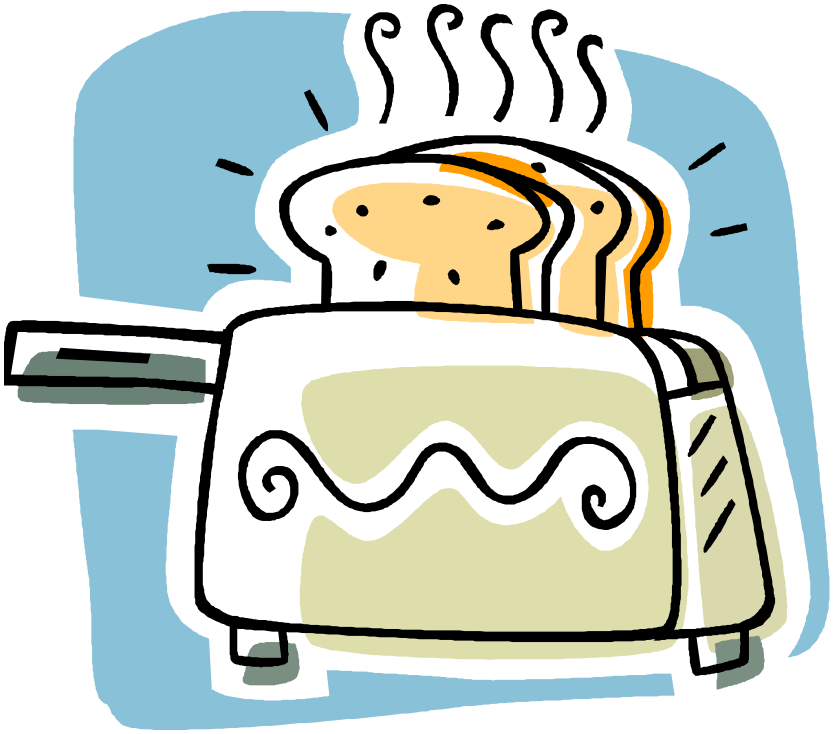
Ingredients for Database Recipes



Dejang Liu
College of DuPage

June ©2006-2011 2nd edition

2006-2011 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Dedication

This publication is dedicated to my loving and supportive father, Chun-Tien. He taught me the key element for success in life: never give up before you try.

Acknowledgements

Thanks to Fu Lin Trade Association's full support in making this book a reality, particularly, the editor. Winshen's technical writing skills have made this a much more comprehensible reading.

Credits should also be given to Mrs. Su-Jwen Huang, the publisher, to her team for designing and packaging this book in a CD format. Her idea to put all materials onto a CD not only saves more trees, but also lightens students' load.

Finally, I thank all students who attended my Access classes for the past 15 years. Without their questions, I would not know what elements a textbook needs to have. This book intends to be a reference resource for any intermediate to advanced Access user. It is an excellent supplement for bridging the gap between beginners and advanced programmers in learning Macro and VBA. You need to have a PDF reader in order to read the book. The PDF reader can be downloaded from the Adobe Acrobat website at **www.adobe.com**

Any suggestions and comments about this book are welcomed. Please send them to either the publisher at **fulin_trade@yahoo.com**, or the author at **liudejan@cod.edu**.

Table of Contents

Access Macros

Appetizers
How to Design an Access database
Chapter 1
Take Advantage of Control Wizard
Flavor 1.1 Record Navigation and Record Operations
Flavor 1.2 Form Operations
Flavor 1.3 Report Operations
Flavor 1.4 Application Categories
Flavor 1.5 Miscellaneous Categories
Flavor 1.6 What's happening in the kitchen?
Chapter 2
Macro Builder and Group Macros
Flavor 2.1 What if the control wizard is inactivated when a command button is created on a form?
Flavor 2.2 Use Macro Builder to Build a Macro
Flavor 2.3 Use Macro Builder to Create a Group Macro
Flavor 2.4 Create a custom switchboard by using option control
Flavor 2.5 How do you convert a macro to a VBA procedure?
Chapter 3
Custom Menu, Toolbars, Pop-up Menu
Flavor 3.1 Why should you use custom menus and toolbars
Flavor 3.2 Create a custom toolbar
Flavor 3.3 Create a custom menu bar
Flavor 3.4 How do you make a submenu
Flavor 3.5 Create custom shortcut menu (popup menu)
Flavor 3.6 Change the Startup settings

Access VBA

Chapter 4
Using Code Builder
Flavor 4.1 Decision structure: IF
Flavor 4.2 Decision structure: nested IF
Flavor 4.3 Select Case structures
Flavor 4.4 Iteration structure: Fixed iteration
Flavor 4.5 Iteration structure: Indefinite iteration
Flavor 4.6 With….End With
Flavor 4.7 Common built-in string-based functions
Flavor 4.8 Common built-in date and time functions
Flavor 4.9 Common built-in conversion functions
Flavor 4.10 Common built-in input validation functions
Flavor 4.11 Code Builder
Flavor 4.12 How to use ActiveX controls

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Chapter 5
Function and Procedures
Flavor 5.1 Basic structure of module
Flavor 5.2 Introductions to Sub procedures
Flavor 5.3 Introduction to function procedure
Flavor 5.4 Many ways to declare variables
Flavor 5.5 Handling errors
Chapter 6
Improve the Taste
Flavor 6.1 Improve the Parameter Query
Flavor 6.2 Finding matches as the value in a range of the second table
Flavor 6.3 Speed up the item selection in one combo box based on the selection in another combo box
Flavor 6.4 Use a cookie cutter: create a template for all custom forms and reports
Flavor 6.5 Display the parameter value on a report
Flavor 6.6 Add line numbers on the report
Flavor 6.7 Make a report with multiple columns
Flavor 6.8 Total on each page
Flavor 6.9 Add a content range indicator on each page
Flavor 6.10 Clean test data after complete the construction
Flavor 6.11 How do you control the enabling/disabling of the Shift key?
Flavor 6.12 How to convert the replica back to a regular database
Flavor 6.13 Reasons why you should consider splitting your database
Flavor 6.14 Remove the security on your database

References

About the Author

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Appetizers: How to Design an Access Database table of contents

Microsoft Access comes with a lot of tools to assist desktop database designers in designing a relational database. It is an easy task to create an Access database, even if you don't have much experience in such database design. This book will cover how to use the tools to create databases. However, the basic concepts of designing relational database should be studied before exploring the tools and wizards.

A relational database is made up of individual tables. These tables are related to each other, either directly or indirectly. Each table consists of related fields or attributes. For example, the Customer table includes the Customer ID, Customer Name, Address, City, State, Zip Code, Contact's First and Last Name, Phone Number, and Position. Those items are related, so they are combined and presented in one table. If you need to create a report listing the orders placed by each customer, you can pull and combine the records from the Customer table and Order table for your report. The first thing you will need to know in order to build these reports or forms is what information you need from the database and how to break the information down into individual tables. The Access will then use the tables you design to provide the needed reports and forms.

Steps in Designing the Database

Step 1: Evaluate the scope of the database project

The Access has its limitations in terms of scalability. Access 2000, 2002, 2003, and 2007/2010 can only handle the database size of up to 2GB. If you expect the size of the database to grow close, or over, 2GB, you may want to consider other options, such as Microsoft SQL Server, IBM DB2, or Oracle from the Oracle Corporation.

Step 2: Analyze the existing database

Most of the time, you will be asked to migrate an existing database to Access. In this case, you will have some preliminary table structures to look at. Of course, in this kind of project, it will be much easier for you to design your table, but you need to be very careful in not copying the problems of the existing database into your new design. Due to the needs of the business or changes in the environment, certain concerns may develop, including: changing the size of the field (extended or reduced), adding new fields (e-mail or web site), modifying data types, or adding new validation rules. You also need to determine if any data element is a "derived field". A *derived field* is a field whose content can be obtained by calculating it from other fields, such as: the commission can be determined by calculating the **Sales Value** and **Commission Rates**. You will then need to figure out if the table structures should be normalized. Normalization will be briefly covered later. If this is a brand-new project, you will need to understand the business operation as much as you can and should start at Step 3 instead.

Step 3: Determine the purpose of the database

Who will be using this database? Talk to the people who will be the primary users. For example, people from the accounting department will be more concerned with

2006 © All Rights Reserved

ISBN 0-9787733-0-6

the account payable/receivable while the management folks will be more interested in the summary of the sales and the inventory.

What are the special requirements? Salespersons will only take orders and will not be able to alter existing orders. Only the manager of the sales can alter and take orders. You will learn how to make one order form for use by both, instead of designing two separate forms to meet each positions' needs.

Think about and try to answer the following questions, among others, when designing your database. What kind of the screen design (i.e. forms) is best for data entry and view? What kind of data summary (i.e. reports) would the users like to have? What kind of search field? How many possible field selections for searches (i.e. criteria)?

Step 4: Consider all possible tables

From Step 3, you have a better understanding of the purpose of the database. You will then start to "pull" the information from Step 3 into separate "groups". "Specific" is a key word in this stage. A Product table, for example, will only store data related to Products. Customers' and Suppliers' information should not be in this table. List all possible tables and remember to:

- Enter any new record into tables only once
- Modify any existing record only once
- Delete any existing record only once

This process can be done through Normalization, which eliminates data redundancy. For instance, say the product's price information has changed; the only place that needs to be updated will be the price field in the **Product** table. You will not need to change the product's prices for all future orders in the **Order** table because of the relationship. Or, for example, say a new customer places a new order. You enter the new customer's basic information into the **Customer** table. A new order can be created by adding the products' information into the **Order** table. Perhaps this customer decides not to keep their order. You will need to cancel the order on the **Order** table, but there is no need to delete the customer information on the **Customer** table.

