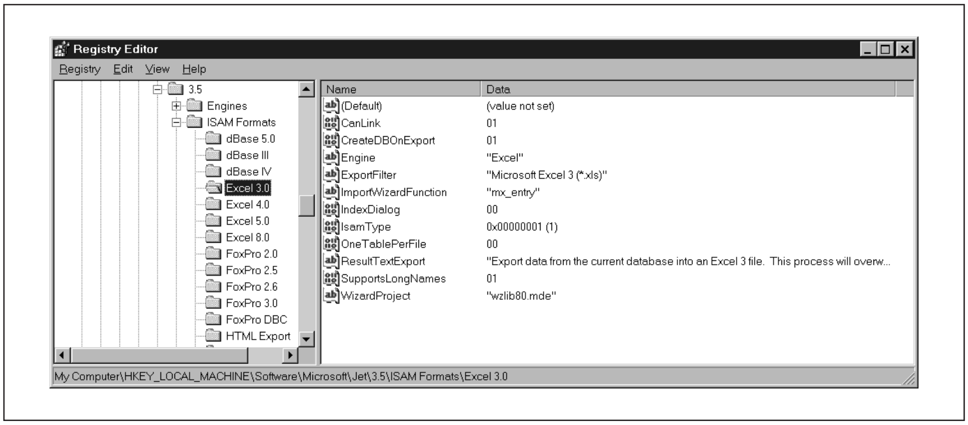
## *LoginTimeout*

### *Data Type*

Integer

### *Description*

Sets or returns the number of seconds allowed before an error occurs when you try to log on to an ODBC database. The default value is 20 seconds. If LoginTimeout is set to 0, no timeout will occur.



| Name | Data |
|---|---|
| (Default) | (value not set) |
| CanLink | 01 |
| CreateDBOnExport | 01 |
| Engine | "Excel" |
| ExportFilter | "Microsoft Excel 3 (*.xls)" |
| ImportWizardFunction | "mx_entry" |
| IndexDialog | 00 |
| IsamType | 0x00000001 (1) |
| OneTablePerFile | 00 |
| ResultTextExport | "Export data from the current database into an Excel 3 file.  This process will overw... |
| SupportsLongNames | 01 |
| WizardProject | "wzlib80.mde" |

My Computer\HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\3.5\ISAM Formats\Excel 3.0

*Figure 2-4. The Windows 98 Registry Editor, showing Excel ISAM formats*

## *SystemDB*

### *Data Type*

String

### *Description*

Sets or returns the workgroup information file (typically *System.mdw*) for Microsoft Jet workspaces. In the interface, you can change to a different workgroup file by using the Workgroup Administrator applet (*Wrkgadm.exe*), usually found in the Windows system folder for Access 97, and the Office folder for Office 2000. Typically, you need to change to a different workgroup information file in order to log on to a secured database.

## *Version* RO

### *Data Type*

String

### *Description*

Returns a string representing the version of DAO currently in use. Table 2-7 shows how Jet versions map with various Microsoft application versions.

*Table 2-7. Jet Engine and Application Versions*

| Jet Version and Year | Access | Visual Basic | Excel | Visual C++ |
|---|---|---|---|---|
| 1.0 (1992) | 1.0 | N/A | N/A | N/A |
| 1.1 (1993) | 1.1 | 3.0 | N/A | N/A |

*Table 2-7. Jet Engine and Application Versions (continued)*

| Jet Version and Year | Access | Visual Basic | Excel | Visual C++ |
|---|---|---|---|---|
| 2.0 (1994) | 2.0 | N/A | N/A | N/A |
| 2.5 (1995) | N/A | 4.0 (16-bit) | N/A | N/A |
| 3.0 (1995) | 95 (7.0) | 4.0 (32-bit) | 95 (7.0) | 4.x |
| 3.5 (1996) | 97 (8.0) | 5.0 | 97 (8.0) | 5.0 |
| 3.6 (1999) | 2000 (9.0) | N/A | N/A | N/A |

### *VBA Code*

This example reports on the Jet version currently in use in the running database:

```
Private Sub cmdVersion_Click()

   MsgBox "Currently using Jet, v. " & DBEngine.Version

End Sub
```

## *DBEngine Methods*

### *BeginTrans*

`DBEnginw.BeginTrans()`

The BeginTrans method is listed in Access 97 Help as a method of the DBEngine object. Actually, it is a method of the Workspace object, one of the members of the Workspaces collection under the DBEngine object, so it will be discussed in Chapter 4, *Workspaces Collection and Workspace Object*. This error has been corrected in Access 2000 Help.

### *CommitTrans*

`DBEngine.CommitTrans()`

As with BeginTrans, this method is actually a method of the Workspace object and will be discussed in Chapter 4.

