Макроси в Access

1. Същност и предназначение на макросите

Макросът автоматично изпълнява едно или няколко действия. Access предоставя възможният списък от действия, които той може да изпълни при създаването на макрос. Когато се стартира един макрос, Access изпълнява действията, в последователността, в която те са включени в него.

Макросите могат да се използват за следните цели:

За да се заставят формите и отчетите да работят по удобен за Вас начин. Например, може често да се налага използването на две форми, за да се разглеждат свързани данни. За да се отвори втората форма (само когато това е необходимо), може да се създаде бутон (стартиращ макрос) в първата, при натискането на който да се отваря втората форма.

За автоматично търсене и филтриране на записи. Макросите могат да повишат скоростта на процесите за търсене на записите, които Ви са необходими. Например, може да се присвои макрос към бутон от форма, които автоматично да филтрира записите в подформа.

За задаване на стойности на контроли. Посредством макрос, може да се присвои стойност на контрол от форма, която е резултат от пресмятане или пък на стойност получена от друга форма. Например, когато се въведе факултетен номер на студент във форма, може да се използва макрос, който да извлече името на студента и да го изведе в контрол от формата.

За осигуряване верност на данните. Макросите са идеално средство, когато трябва да се управляват данните от форма. Например, може да се използва макрос за да се

извеждат различни информационни съобщения в зависимост от конкретни стойности на данни от едно поле.

За задаване на характеристики на форми, отчети и контроли. Макросите могат да намерят приложение и при задаване на повечето от характеристиките на форми, отчети и контроли. Например, може да се създаде макрос, който да скрива една форма, която не трябва да се вижда върху екрана, но която в същият момент трябва да бъде отворена.

За автоматизиране трансфера на данни. Макросите могат да намерят приложение и при автоматизирането на процесите по въвеждане и извеждане на данните между различните файлови формати. Например, ако в края на всяка седмица се налага да се трансферират данни към Excel, то може да се създаде макрос за автоматизиране на тази задача и тя да се сведе до простото щракване върху един бутон от форма.

За създаване на потребителска работна среда. Възможно е да се използва макрос за автоматично отваряне на група запитвания, форми и отчети, когато се отвори една база от данни. В допълнение на това, възможно е да се персонализират ивиците с бутони за формите посредством макроси.

2. Създаване на макрос

Последователността на действията по създаване на макрос е следната:

1. 1. В прозорец Database, се избира подпрозорец Macro.

2. Натиска се бутон New, в отговор на което Access отваря прозорец Macro (вж. Фиг. 15.1), в който се избират действията, които ще се изпълняват от макроса.

Горната част на прозорец Macro се използва за добавяне на действия. Желателно е добавянето и на коментарен текст, обясняващ същността на всяко от действията. Долната част на прозорецът се ползва за задаване на аргументи за отделните действия. Аргументите предоставян на Access допълнителна информация за това, как се изпълнява действието - такава като кой обект или дата се използват.

3. Добавяне на действия към макрос

Възможно е добавяне на действие към макрос по два начина. Възможно е да се избере действие от списъка с действия в Macro прозорецът, или за обикновените действия с обектите в базата от данни, да се приложи техниката на “влачене и пускане” на обект от прозорец Database в клетката за действие в прозорец Macro.

Добавяне на действие към макрос чрез избирането му от списък

1. В прозорец Macro се чуква в първата свободна клетка на колонката Action.

2. Натиска се появилият се в дясно на клетката бутон за отваряне на списъкът с възможните действия или (ако се знае начина за изписване му) действието се изписва директно в клетката.

3. Добавя се коментарен текст (по желание) за действието.

4. Задават се аргументи за действието, ако това е необходимо.

Добавяне на действие чрез “влачене и пускане”

1. Позиционират се двата прозореца - Database и Macro, така че и двата да са разположени удобно за работа върху екрана.

2. В Database прозорецът се прави активен този подпрозорец, който съдържа обекта, който ще се “влачи и пуска”.

3. Избира се нужният обект и се “влачи и пуска” от Database прозорец в първата свободна клетка на Macro прозорец.

* Ако обекта, който е “влачен” е таблица, запитване, форма или отчет, то Access добавя действие, което отваря таблицата, запитването, формата или отчета.

* Ако обекта, който е “влачен” е макрос, то Access добавя действие за стартиране на макроса.

Преподреждане и изтриване на действия

Когато последователността на включените в макроса действия не ви удовлетворяват, то е възможно да се промени тяхната последователност.