Step 5: List all possible fields (attributes)

The *field* is the column of each table. You can divide the information you have from Step 4 into pieces; each piece is a field. In order to consider all possible fields for each table, work on one table at a time in this stage. For the **Supplier** table, for instance, you will need the **Supplier ID**, **Supplier name**, **Address**, and **Contact** information. If the field can be derived, this field should not be in the table.

2006-2009 © All Rights Reserved

ISBN 0-9787733-0-6

Step 6: Consider the key field for each table

In this step, you will need to consider possible key field(s) as the primary field(s) on each table. This task will help you to build the relationship between tables. Although it is not necessary to have a key field on each table, it is required to have primary-foreign keys in order to establish the relationship. Also in this stage, you will need to determine the properties (field size, data type, format, etc.) of each field. Here is some important information you need to keep in mind when choosing primary key fields:

- Primary key fields should not have duplicate or null values.
- A primary key field should be easy to remember, as it is often used to look up records. It may be helpful to restricted it to a certain number of letters or digits, or be within a certain range.
- The speed of database operations are influenced by the size of primary keys. Due to this relationship, the size of primary key fields, through properties, can and should be limited. It is best to use the smallest possible size in order to maximize the performance of your database.

Before you move on to the next step, you should have all table information completed at this point. Look at the following Normalization rules to help polish up your table design:

- **First Normal Form (1stNF):** All fields in a table should be atomic. Data should be broken down as much as possible, such as having Last Name in one field and First Name in another, as opposed to putting the Full Name in one field.
- **Second Normal Form (2ndNF):** All non-key fields should be dependent on the primary key. Each table must store data for only one subject. See Step 5.
- **Third Normal Form (3rdNF):** Eliminate all derived fields. The fields should meet all the requirements of 1stNF and 2ndNF.

Step 7: Making a prototype of the database

The rough database structure is done. Now you will need to enter some dummy records, simple and short sample data, into these "potentially correct" tables. The dummy sample record can be easily removed, which will be discussed in **Chapter 6**. The most important issue in this stage is to make sure that the infrastructure of the tables is solid and ready to move on to the next object design, such as queries or forms. It is unwise to rush into an object design without confirming the completion of the table design.

With Access, it is not very burdensome to alter the design of tables before numerous real records are entered. However, it becomes a challenging task, and a designer's nightmare, to do so once the database is built with forms and other objects or filled with tons of data and implemented. It is worthwhile to spend more time in table design and establish a solid foundation before moving into other object designs.

2006-2009 © All Rights Reserved

ISBN 0-9787733-0-6

Step 8: Conduct testing the database

This stage should be conducted offsite and onsite. The offsite testing is performed in your office mainly to test the database for bugs and errors. Try to test it in all possible operating systems. The testers should not be from the database design team. The technical background of testers should range from the savviest of users to the least computer-skilled.

The onsite testing focuses mainly on the integration between your database and the user's existing system. With the completion of the offsite testing, on very few occasions will you find problems caused by your database design. Instead, the causes will be from the user's existing system, such as the update of Access or Windows service pack (SP) or a network configuration.

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

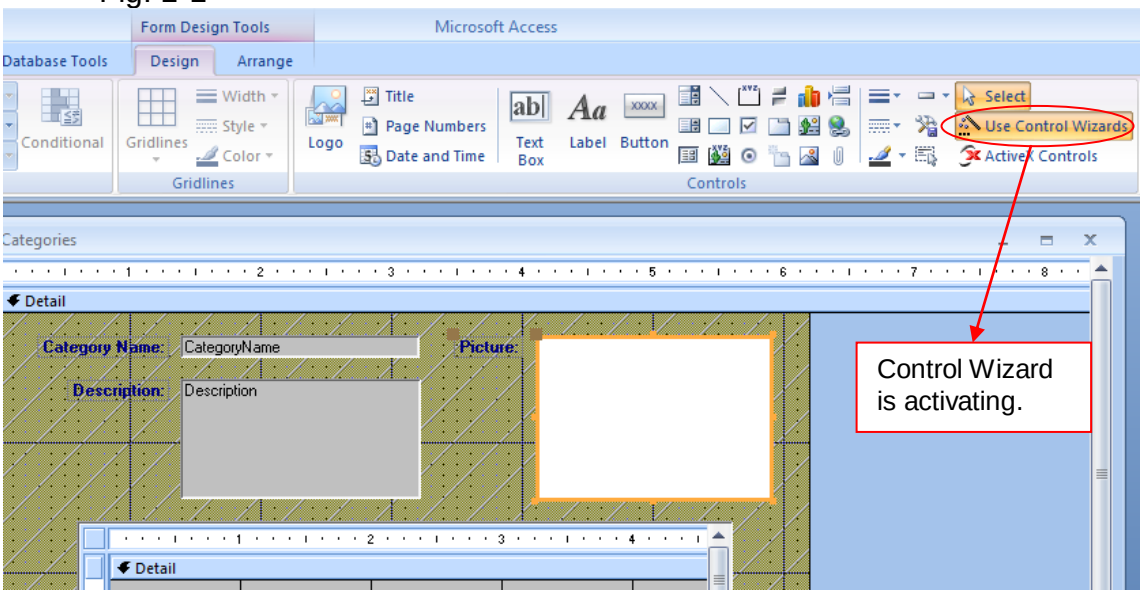
Chapter 1 Take Advantage of Control Wizards

table of contents

The **Control Wizard** creates a fully functioning command without requiring any coding to be written by the developer. Even for experienced users, taking a look at the available controls in the **Control Wizard** is the first thing they do before using either a Macro or Visual Basic for Application (VBA) procedures.

The **Control Wizard** can be found in the Toolbox, which can be accessed from numerous places, as shown in Fig. 1-1. Make sure to click **Control Wizard** to activate it, like the one in Fig.1-1, if it is not already activated.

Fig. 1-1



To use the **Control Wizard**, you need to select a command button, combo box, list box, option group, or sub form/sub report from the Toolbox toolbar.

The first one for discussion is the command button. Make sure the **Control Wizard** is selected (it may be a sunken image, Fig. 1-2) before clicking the command button.

Fig. 1-2

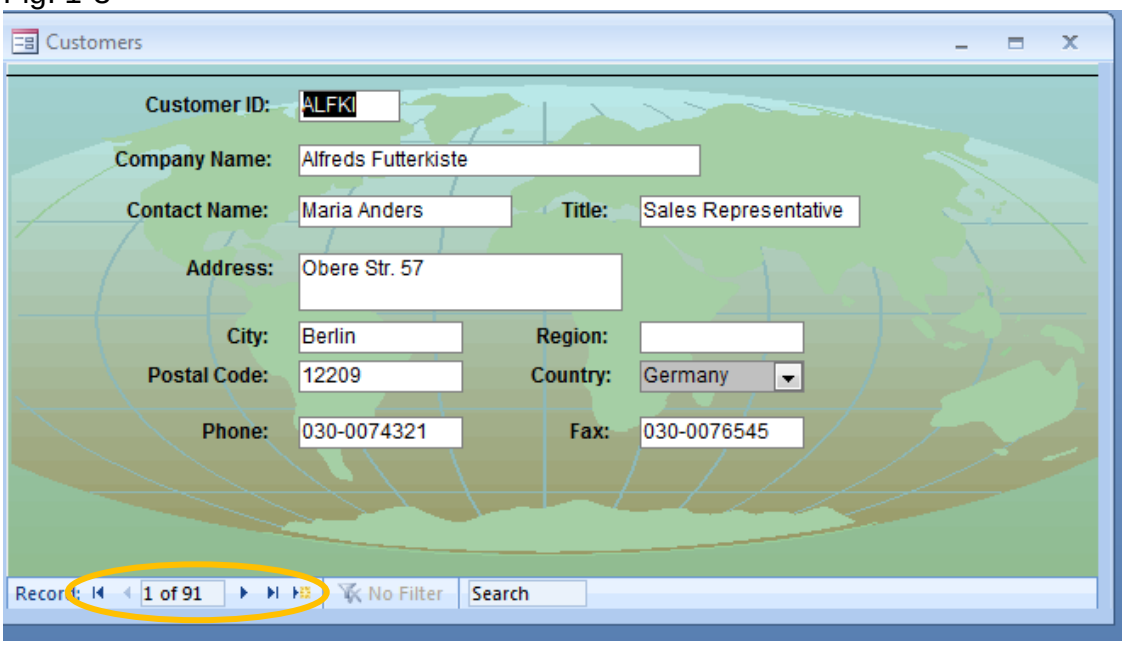
2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Flavor 1.1 Record Navigation and Record Operation table of contents

You will create a set of command buttons to navigate between records on a form. As a default, a record navigator, as shown in the Fig. 1-3, will be available on the bottom left of every form created from a table or a query. Due to its location on the form, most applications use command buttons instead. The alternative is to use a set of command buttons to replace the navigator.

Fig. 1-3



By doing this, you will make the set of commands more accessible to the user. Users do not need to know how to use the navigation buttons. After all, new users will not be as familiar with Microsoft Access as you are.