### *CompactDatabase*

DBEngine.CompactDatabase *srcname*, *dstname*, [*dstlocale*], [*options*], [*srclocale*]

| Argument | Data Type | Description |
|---|---|---|
| *srcname* | String | The filename (including extension) of a closed database. May include full path and can be in UNC convention (*\\server1\share1\dir1\db1.mdb*). If *srcname* is currently open, DAO generates runtime error 3049 ("Can't open database…") or 3356 ("You attempted to open a database that is already opened exclusively…"). |

| Argument | Data Type | Description |
|---|---|---|
| *dstname* | String | The filename and path of the new, compacted database. Must be different from *srcname*. If *dstname* already exists, runtime error 3204, "Database already exists," is generated. |
| *dstlocale* | Variant | (Optional) Sets the collating order for creating *dstname*. If omitted, the locale is the same as that of *srcname*. See Table 2-8 for a list of constants that can be used for this argument and their values. Help says that you can create a password for *dstname* by concatenating the password string (starting with `";pwd="`) with a constant in the *dstlocale* argument to save having to specify two parameters. If *dstname* is to be a password-protected database that uses the same locale as *srcname*, you can omit the *dstlocale* constant and supply just the password preceded by the string `";pwd="`. |
| *options* | Integer | (Optional) Defines the format (version and encryption) of the database. See Table 2-9 for a list of constants that can be used for this argument, and their integer values. If omitted, the encryption and version of *dstname* is the same as that of *srcname*. If supplied, the version constant must represent the same or a later version than that of *srcname*. |
| *srclocale* | Variant | (Optional) For password-protected databases, a Variant containing a String expression. The string `";pwd="` must precede the password. This setting is ignored if you include a password setting in the *dstlocale* argument. |

This method must have been changed at the last moment before the release of Office 97, because Access 97 Help incorrectly lists the parameters as *olddb*, *newdb*, *locale*, *options*, and *password*. The correct parameter names are listed in the IntelliSense popup. These incorrect parameter names are still listed in Access 2000 Help.

*Table 2-8. The dstlocale Named Constants*

| Named Constant | Value | Description |
|---|---|---|
| `dbLangGeneral` | `";LANGID=0x0409;CP=1252;COUNTRY=0"` | English, German, French, Portuguese, Italian, and Modern Spanish |
| `dbLangArabic` | `";LANGID=0x0401;CP=1256;COUNTRY=0"` | Arabic |
| `dbLangChineseSimplified` | `";LANGID=0x0804;CP=936;COUNTRY=0"` | Simplified Chinese |
| `dbLangChineseTraditional` | `";LANGID=0x0404;CP=950;COUNTRY=0"` | Traditional Chinese |
| `dbLangCyrillic` | `";LANGID=0x0419;CP=1251;COUNTRY=0"` | Russian |
| `dbLangCzech` | `";LANGID=0x0405;CP=1250;COUNTRY=0"` | Czech |

*Table 2-8. The dstlocale Named Constants (continued)*

| Named Constant | Value | Description |
|---|---|---|
| `dbLangDutch` | `";LANGID=0x0413;CP=1252;COUNTRY=0"` | Dutch |
| `dbLangGreek` | `";LANGID=0x0408;CP=1253;COUNTRY=0"` | Greek |
| `dbLangHebrew` | `";LANGID=0x040D;CP=1255;COUNTRY=0"` | Hebrew |
| `dbLangHungarian` | `";LANGID=0x040E;CP=1250;COUNTRY=0"` | Hungarian |
| `dbLangIcelandic` | `";LANGID=0x040F;CP=1252;COUNTRY=0"` | Icelandic |
| `dbLangJapanese` | `";LANGID=0x0411;CP=932;COUNTRY=0"` | Japanese |
| `dbLangKorean` | `";LANGID=0x0412;CP=949;COUNTRY=0"` | Korean |
| `dbLangNordic` | `";LANGID=0x041D;CP=1252;COUNTRY=0"` | Nordic languages (Microsoft Jet database engine version 1.0 only) |
| `dbLangNorwDan` | `";LANGID=0x0414;CP=1252;COUNTRY=0"` | Norwegian and Danish |
| `dbLangPolish` | `";LANGID=0x0415;CP=1250;COUNTRY=0"` | Polish |
| `dbLangSlovenian` | `";LANGID=0x0424;CP=1250;COUNTRY=0"` | Slovenian |
| `dbLangSpanish` | `";LANGID=0x040A;CP=1252;COUNTRY=0"` | Traditional Spanish |
| `dbLangSwedFin` | `";LANGID=0x040B;CP=1252;COUNTRY=0"` | Swedish and Finnish |
| `dbLangThai` | `";LANGID=0x041E;CP=874;COUNTRY=0"` | Thai |
| `dbLangTurkish` | `";LANGID=0x041F;CP=1254;COUNTRY=0"` | Turkish |

*Table 2-9. The Options Named Constants*

| Named Constant | Value | Description |
|---|---|---|
| `dbEncrypt` | 2 | Encrypts the database while compacting |
| `dbDecrypt` | 4 | Decrypts the database while compacting |
| `dbVersion10` | 1 | Creates a database that uses the Microsoft Jet database engine version 1.0 file format while compacting |
| `dbVersion11` | 8 | Creates a database that uses the Microsoft Jet database engine version 1.1 file format while compacting |
| `dbVersion20` | 16 | Creates a database that uses the Microsoft Jet database engine version 2.0 file format while compacting |