За промяна последователността на действията, е нужно да се избере нужното действие чрез селектора му, намиращ се в лявата част на името на действието. След това се ползва техниката “влачене и пускане” за преместването на избраното действие на нова позиция в последователността от действия.

За изтриване на действие от макрос, е нужно то да се избере и след това се натиска клавиш Del от клавиатурата.

4. Задаване на аргументи

След като е добавено действие към макрос, трябва да се зададат аргументи за действието в долната половина на Macro прозорецът. Тези аргументи предоставят на Access допълнителна информация за това, как да се изпълни добавеното в макроса действие.

При изпълнението на тези действия трябва да се знае:

1. Стойността за конкретен аргумент може да се въвежда директно в съответната текстова кутия, или да се избира от списък.

2. Основно правило при създаване на макрос е, че трябва да се задават аргументите в последователността, в която те се извеждат в macro прозорецът, тъй като зададената стойност за един аргумент в много от случаите предопределят възможните стойности за следващите аргументи.

3. Ако се добави действие в макрос чрез “влачене и пускане” на обект от прозорец Database, Access автоматично настройва съответните аргументи за това действие.

4. Може да се използват изрази, предшествани от знак “=” за указване на произволен аргумент за действието от макроса. Не може да се ползва израз за задаване на следните аргументи:

Аргумент

Действие

Object Type

Close, DeleteObject, GoToRecord, OutputTo, RepaintObject, Rename, Save; SelectObject, SendObje

Source Object Type

CopyObject

Database Type

TransferDatabase

Spreadsheet Type

TransferSpreadsheet

Specification Name

TransferText

Toolbar Name

ShowToolbar

Output Format

OutputTo, SendObject

All arguments

RunCommand

С командата File | Print Preview, след като е избран един макрос се стартира Print Definition Wizard за разглеждане на макроса в този режим или за отпечатването му.

5. Съхранение на макрос

Преди един макрос да бъде стартиран трябва да бъде съхранен. За целта се изпълнява командата File | Save.

Ако за пръв път се съхранява макрос, то изпълнението на тази команда изисква и въвеждане на подходящо име, съответстващо на заложените в макроса действия.

6. Стартиране на макрос

Стартиране на макрос от прозорец Macro

Отворен в прозорец Macro макрос може да бъде стартиран чрез изпълнението на команда Run | Run или чрез бутон Run от ивицата с бутони.

Стартиране на макрос от прозорец Database, както и от всеки един произволен прозорец на базата от данни

Един избран в подпрозорец Macro на прозорец Database се стартира чрез изпълнение на командата Tools | Macro | Run Macro... . Отваря се диалогов прозорец от който трябва да се избере името на макроса от списъка на наличните макроси и се натиска бутон OK.

Стартиране на макрос от друг макрос

За целта трябва да се добави във викащият макрос действието Run Macro.

За аргумента Macro Name се указва името на виканият макрос.

Стартиране на макрос в отговор на event във форма или отчет

За стартирането на макрос в отговор на събитие, настъпило във форма или отчет трябва в съответната event характеристика да се укаже името на макроса.

7. Създаване на Macro Group

Понякога може да се прецени като по-удобно няколко отделни макроса да бъдат представени като един в Macro прозореца. Посредством групиране на макроси се създава т.нар. macro group. Отделните макроси от групата се стартират само независимо един от друг.

Да приемем, че в една форма е нужно да присъстват три бутона, всеки един от които отваря една таблица. Едно от решенията е за всеки бутон да се създаде отделен макрос, и всеки един от тези макроси ще присъства с името си в Macro прозореца (което води до това, че в този прозорец ще има много имена и сходните макроси трябва да се разглеждат в различни Design прозорци). Другият подход е сходните макроси да се обединят в един макрос (наречен macro group), като само неговото име ще присъства в Macro прозорецът. Съставните макроси на макроса могат да се разглеждат в един Design прозорец.

Създаване на macro group

1. Изпълняват се началните стъпки по създаване на макрос.

2. При активен прозорец Macro се изпълнява командата View | Macro Names или се натиска бутон Macro Names от лентата с инструменти.

В резултат на това действие отляво на съществуващите две колонки в прозорец Macro се визуализира нова колонка с име Macro Name (вж. Фиг. Xxxxx).

1. 2. В колонката macro Name се изписват имената на всеки един модул в създавания модул.

Забележка: Ако се затвори прозорец Macro с активизирана за показване колонка Macro Name, то при следващото отваряне на този прозорец тя отново ще присъства.