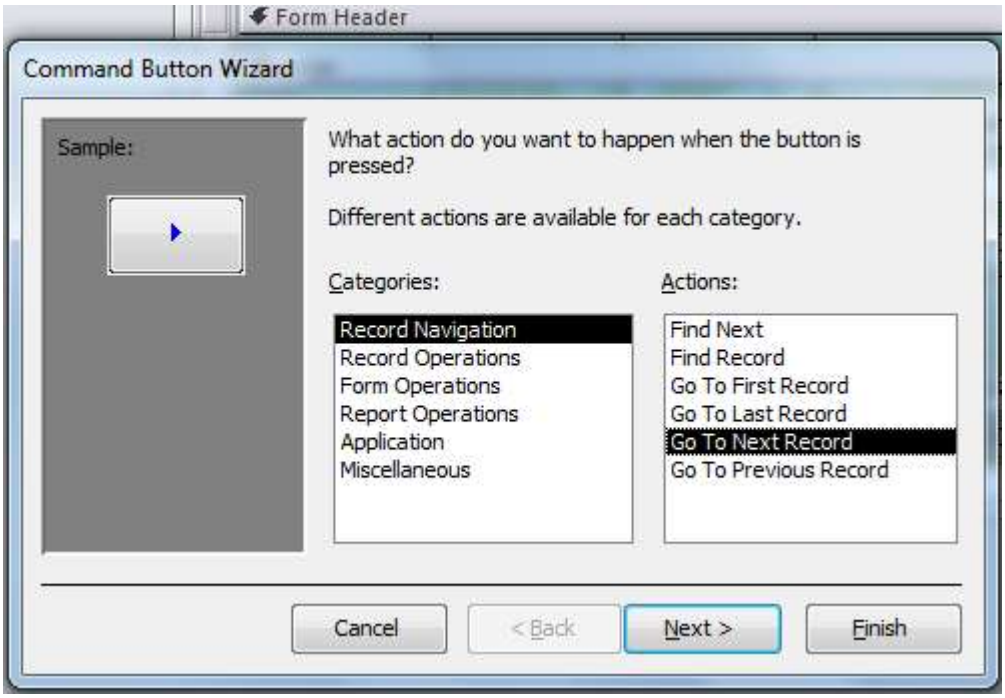
To bring commands to the forefront, go into the form's **Design View** and click the Ribbon **Design** in the **Controls** and select the **command** button. Put the command into one of the sections where the user can perform the actions: **Form Footer**, **Form Header**, or **Detail**. The **Command Button Wizard** makes the button work automatically by writing a Visual Basic code for you.

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

The detailed steps to complete such a task are:

- Select the **Record Navigation Button**
- Select the items in the sub-category, such as **Go to Next Record** as shown in Fig.1-4.



Click NEXT, and choose either **TEXT** or **PICTURE** (see Fig.1-5). Make sure all the buttons you choose are consistent in characteristics like size, shape, or appearance.

Fig. 1-5

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

- Click NEXT. Give a proper name to this button, as shown in Fig.1-6. You can see the VBA code created by this wizard in the VBA Editor (discussed in the next chapter). Click FINISH to close this Wizard.

You can change the text content of the Tooltip by opening each button's property and changing the text content in **ControlTip Text**, as shown in Fig. 1-7.

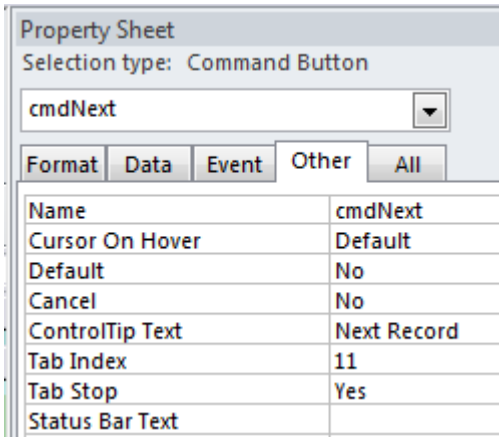


Fig. 1-7

You will see the revised content of the Tooltip (refer to Fig.1-8) when the mouse moves over the button.

The first button is done. Follow the same steps to complete the rest of the buttons:

Record Navigation category
- **Go To First Record**
- **Go To Last Record**
- **Go To Previous Record**

Record Operation category
- **Add New Record**
- **Delete Record**
- **Print Record**
- **Save Record**
- **Undo Record**

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Figure 1-9 shows a completed set of custom navigators with name labels. Placing a name label next to each command button is up to the individual design style. It doesn't matter if you add the labels or not, as long as users can easily understand how to use these buttons.

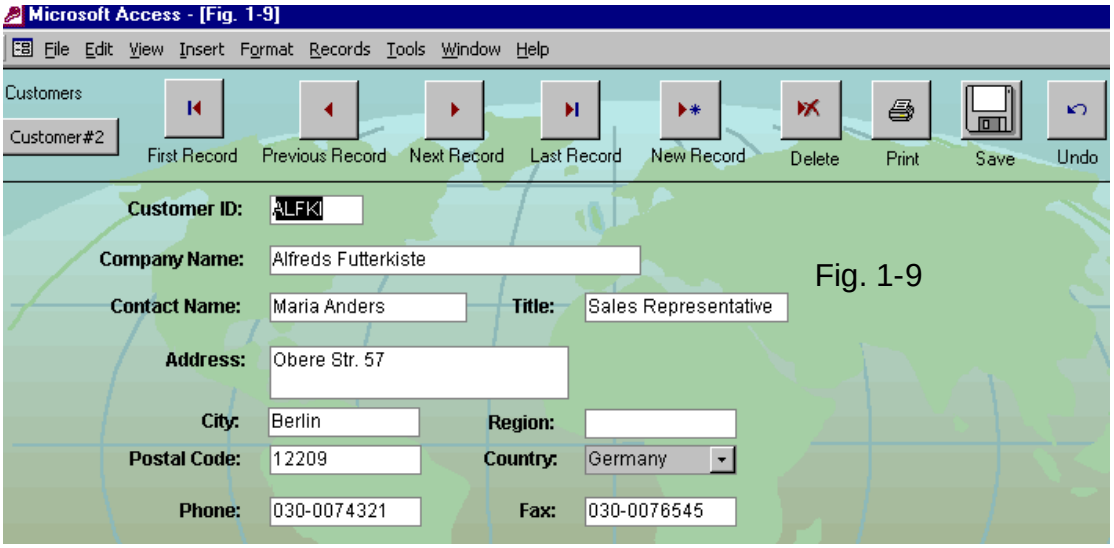


Fig. 1-9

After adding all the command buttons onto the form, you will not need the default record navigator. In order to remove this navigator, open the form property and change the navigator button from **YES** to **NO** (see Fig. 1-10).

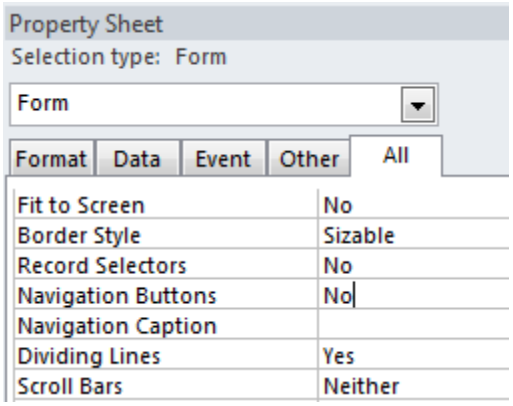


Fig. 1-10

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

After you've completed both categories, **Record Navigation** and **Record Operation**, the form should be the same as the one in Fig. 1-11.

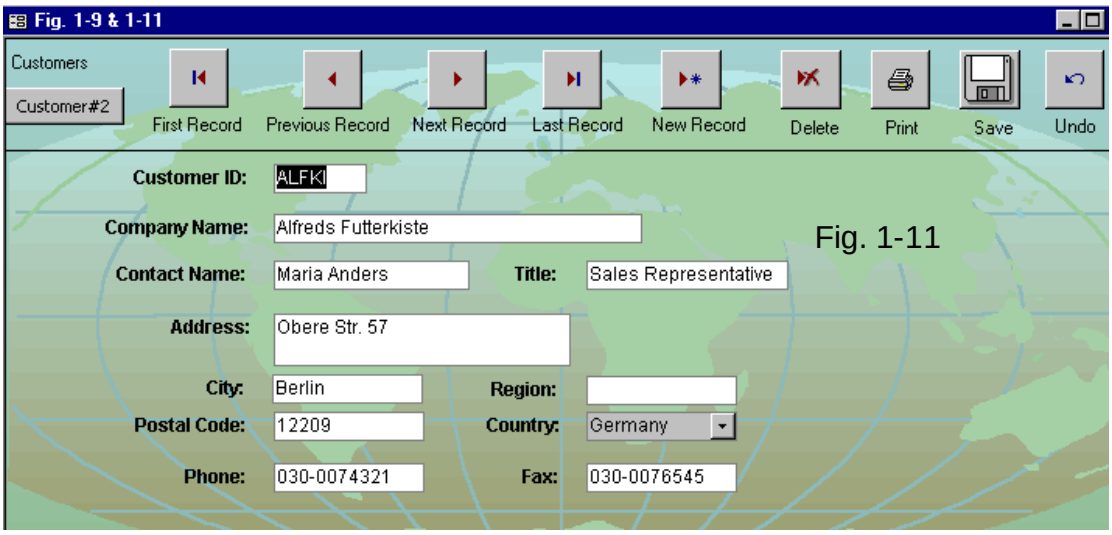


Fig. 1-11

Flavor 1.2 Form Operations

table of contents

Now let's move on to another category—**Form Operation**. Here's a simple exercise to familiarize yourself with this feature. First, create a new form that has no data source, just like the one in Fig. 1-12, by choosing a Blank Form from Create Ribbon.