Стартиране на макрос от macro group

Където е необходимо да се ползва макрос включен в macro group, трябва да се изпише името на macro group, свързваща точка и след нея името на макроса. За активизиране на макрос с име “Parvi macros” от macros group с име Macros201 (вж. Фиг. 14.2.), то трябва записването да бъде: Macros201.Parvi Macros.

Ако се стартира macro group от прозорец Database, чрез бутон Run или чрез команда Tools | Macro | Run Macro, то изпълнението му започва от първият ред където има указано действие и продължава до срещането на ново име в колонката Macro Name, т.е. в представената на Фиг. 14.2. macro group, ще се изпълнят първите две действия и изпълнението на Macros201 ще бъде преустановено при срещането на името “Vtori macros” в колонката Macro Name. За стартиране на специфичен макрос (макрос участващ в macro group) по описваният тук начин трябва да се изпълни команда Tools | Macro | Run Macro и в диалоговият прозорец Run Macro да се избере от списъка на полето Macro Name името на специфичния макрос.

8. Задаване на имена на контроли в изрази

Понякога се налага да се ползва стойност от контрол на форма или отчет в макрос. В такива случаи трябва да се ползва следният синтаксис:

Forms![*formname*]![*controlname*]

Reports![*formname*]![*controlname*]

Access предоставя и възможност за съкратено задаване име на контрол от форма или отчет, от които е стартиран, като синтаксиса в такъв случай е:

[*controlname*]

Допустимо е да става обръщане и към стойност от поле на таблица, на базата на която е построена формата от която е стартиран макроса (в случаите когато формата не съдържа всички полета от таблицата). Допуска се и извличане на стойност от подформа или подотчет на формата/отчета, от където е стартиран макроса. Допуска се извличане на повечето от характеристиките на контролите от форма или отчет, от където е стартиран макроса.

9. Използване на условия в макроси

При създаването на един макрос може да се наложи изпълнението на различни действия в зависимост от това, дали едно условие е изпълнено или не е изпълнено. В такива случаи в макрос може да бъде указано условие, при наличието на което да се изпълнят едно или няколко действия.

Условието е израз, който еволюира до стойност true или false. В следващата таблица са показани примери на условия, които биха могли да присъстват в един макрос.

Use this expression	To carry out the action if
[City]="Paris"	Paris is the City value in the field on the form from which the macro was run.

DCount("[OrderID]", "Orders")>35

There are more than 35 entries in the OrderID field of the Orders table.

DCount("*", "Order Details", "[OrderID]=Forms![Orders]![OrderID]")>3

There are more than three entries in the Order Details table for which the OrderID field of the table mat

[ShippedDate] Between #2-Feb-1995# And #2-Mar-1995#

The value of the ShippedDate field on the form from which the macro is run is no earlier than 2-Feb-19

Forms![Products]![UnitsInStock]100

The value in the Country field on the form from which the macro is run is UK, and the value of the Total

[Country] In ("France", "Italy", "Spain") And Len([PostalCode])5

The value in the Country field on the form from which the macro is run is France, Italy, or Spain, and th

MsgBox("Confirm changes?",1)=1

You click OK in a dialog box that the MsgBox function displays. If you click Cancel in the dialog box, Mi

Tip To cause Access to temporarily ignore an action, enter False as a condition. Temporarily ignoring an action can be helpful when you are trying to find problems in a macro.

Условието се въвежда в колонката Condition на прозорецът Macro. Ако условието е вярно (true), то се изпълнява действието от същият ред. Възможно е да се задава изпълнението на няколко действия при положение, че условието е вярно, като пред следващите действия в колонката Condition се постави многоточие.

Последователността на работа при добавяне на условие в макрос е следната:

1. При отворен макрос в прозорец Macro се изпълнява командата View | Conditions или се натиска бутон Conditions от лентата с бутони.

В резултат на това действие Access извежда Conditions колонката в прозорец Macro.

2. В колонката Conditions се въвежда условието в реда за който ще се използва условието.

3. В колонката Action се въвежда действието, което ще се изпълни, ако условието е вярно.

Ако е нужно Access да изпълни повече от едно действие в зависимост от въведеното условие, то се въвежда многоточие в колонката Condition за всяко едно от действията.

Когато се стартира макросът, Access преобразува всеки израз от колонката Condition до стойност true или false. Ако стойността е false, Access не изпълнява действието и продължава със следващото действие, пред което няма многоточие.