A blank new form is created. Make sure the **Toolbox** toolbar is available. If it is not shown on the screen, you can find it using the instructions at the beginning of this chapter.

Follow the same steps from the previous example to add a command button after activating the **Control Wizard**. Then select the **Form Operation** from the **Categories** column and the **Open Form** from the **Actions** column, as shown in Fig. 1-13. Click NEXT.

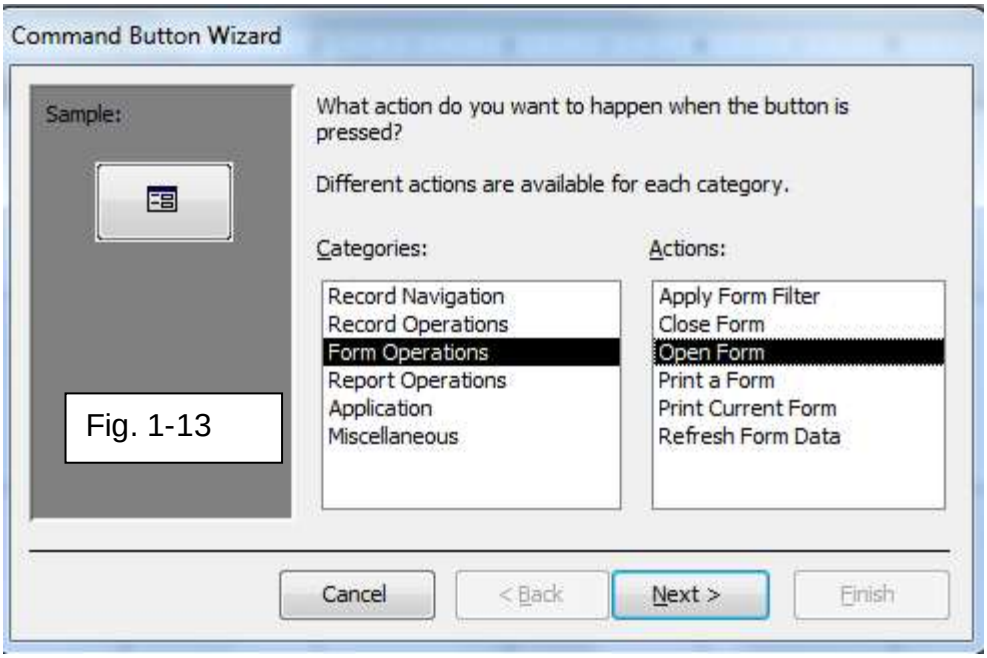


Fig. 1-13

The dialog Windows will then display all available forms the command can open. Select the **Customer Orders** as shown in Fig. 1-14. Click NEXT. Then leave the default selection "**Open the form and show all the records**" blank. Click the NEXT button again.

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Fig. 1-14

In the window, as depicted in Fig. 1-15, select the **TEXT** option and type "**View Order Information**" into the box. Click NEXT.

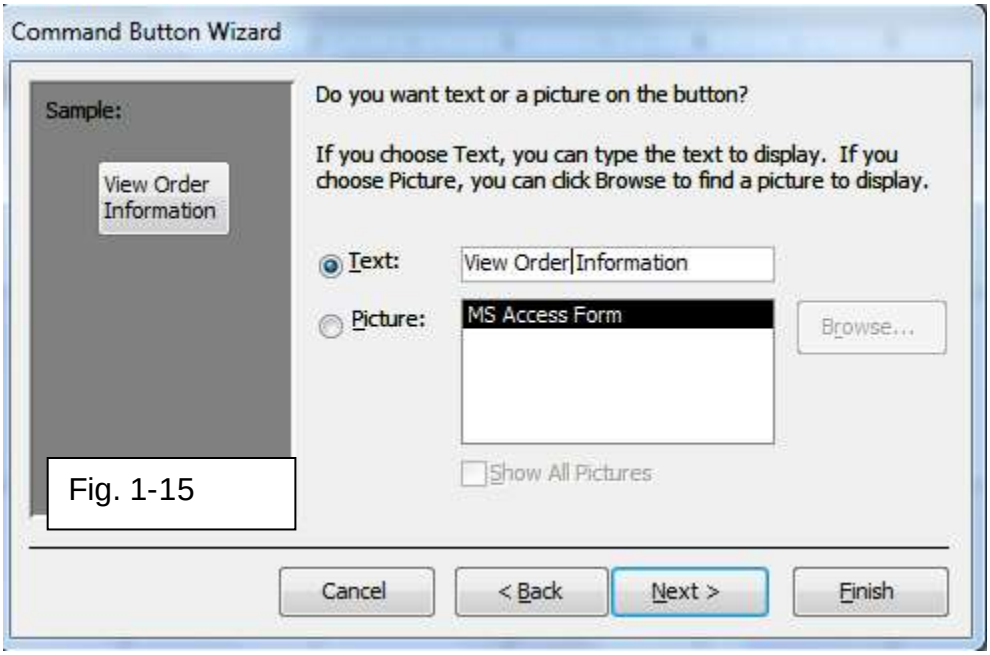


Fig. 1-15

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

In the next window, give a name to this button. Use the prefix "**cmd**" to represent the type of object, followed by the action name that can be executed by the command, which, in this case, is viewing order information on the order form. The name of the command in Fig. 1-16 is **cmdViewOrder**. It is best to give names as short as possible to objects and take advantage of abbreviations. Click FINISH.

Fig. 1-16

Sometimes, during the Wizard, you may decide not to continue and click Cancel. Then you will find the object left on the form, see Fig.1-17. How do you remove the unwanted object, which, in this case, is a button? Left-click to select the button image, as shown in Fig.1-17, and press the delete key **Del** on the keyboard to remove this button object.

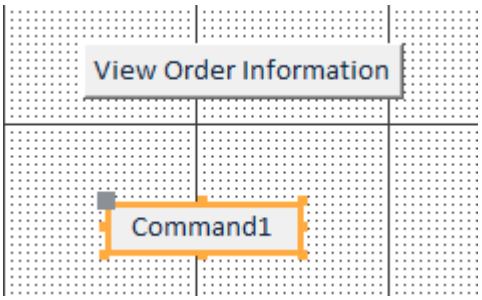


Fig. 1-17

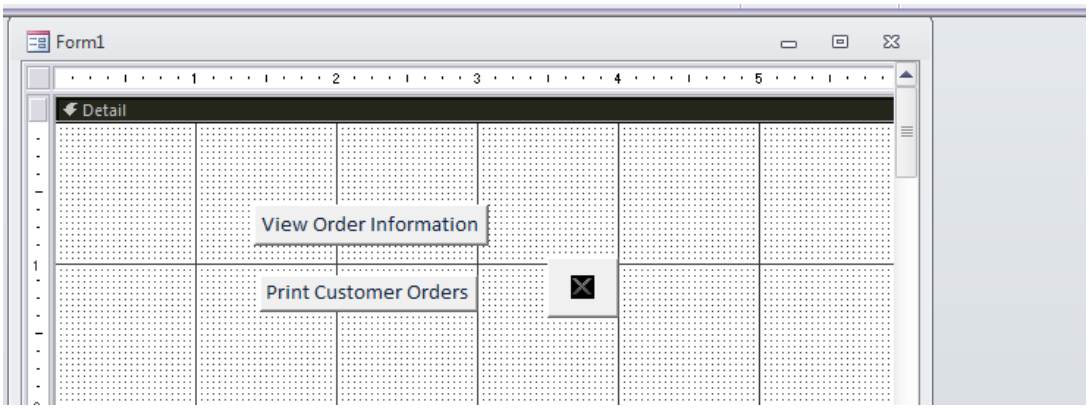
Save the form under the name of **frmFormOperation1**, then continue with the action choices from the **Form Operation** category.

Actions	**Object Text or Picture**	**Object Name**
Print a form	Print Customer Order	cmdPrintOrder
Close form	Exit picture	cmdClose

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

The form then looks like Fig. 1-18a. There are some items on the form we can improve, such as the record navigator, the form caption, and the "X" close button on the upper right corner. Here is how you can change the above items on the form property.



The final form will look like Fig. 1-18b.

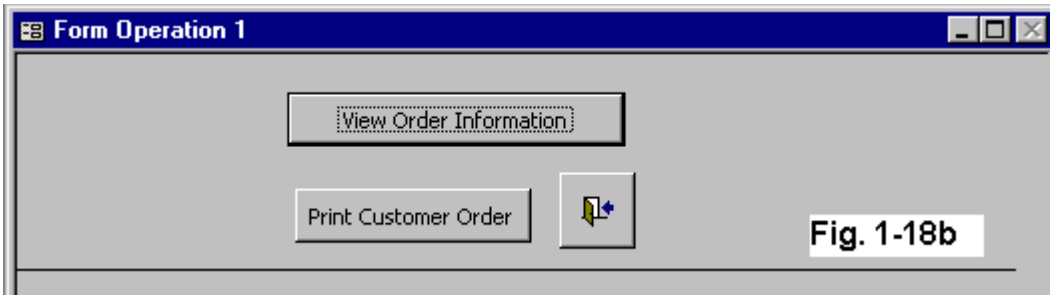


Fig. 1-18b

Switch from the **Form View** to **Design View**, then open the **Form Property** and select the **ALL** tab to view all properties. Enter the caption, **Form Operation 1** and change from **YES** to **NO** in **Record Selectors**, **Navigation Buttons**, and **Close Button**. See Fig. 1-19.

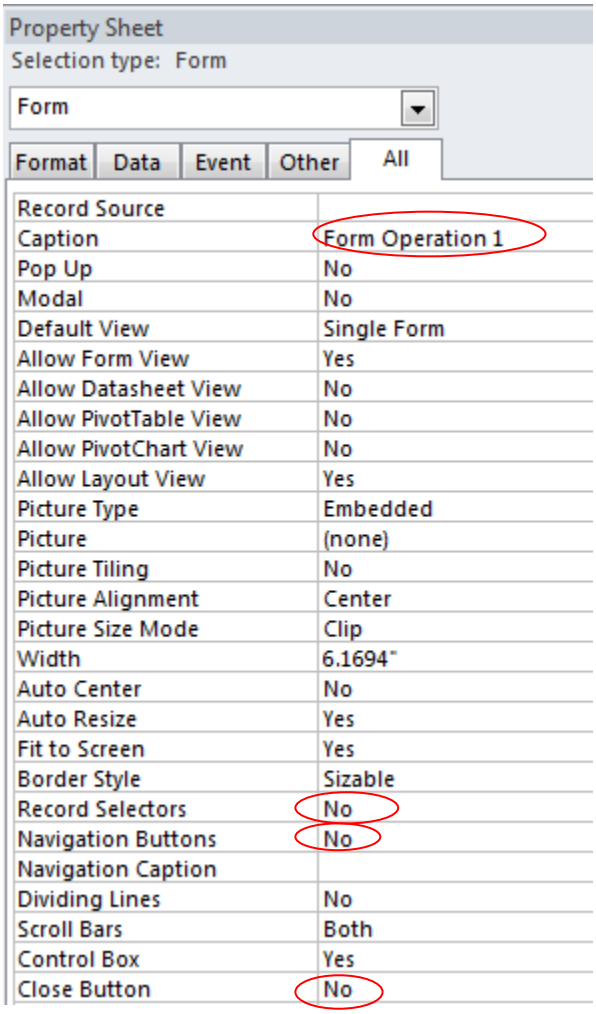


Fig. 1-19

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Flavor 1.3 Report Operations table of contents

You would like to send a summary report of quarterly sales, **Summary of Sales by Quarter**, via e-mail to somebody. However, you cannot just send the report, which is an object in the Access database. Instead, you will need to capture a snapshot of the report and send it as an attachment. How do you create such a snapshot without use of any coding in Access? Once again, your very own Houdini, the **Control Wizard**, is here to help.

Create a new form named **frm1_3RptOperation**, and add a command button with the **Control Wizard** activated. Select the **Report Operation** in the **Categories** column and **Mail Report** in the **Actions** column in the wizard, as shown in Fig. 1-20.

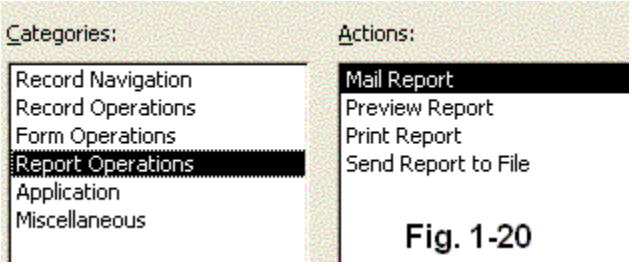


Fig. 1-20

Click NEXT. Select the report you want to send via e-mail. In this case, it is the **Summary of Sales by Quarter** (see Fig. 1-20a). Click NEXT.

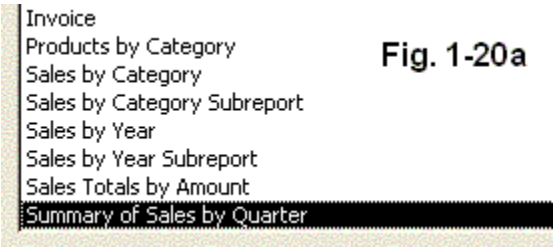


Fig. 1-20a

Select the **Text** option button and type "**Mail Quarterly Sales Report**" as indicated in Fig.1-20b. Click NEXT and enter the command name, **cmdQSalesRpt**. Click FINISH to end the Wizard.

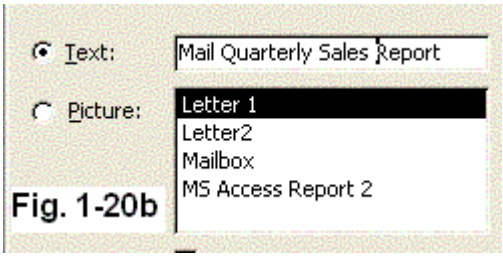


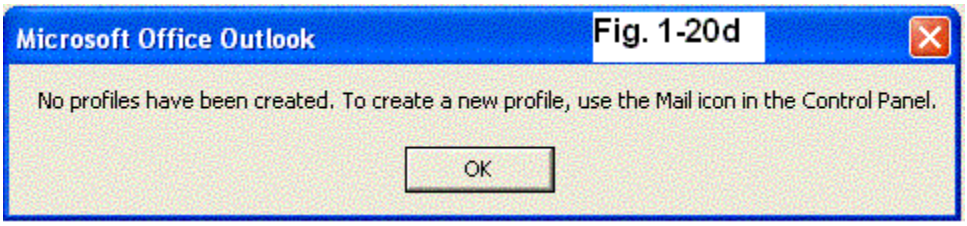
Fig. 1-20b

2006-2009 © All Rights Reserved

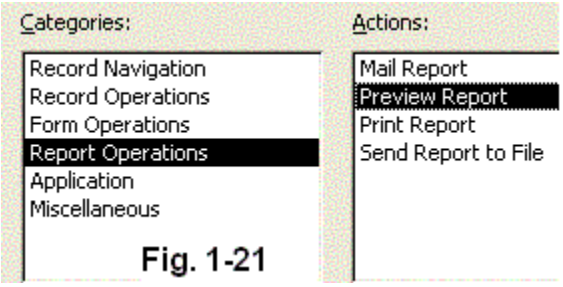
ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Now let's try it to see if the command works. Click the **Mail Quarterly Sales Report** button on the form **frm1_3RptOperation**. A dialog window appears, like the one in Fig. 1-20c. Select the **Snapshot Format**, as it is the format you want to send the report in. Click OK.

If your mail service software is ready, you will be presented with an e-mail editing window, or you will get a message as shown in Fig.1-20d to indicate there is no e-mail service ready for use.



The next operation is to preview a report. Use the same form, **frm1_3RptOperation,** to add a new command button. Select the **Preview Report** in the **Actions** column as shown in Fig. 1-21. Click NEXT.



Click the **Text** option button and type in "**Preview Quarterly Sales Report**" as illustrated in Fig. 1-21a. Click **Next**.

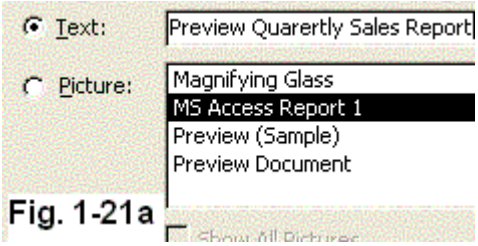


Fig. 1-21a

Enter **cmdViewQSalesRpt** as the name of the command. Click FINISH. Switch to **Form View** and click the **Preview Quarterly Sales Report** button to test it. A report print preview is shown in Fig. 1-21b.

Summary of Sales by Quarter

16-Jan-2012

Quarter: 1

Year:	*Orders Shipped:*	*Sales:*
1997	92	$143,703
1998	178	$276,330

Quarter: 2

Year:	*Orders Shipped:*	*Sales:*
1997	92	$145,655
1998	90	$161,362

Fig. 1-21b

The **Print Report** action is the same as **Preview Report**, except the former will not display a preview but will instead go ahead and print the report. You can decide which action your design needs and choose the appropriate one.

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

The last one in the **Actions** column is the **Send report to file**. If you need an electronic format of this report, then you select this one. The procedure to create it is the same as the one used for previous actions. **Output to MS Word** is selected as shown in Fig. 1-21c. Click next and finish.