10. Откриване на грешки в макрос

Access предоставя на потребителите възможност за търсене на съществуващите грешки в един макрос чрез постъпковото му изпълнение. [1] [1]

Ако един макрос не дава очакваните резултати, или ако е нужно да се видят резултатите от изпълнението на всяко едно действие, за да сме уверени че е указана коректна последователност на отделните действия, се използва възможността за постъпковото му изпълнение. По този начин е възможно да се изследват резултатите на всяко едно действие от макроса и открие това от тях, изпълнението на което води до некоректните резултати.

Последователността на работа е следната:

1. Отваря се макросът в Design View.

2. Изпълнява се командата Run | Single Step или се натиска бутон Single Step от ивицата с бутони.

Access поставя знак за отметка пред командата Single Step, от което е видно, че тя е активна.

3. Стартира се макросът по един от възможните начини.

Access извежда Macro Single Step диалоговата кутия (вж. Фиг. 14.4.), в която се виждат името на макроса, името на първото действие в макроса и аргументите за изпълнението на действието, като тези аргументи се извеждат в този прозорец в последователността им, в която се появяват в Macro прозорецът. Ако има зададено условие за изпълнението на действието, в диалоговата кутия се извежда самото условие и до каква стойност еволюира действието - true или false. Ако действието няма зададено условие, то в Condition кутията се извежда “True”.

4. Натиска се бутон Step, за да се изпълни действието, което е показано в диалоговата кутия. Ако в диалоговата кутия има условие, което еволюира до False, то действието не се изпълнява. Step бутонът води до активизирането на следващото действие пред което няма многоточие.

Следващата таблица описва предназначението на бутоните в Macro Single Step диалоговата кутия:

Бутон	Действие на бутона
Step	Изпълнява действието показано в диалоговата кутия. Ако в макроса няма открита грешка се из
Halt	Прекъсва изпълнението на макроса и затваря Macro Single Step диалоговата кутия.

Continue

Изключва постъпковото изпълнение и изпълнява останалите действия от макроса така както

Macros: Tips and Tricks

Last updated 3 February 1997

Robert Flake
Microsoft Corporation

Introduction

The following articles contain useful information about using macros in Microsoft® Access 97 and converting macros to Visual Basic® for Applications code.

Macros are very useful for automating simple tasks, such as carrying out an action when the user clicks a command button. You don't need to know how to program to use macros. Macros can perform a number of the common tasks that you can also use Visual Basic code to perform. However, using Visual Basic code instead of macros gives you much more flexibility and power, and there are many things you can only do in code, such as returning values or iterating through recordsets.

In Microsoft Access 97, you can convert your existing macros to Visual Basic code. You can also convert custom menu bars and shortcut menu bars, which were created using menu bar macros in previous versions of Microsoft Access, to command bars.

Using macros, and converting macros to Visual Basic code, are described in the Microsoft Access 97 Help topics. The following articles are provided to give useful tips and solutions to common macro problems that may not be covered explicitly in the Help topics, or to point out especially useful techniques for working with macros.

• Should I Use a Macro or Visual Basic Code?>

• Example: Finding and Changing a Customer Address>

• Using Methods Instead of Macro Actions>

• Using Macros to Improve Performance in Microsoft Access 97>

• Converting Custom Menu Bars and Shortcut Menus to Command Bars>

• The RunCommand Action Replaces the DoMenuItem Action>

• Should I Use the RunCommand Method or the DoCmd Object Methods?>

Should I Use a Macro or Visual Basic Code?

You can use macros to automate many common tasks in Microsoft Access, such as opening and closing forms or printing reports. Generally speaking, you should use macros if you have simple tasks you want to automate, and you don't want to have to learn to program in Visual Basic. Since you set arguments for macros in the Macro window, often from a list of selections, you don't have to remember complicated syntax. There are also some situations that either

require macros or where macros may be a better choice than Visual Basic code:

- Making key assignments. If you want a particular key or key combination to carry out an action or actions throughout your application, you must use the key-assignment macro, AutoKeys. You also need to use the AutoKeys macro to assign a key or key combination to a built-in command button control on a command bar.

- Performing actions when your database first opens, such as opening a switchboard form or setting certain properties. You can use an AutoExec macro to do this. You can, however, also use the Startup dialog box (available on the Tools menu) and/or set one or more of the startup properties in Visual Basic code to perform many common startup tasks.

- Performing simple automation. If you have forms or reports that require only simple automation (for example, the user can click one of several command buttons that do common tasks such as opening a related form, synchronizing records on two forms, or printing the records displayed in the form), you may want to use macros to do this. If you use macros rather Visual Basic code, your form or report will still be a "lightweight" object; that is, an object with no form or report module associated with it. In some cases, lightweight forms and reports may load and display faster than forms and reports with associated code modules.