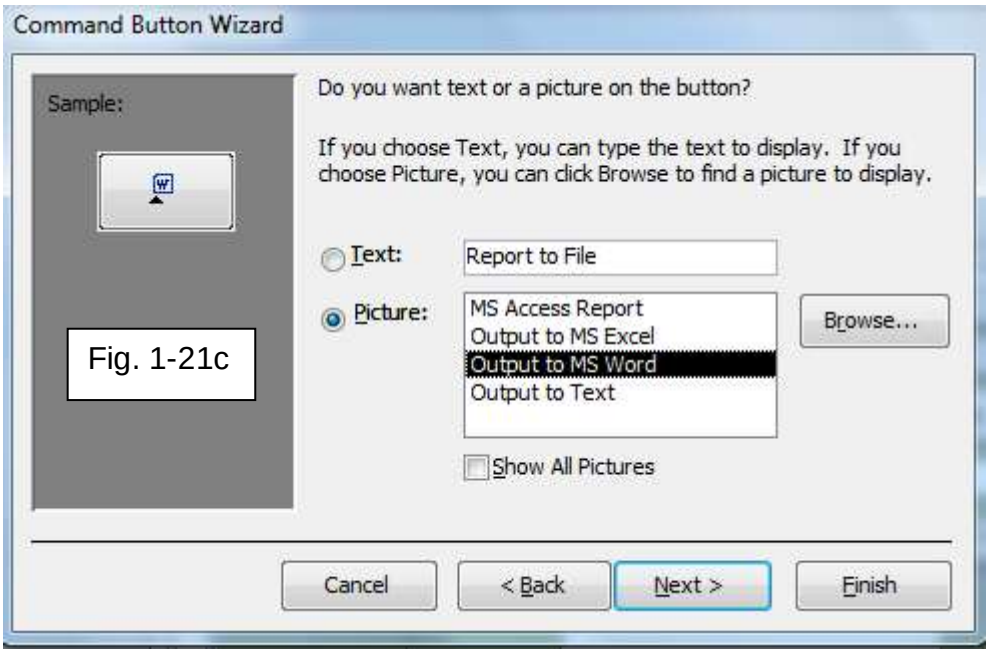


Fig. 1-21c

On the form, click the newest built button to open the report. Fig. 1-21d offers the output format selection and RTF is selected.

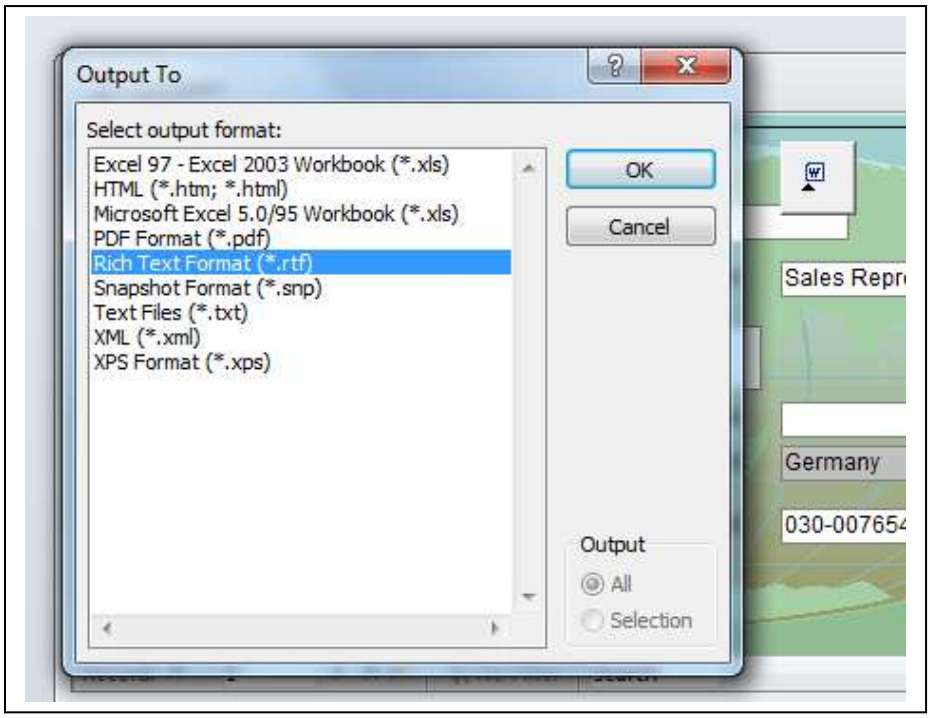


Fig. 1-21d

Flavor 1.4 Application Category table of contents

Sat you want to open Excel or Word from an Access database. You can simply take the actions in the **Application** category of the **Control Wizard** without any coding.

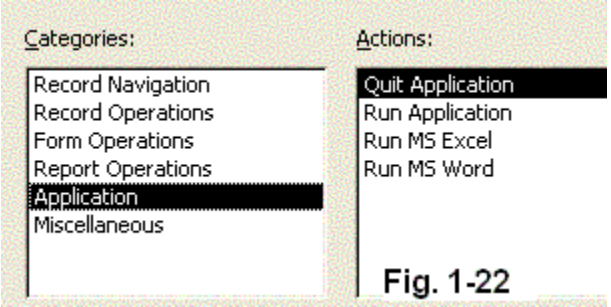


Fig. 1-22

Make a new form, **frm1_4AppOperation**, and add a new command button with the **Quit Application** selected in the **Actions** column as displayed in Fig. 1-22. Click NEXT. Click the **Picture** option button and pick **Stop Sign** and you will see the sample image in the left panel. Click NEXT. Enter the command name **cmdExitApp**. Click FINISH to end the wizard.

Fig. 1-22a

Continue to add two more new command buttons, one to open Excel and the other to open Word. They are straightforward, without demanding any special attention. In order for the **Run Application** action to work, you need to know where a specific application is located, particularly the application executable file (i.e. exe file).

You can turn the form to **Form View** and test these buttons. They should open Excel or Word when you click them. The **Quit Application** button (with the Stop Sign image) will close your database and Access program at the same time.

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

Flavor 1.5 Miscellaneous Categories table of contents

The last item in the categories column is the **Miscellaneous**. Create a new form named **frm1_5MisOperation**, and put a **Print Table** command button (see Fig. 1-23). Click **NEXT**. You can select any table you want to print. For this exercise, pick the table **Categories** to print. Click **NEXT**.

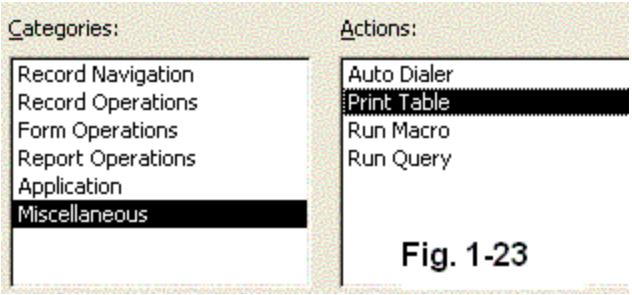


Fig. 1-23

Select the **Text** option button and enter the text **Print Table Categories** like in Fig.1-23a.

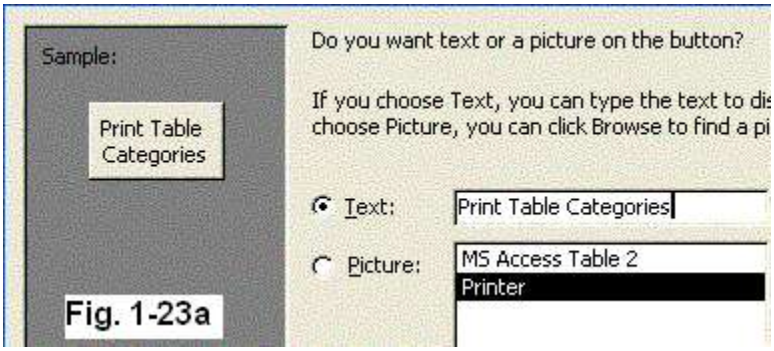


Fig. 1-23a

Click NEXT. Enter the command name **cmdPrintCateTbl**. Click FINISH.
Add another new command button with the action of **Run Query** selected. Click NEXT**.** In the next window, pick one of the available queries you want to run. Here, refer to Fig. 1-23b and choose the **Current Product List** query. Click NEXT**.** Select the **Text** option button and type **Run Current Product List Query** into the box. Click NEXT and enter the command name **cmdQryProduct**. Click FINISH.

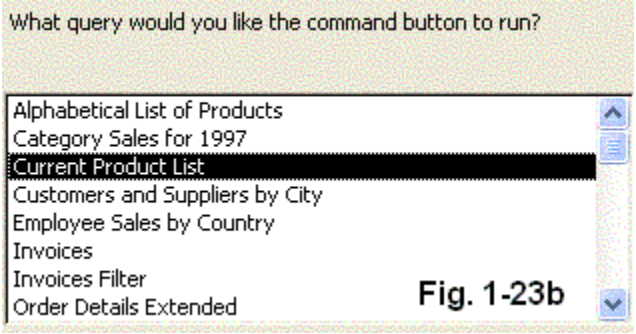


Fig. 1-23b

2006-2009 © All Rights Reserved
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes, 2nd edition

If you want to run a Macro just by clicking a button, then here is the way to make that happen. Add a new command button on the form and select the **Run Macro** in the **Actions** column of the **Miscellaneous** category. Choose one of the macros, **mcrDemo1** for this example, to run. Select the **Text** option and enter the text **Run Macro Demo 1**. Click NEXT. Enter the command button name **cmdRunMcrDemo1**. Click FINISH.

Flavor 1.6 What's Happening in the Kitchen?

table of contents

Behind the Scenes: What did the wizard do to make the button perform as you asked?
Here's how it works: Whenever you perform any action on a form—click a button or move the cursor from one field to the other, for example—an **event** (**On Click** for instance) occurs. Every event is a chance for the application to perform certain actions, such as running a Macro to open a form or a Visual Basic procedure to print a report.

When the **Command Button Wizard** creates the button, Access automatically generates codes in a VBA procedure according to the event and attaches this **event procedure** to the button's **On Click** event. This technical concept is very similar to the **Macro Recorder** feature found in Excel. So, every time the button is clicked or any event occurs, the Visual Basic code in the event procedure is triggered and runs, making Microsoft Access perform the desired action.