In other situations, however, you should probably use code. Visual Basic code is more flexible and powerful than macros. Following are some reasons to use Visual Basic code instead of macros. To see an example of performing a task using a macro, and, alternately, using Visual Basic, code, see *Example: Finding and Changing a Customer Address*.

- Visual Basic code is easier to maintain. Macros are separate objects from the forms and reports they're associated with, and it can be difficult and time-consuming to determine which macros apply to a particular form or report. However, the Visual Basic procedures that apply to a particular form or report are stored in the form's or report's accompanying module. Also, when you move a form or report to another database, the Visual Basic code associated with the form or report moves with it.

- In Visual Basic code, you can determine the error number when a particular error occurs, and respond to different errors in different ways. For example, you can display a custom

message box, or bypass a particular error completely. You can't determine what error has occurred when a macro is running.

- You can create and manipulate objects using the Data Access Objects (DAO) and Visual Basic for Applications object models in Visual Basic code. This gives you tremendous power over the objects in your application.

- There are a number of techniques you can use to work with other applications and to do system-level tasks in Visual Basic code. For example, you can check to see if a particular file exists on your system. You can also use Automation and dynamic data exchange (DDE) to communicate with other applications, and you can call functions in the Windows® application programming interface (API) or other dynamic-link libraries (DLLs).

- You can iterate through recordsets, and manipulate records one at a time. With macros, you can only work with an entire set of records.

- You can create your own functions by using Visual Basic code. Although you can use expressions in the Condition column of macros, it's much more efficient to write functions that replace complex expressions. You can also use a custom function to apply a common operation to more than one object. In addition, note that you can't pass an argument to a function or receive a return value in a macro; you can only do this in Visual Basic code.

- More generally, when using Visual Basic procedures, you have all the power and flexibility inherent in structured code. You can pass arguments, receive return values, and use variables for arguments. You can use conditional code structures such as If…Then…Else and Select Case loops. None of these features are available in macros.

For more information, search the Microsoft Access Help index for "macros, compared to Visual Basic code."

If you decide you want to start using Visual Basic code instead of macros in your application, Microsoft Access 97 provides simple procedures that allow you to convert your existing macros

to Visual Basic code.

• To convert the macros associated with a particular form or report to Visual Basic code, open the form or report in Design view, point to Macro on the Tools menu, and then click either Convert Form's Macros To Visual Basic or Convert Report's Macros To Visual Basic.

• To convert a global macro (one not associated with a particular form or report) to Visual Basic code, highlight the name of the macro in the Database window, click Save As/Export on the File menu, and then click Save As Visual Basic Module in the Save As dialog box.

For more information, search the Microsoft Access Help index for "macros, converting."

Example: Finding and Changing a Customer Address

Suppose you want to add a command button to your Orders form that allows you to search the customer records and change a particular customer's address.

To do this through macros, you need to create a custom dialog box and two macros. The custom dialog box, named New Customer Address, contains two text fields, CustomerName and NewAddress. It also has an OK button.

From the OK button, you want to run the *HideForm macro*, which hides the New Customer Address dialog box by setting the Visible property of the dialog box form to False.

On the Orders form, add a command button named ChangeCustomerAddress that runs the *ChangeAddress macro*
. This macro finds the customer whose address you want to change and then changes the address.

These macros work for the task you had in mind. However, they can be hard to keep track of, since there are two macros. There's also only simple error handling. Note that if you enter an invalid or misspelled company name in the dialog box, the macro will change the address of the current record in the form without catching the error. It's much easier to catch these sorts of errors in Visual Basic. To see how you'd perform the same task in Visual Basic, click Visual *Bas ic Example of Finding and Changing a Customer Address*
.

HideForm macro

Condition

Action

Action Arguments

Comment

This macro is attached to the OK button on the New Customer Address form.

Forms![New Customer Address]!CustomerName Is Null Or Forms![New Customer Address]!NewAddr

MsgBox

Message: You must enter a valid company name and a new address.

Beep: Yes

Type: None

If the user doesn't enter a valid company name or a new address, the macro shouldn't run.

…

StopMacro

The ellipsis (...) in the Condition column applies the condition to this action row as well.

SetValue

Item: Forms![New Customer Address].Visible

Expression: False

Hide the New Customer Address form to allow the ChangeAddress macro to resume execution.

ChangeAddress macro

Action

Action Arguments

Comment

This macro is attached to the ChangeCustomerAddress button on the Orders form.

OpenForm

Form Name: New Customer Address

View: Form

Window Mode: Dialog

You want to set the Window Mode argument of the OpenForm action to Dialog so that the macro susp

FindRecord

Find What: =Forms![New Customer Address]!CustomerName

Match: Whole Field

Match Case: No

Search: All

Search As Formatted: Yes

Only Current Field: No

Find First: Yes

Find the first record for the company name that the user entered in the dialog box. Note the equal sign

SetValue

Item: Address

Expression: Forms![New Customer Address]!NewAddress

Change the address. (In the Northwind sample database, you would probably also set the ShipAddres

Close

Object Type: Form

Object Name: New Customer Address

Save: Prompt

Close the dialog box.

Visual Basic Example of Finding and Changing a Customer Address

Attach the following Visual Basic code to the ChangeCustomerAddress button on the Orders form.

```
Function ChangeCustomerAddress() As Boolean

Dim dbs As Database

Dim rst As Recordset

Dim strCustomerName As String, strNewAddress As String

Dim strCriteria As String

On Error GoTo ChangeCustomerAddress_Err

Set dbs = CurrentDb()
```

```
Set rst = dbs.OpenRecordset("Customers", dbOpenDynaset)

strCustomerName = _

InputBox("Enter the company name", "Company Name")

If Len(strCustomerName) = 0 Then Exit Function

strNewAddress = InputBox("Enter the new address", "New Address")

If Len(strNewAddress) = 0 Then Exit Function

strCriteria ="[CompanyName] = " & """" & strCustomerName & """"

With rst

.MoveFirst

.FindFirst strCriteria

If .NoMatch Then

MsgBox("The company name you entered isn't valid.")
```

```
Exit Function

End If

' Set the Address field to strNewAddress.

.Edit

!Address = strNewAddress

.Update

End With

MsgBox("The address has been changed.")

ChangeCustomerAddress = True

ChangeCustomerAddress_Bye:

Exit Function
```

```
ChangeCustomerAddress_Err:

MsgBox Err.Description, vbOKOnly, "Error = " & Err.Number

ChangeCustomerAddress = False

Resume ChangeCustomerAddress_Bye

End Function
```

Using Methods Instead of Macro Actions

Macros are simple to use and perform many common tasks in Microsoft Access. You can use the methods of the DoCmd object to carry out most actions in Visual Basic code. However, in many cases there are other Visual Basic methods (or other language elements or techniques) that accomplish the same tasks and give you more power and flexibility. The following table gives some suggestions regarding Visual Basic language elements that might be used to replace the methods of the DoCmd object. If the DoCmd method is preferred, this is noted.

Macro Action/Method	Visual Basic Language Element
AddMenu	

Obsolete (and not available in Visual Basic). Create custom command bars using the command bars

ApplyFilter

Use SQL to apply the desired filter.

Beep

Beep statement

CancelEvent

Use the Cancel argument for the event in the desired event procedure.

Close

Various object models include Close methods; for Microsoft Access, the Close method of the DoCmd

CopyObject

Use the CopyObject method of the DoCmd object to copy Microsoft Access objects.

DeleteObject

Use the DeleteObject method of the DoCmd object to delete Microsoft Access objects.

Echo

Echo method of the Application object

FindNext

DAO FindNext method

FindRecord

DAO FindFirst, FindLast, FindNext, FindPrevious methods

GoToControl

SetFocus method of the Control object

GoToPage

GoToPage method of the Form object

GoToRecord

SetFocus method of the Control object

Hourglass

Hourglass method of the DoCmd object

Maximize

Maximize method of the DoCmd object

Minimize

Minimize method of the DoCmd object

MoveSize

Use the MoveSize method of the DoCmd object to move and size Microsoft Access objects.

MsgBox

(There is no MsgBox method available in Visual Basic.) Use the MsgBox function.

OpenForm

OpenForm method of the DoCmd object

OpenModule

OpenModule method of the DoCmd object

OpenQuery

OpenQuery method of the DoCmd object. You can also enter SQL statements directly in your Visual B

OpenReport

OpenReport method of the DoCmd object

OpenTable

OpenTable method of the DoCmd object

OutputTo

OutputTo method of the DoCmd object

PrintOut

Print method of the Report object. However, the PrintOut method of the DoCmd object method does p

Quit

Quit method of Application object

Rename

Rename method of the DoCmd object

RepaintObject

Repaint method of the Form object

Requery

Requery method of the Control or Form object; DAO Requery method.