It is not a bad idea for users to learn how to code VBA procedure by viewing the procedure generated by the Wizard. As you add the controls to the form using the **Control Wizard**, you can view the codes written by Access. Let's pick one of the commands you have made: **cmdViewOrder** on the form **FormOperation1** in Flavor 1.2.

There is no need to open the form **FormOperation1** in **Design view** or **Form view.** Select **Code** on the **View** menu or on toolbar as shown in Fig. 1-24.

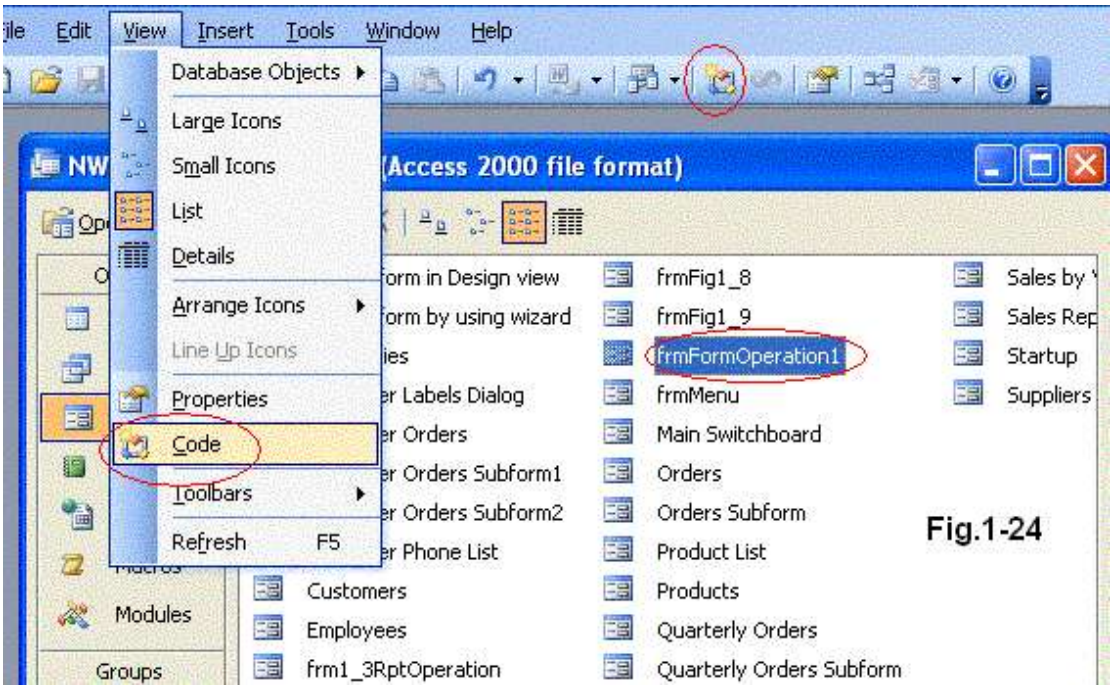


Fig.1-24

The Visual Basic Editor appears and displays the objects, properties, and procedure windows (see Fig. 1-25). When an object in the left panel changes, the procedures associated with the specific object will change in the right panel. Between the Private Sub line and the End Sub line, there are VBA codes for specific actions. Reading the procedures behind commands is a quick way to learn and understand VBA coding. More on VBA will be discussed in Dish 5.

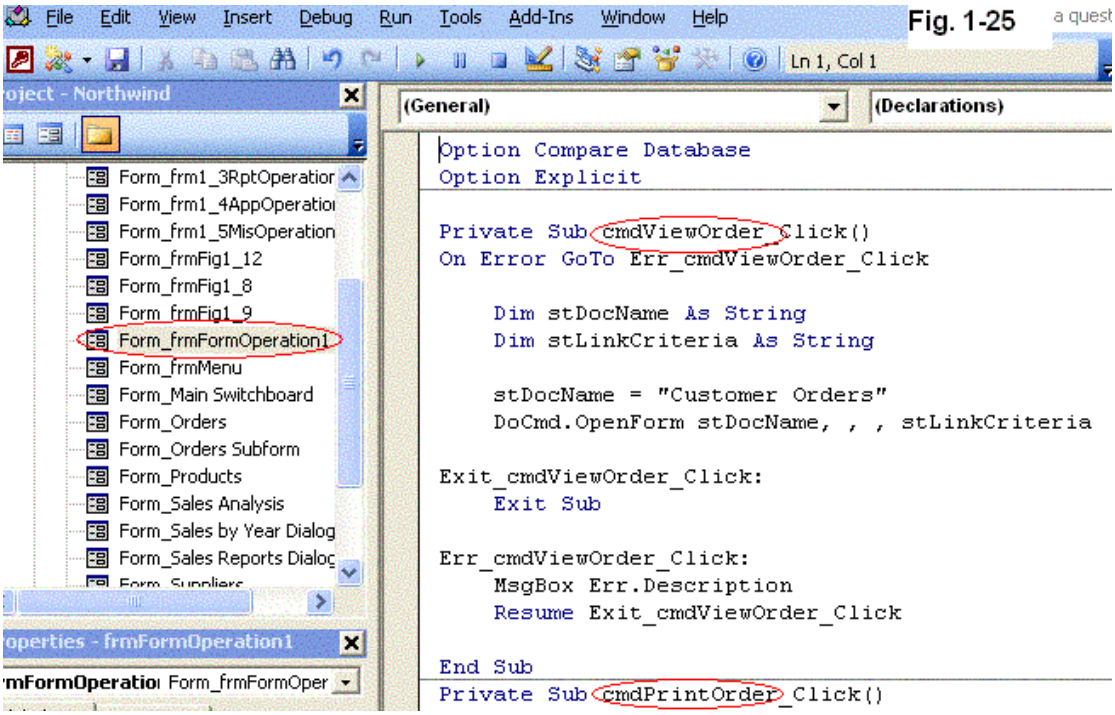


Fig. 1-25

Chapter 2 Macro Builder and Group Macros

table of contents

In Access, there are three builders: **Expression Builder**, **Macro Builder**, and **Code Builder**. **Expression Builder** should be the first builder introduced and used by most novice Access users, so will not be covered. In this chapter, the **Macro Builder** will be discussed**.** The **Code Builder** can be found in Chapter 4.

One reason to use the **Macro Builder** is that it can create a macro or group macro quickly and easily. However, that is not the only way to create a macro or group macro. You can create a macro just like other objects (tables, forms, etc.) by clicking the **New** button in **Macro Object** window.

Before you get into the **Macro Builder**, let's take a look at the following scenarios.

Flavor 2.1 What if the Control Wizard is inactivated when a command button is created on a form?

table of contents

If a button is created on a form without activating the **Control Wizard**, this button is just an image and won't do anything when a user clicks it. You can change that by doing either one of the following:

If a macro is already created, such as **McrDemo1**, you can then connect the macro to the button. First, right-click on top of the command button and select **Properties** from the bottom of the short-cut menu, as shown in Fig.2-1. Second, select **McrDemo1** from the **On Click** event drop-down list in the **Event** tab of the command button property as depicted in Fig.2-1a. This button will then do exactly what the macro is designed for.

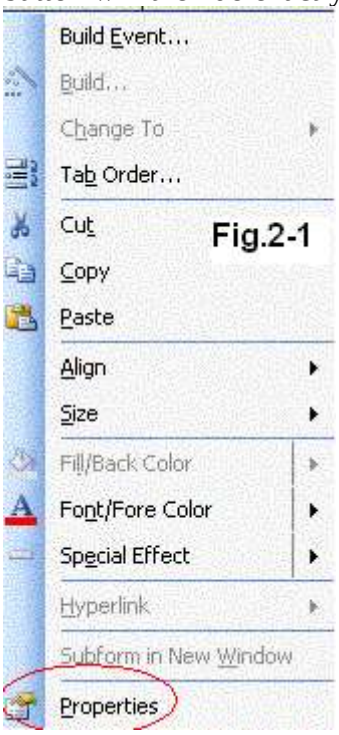


Fig.2-1

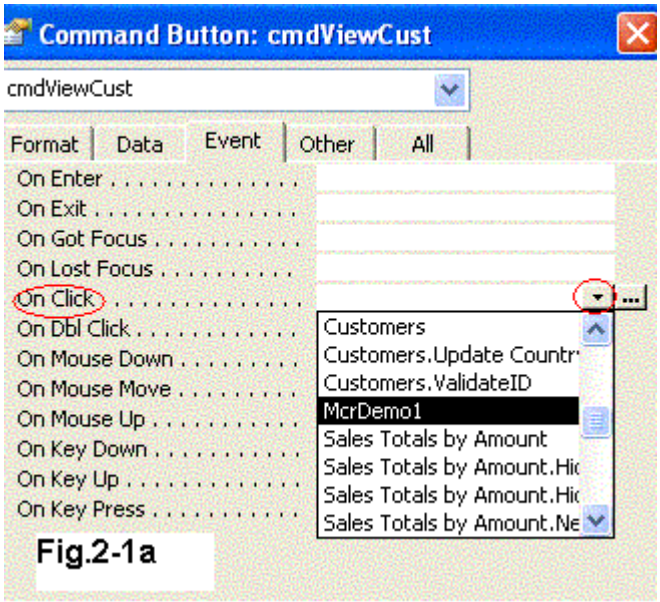
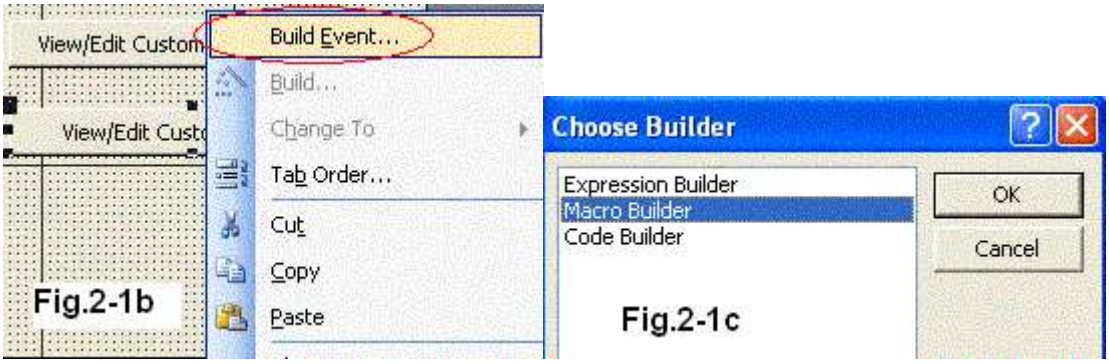