Restore

Restore method of the DoCmd object

RunApp

(There is no RunApp method available in Visual Basic.) Use the Shell function.

RunCode

(There is no RunCode method available in Visual Basic.) Run the code directly.

RunCommand

RunCommand method (applies to either DoCmd or Application object).

RunMacro

RunMacro method of the DoCmd object

RunSQL

Use the SQL statement for the action query.

Save

Use the Save method of the DoCmd object to save Microsoft Access objects.

SelectObject

Use the SelectObject method of the DoCmd object to select Microsoft Access objects.

SendKeys

(There is no SendKeys method available in Visual Basic.) Use the SendKeys statement.

SendObject

SendObject method of the DoCmd object

SetMenuItem

Obsolete except for menu bar macros. Use the command bars object model (the Enabled property an

SetValue

(There is no SetValue method available in Visual Basic.) Set the value directly in code.

SetWarnings

SetWarnings method of the DoCmd object

ShowAllRecords

ShowAllRecords method of the DoCmd object

ShowToolbar

Use the Visible property for command bars and command bar controls. The ShowToolbar method of t

StopAllMacros

Not available in code.

StopMacro

Not available in code.

TransferDatabase

TransferDatabase method of the DoCmd object

TransferSpreadsheet

TransferSpreadsheet method of the DoCmd object

TransferText

TransferText method of the DoCmd object

For an example of using Visual Basic code instead of a macro, see .

Converting Custom Menu Bars and Shortcut Menus to Command Bars

In Microsoft Access 97, you use the Customize dialog box, available by pointing to Toolbars on the View menu and then clicking Customize, to create command bars (menu bars, toolbars, and shortcut menus) and set their properties. You can also use the command bars object model in Visual Basic code to create and modify command bars.

In previous versions of Microsoft Access, you used menu bar macros to create custom and global menu bars and shortcut menus. In a menu bar macro, a set of AddMenu actions defined the menus on the menu bar. Each menu was in turn defined by a menu macro group specified in one of the AddMenu actions' arguments. For each macro in the macro group, an action or set of actions defined a command on the menu. You could use almost any action to define a custom command. The DoMenuItem action was used to add a built-in command to a custom menu.

In Microsoft Access 97, all of your menu bar macros will display the custom menu bars and shortcut menus as before. These menus and menu bars will look like the new command bars. However, you can't modify them in the Customize dialog box or in Visual Basic code. To convert a custom menu bar or shortcut menu to a command bar, highlight the name of the menu bar macro for this custom menu bar or shortcut menu in the Database window, point to Macro on

the Tools menu, and then click Create Menu From Macro, Create Toolbar From Macro, or Create Shortcut Menu From Macro.

When you convert a custom menu bar or shortcut menu to a command bar, the menu bar macro and macro groups that define this menu still exist and the menu bar or shortcut menu carries out the actions in the appropriate menu macro group for each command. If the menu bar macro contains only AddMenu actions (which it should) and all the associated menu macro groups contain only RunCommand actions, the macros that define the custom menu bar or shortcut menu are no longer needed. However, it's wise not to delete menu bar macros and menu macro groups unless you're absolutely sure that the new command bar doesn't rely on them.

Once you've converted a custom menu bar or shortcut menu, you can use the Customize dialog box or the command bar object model to modify the new command bar. You can even use the OnAction property of a command on a command bar to run a different macro or function than the one originally associated with the command in the menu macro. Note, however, that once a custom menu bar or shortcut menu has been converted, modifying its menu bar macro or one of its menu macro groups does not modify the new command bar. If you want to edit the macros and see these changes reflected in the command bar, you must reconvert the menu bar macro. Note that reconverting the macro replaces the command bar and removes any changes you've made using the Customize dialog box or the command bar object model. Once you've converted a custom menu bar or shortcut menu to a command bar, you generally should not use the menu bar macro or menu macro groups to make changes to the command bar.

Note In order to program with command bars in Microsoft Access 97, you must first set a reference to the Microsoft Office object library. Click References on the Tools menu while in module Design view, and select the check box next to Microsoft Office 8.0 Object Library.

The RunCommand Action Replaces the DoMenuItem Action

In previous versions of Microsoft Access, you used the DoMenuItem action (and the DoMenuItem method in Visual Basic code) to carry out built-in menu commands. With the DoMenuItem action, you had to set Menu Bar, Menu Name, Command, Subcommand (and for the method, *version*) arguments. These argument settings specified what menu bar the desired command appeared on (and in what version of Microsoft Access).

In Microsoft Access 97, the RunCommand action and RunCommand method replace the DoMenuItem action and DoMenuItem method. You can use this action or method to carry out any of the built-in menu or toolbar commands.

The RunCommand action has a single argument, Command. In the Macro window, select the command you want to carry out from the drop-down list in the Action Arguments section of the window.

The RunCommand method uses the following syntax

[Application].RunCommand *command*

or

[DoCmd].RunCommand *command*

where *command* is the intrinsic constant corresponding to the menu or toolbar command. The Application or DoCmd prefix is optional. To see a list of the RunCommand method intrinsic constants, search the Microsoft Access Help index for "RunCommand method" and select the "RunCommand Method Constants" topic.

If you have an existing database containing DoMenuItem actions or DoMenuItem methods, these actions or methods will carry out the appropriate menu command when you open the database in Microsoft Access 97. If you convert the existing database to Microsoft Access 97, any DoMenuItem actions are automatically converted to the equivalent RunCommand actions when you open and save a macro containing them. The DoMenuItem action no longer appears in the list of actions in the Macro window. Note that DoMenuItem methods in modules in the database are not converted automatically. You must convert these yourself if you want to replace the DoMenuItem methods with the equivalent RunCommand methods.

Some commands from previous versions of Microsoft Access aren't available in Microsoft Access 97. If you open and save a macro containing a DoMenuItem action that tries to carry out one of these commands, the Command argument is blank for this DoMenuItem action. You must edit the macro and enter a valid RunCommand action, or delete the action.

If your Visual Basic code contains a DoMenuItem method that tries to carry out a command that is no longer available, an error occurs when the code runs. You must edit your Visual Basic code and replace or delete occurrences of such DoMenuItem methods. For a list of commands from previous versions of Microsoft Access that aren't available in Microsoft Access 97, search the Microsoft Access Help index for "RunCommand action" and select the "DoMenuItem Action Commands Not Available with the RunCommand Action" topic.

Should I Use the RunCommand Method or the DoCmd Object Methods?

The methods of the DoCmd object are used in Visual Basic to carry out Microsoft Access macro actions. For example, to carry out the OpenForm action in a Visual Basic procedure, you use the following syntax

DoCmd.OpenForm *arguments*

where *arguments* are the action argument settings you want to use for the OpenForm action.

The RunCommand method is used to carry out a Microsoft Access menu or toolbar command. The RunCommand method replaces the DoMenuItem method used in previous versions of Microsoft Access. To run a particular menu or toolbar command in Visual Basic, use the following syntax

[Application].RunCommand *command*

or

[DoCmd].RunCommand *command*

where *command* is the intrinsic constant corresponding to the menu or toolbar command. The Application or DoCmd prefix is optional. To see a list of the RunCommand method intrinsic constants, search the Microsoft Access Help index for "RunCommand method" and select the "RunCommand Method Constants" topic.

Although in some cases a macro action and a menu command do similar things (for example, the Find action and the Find command on the Edit menu), the methods of the DoCmd object and the RunCommand method are distinct and should be used differently. (The RunCommand method is a method of the DoCmd object, but it has a special purpose: to carry out Microsoft Access menu commands.) The most important distinction is that you when you use the DoCmd methods, you can specify the action argument settings. When you use the RunCommand method, it simply carries out the specified command. If the command brings up a dialog box, the dialog box appears. Using the DoCmd methods allows you to carry out actions in Microsoft Access without having to use menu commands and their dialog boxes.

For example, you may want to have a function procedure that selects the current record in a form and prints it.

```
Function PrintRecord()

' This procedure selects the current record and prints it.

RunCommand acCmdSelectRecord

DoCmd.PrintOut acSelection
```

```
End Function
```

This function uses the RunCommand method and the intrinsic constant acCmdSelectRecord to carry out the Select Record command on the Edit menu, which selects the current record. Then the PrintOut method of the DoCmd object, with the *printrange* argument set to acSelection, prints the selected record. Note that if instead of using the PrintOut method of the DoCmd argument, you use the RunCommand method to carry out the Print command on the File menu, the function selects the current record, but brings up the Print dialog box:

```
Function PrintRecord()

' This procedure selects the current record but doesn't

' automatically print it.

RunCommand acCmdSelectRecord

RunCommand acCmdPrint

End Function
```

If you use the SendKeys statement to send an ENTER key to choose OK in the dialog box, the entire set of records is printed, not just the selected record. You could use a series of SendKeys statements to move to the appropriate field in the dialog box and select the Selected Record(s) option, but the PrintOut method of the DoCmd object accomplishes this much more efficiently.