Fig.2-1a

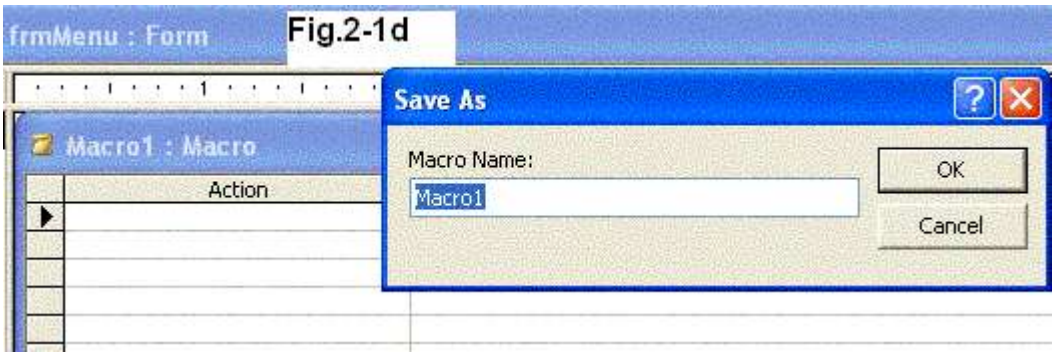
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

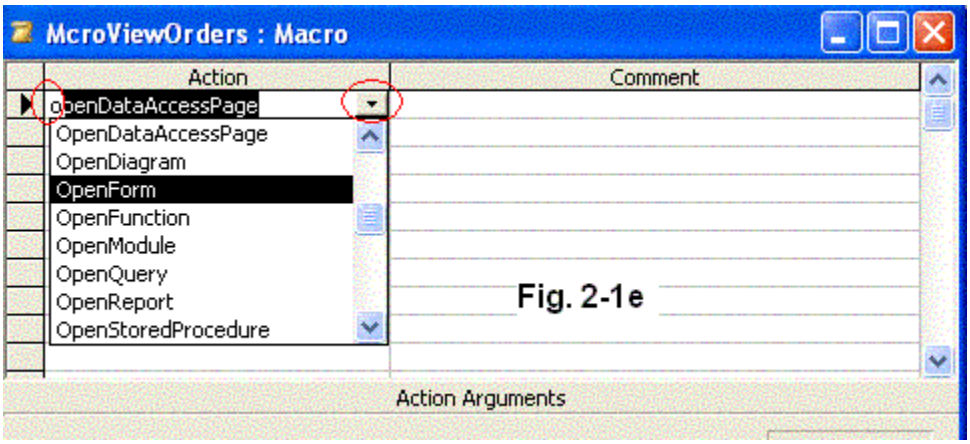
If no macro has been created in advance, you can click the button as indicated in the Fig.2-1b, then select **Build Event** on the pop-up menu. Fig.2-1c shows the **Builder** selection dialog. Select the **Macro Builder** to make a macro or group macro.



Click **OK**. You will be asked to enter the name of this new macro (refer to Fig. 2-1d). Enter **McrViewOrders** as the macro name. Click **OK**.



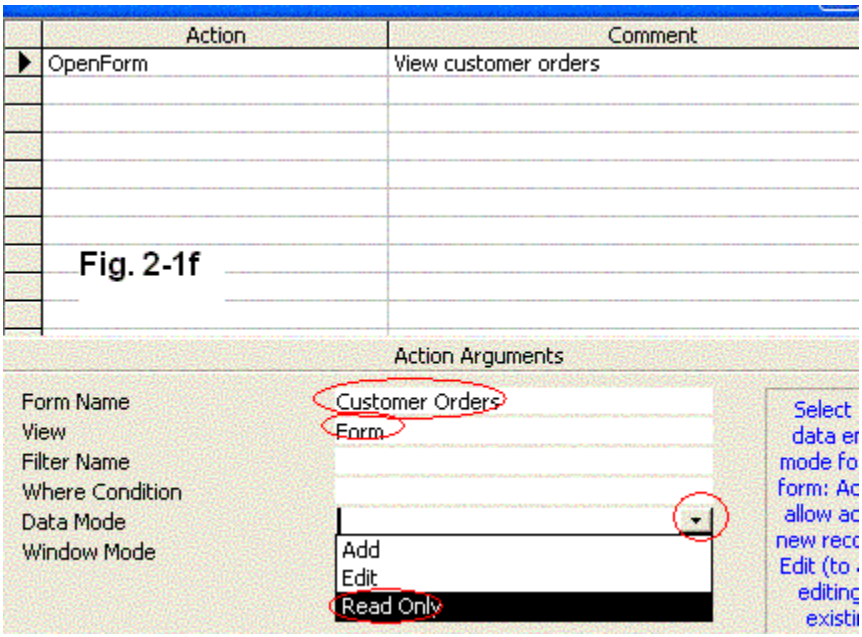
A new macro now is ready for you to select the actions. Click on the triangle on the right-hand side of the **Action** column to see the drop-down menu. You can type the first letter of action you want to select, such as O for **OpenForm** action, then the list box will show the first action that starts with the letter you entered, as shown in Fig. 2-1e.



2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

After the action is selected, you will see a list of available arguments in the **Action Arguments** area as illustrated in Fig. 2-1f.

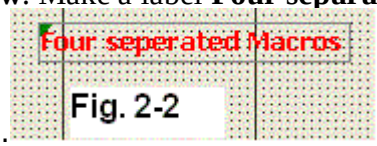


You may not need all the arguments to be filled. Yet, the **Form Name**, **View**, and perhaps **Data Mode** are required, or no object will be shown. When you click inside each box, the drop-down list will appear for you to select arguments from the list. You may close it as the macro is completed. As long as you can remember one of the above methods to create macros, it does not matter which one you use.

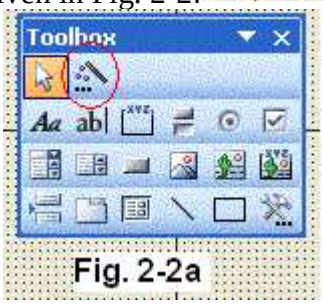
Flavor 2.2 Use Macro Builder to Build a Macro

table of contents

In the previous section, you have experienced two different ways to make a macro work. Here you will see how to make four simple macros work on the menu form. Open form **frmMenu** in **Design View**. Make a label **Four separated Macros** with font size 10 and red in color as given in Fig. 2-2.



Inactivate the Control Wizard as shown in Fig. 2-2a.



Add a command button control on the form. Right-click on the command button and select the **Build event** from the top of the pop-up menu (see Fig. 2-1b). A **Choose Builder**

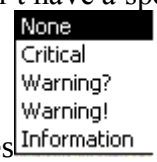
2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

dialog window appears, as shown in Fig. 2-1c. Select **Macro Builder** and click **OK** to close the dialog window. A **Save As** input box appears for you to enter the name for a macro, **mcrOpenProducts**, for example. Click **OK** and a blank macro window appears. Click inside the first action box. You will see drop-down list available for you to choose actions, listed alphabetically. Type the first letter "M" of **MsgBox** in the box and the list will change to display all action items starting with "M". Pick the **MsgBox** from the list.

Let's look at the environment of the macro window before you go any further. The first column is the **Action** column, which lists the order, from top to bottom, of command execution when a user executes this macro. The available commands, in alphabetical order, can be found by clicking the drop-down button as shown in Fig.2-1e. The second column is the **Comment** column, which is very similar to the **Description** column found in a **Table Design View**. It is better to write proper and short descriptions for each macro command in this column, such as its purpose or what will happen upon execution of this command. By doing so, you or the next programmer can understand this macro very easily even after much time has passed.

The section below the **Action** and **Comment** columns is for the needed Arguments of each command. The list of arguments will be different from action to action. The first action is to pop-up a message to indicate the next movement, which is to display product information, when users click **OK** in the message box. The **MsgBox** arguments are shown in Fig. 2-2b. You can enter any comment into the comment column next to the **MsgBox** Action. The text message entered in the **Message** argument does not need to have " " to enclose the text. The **Beep** is either **Yes** or **No**, depending on whether you want the message to make a sound when it pops up. Keep in mind that if users don't have a speaker, they will not hear the beep sound. The **Type** argument has four kinds of choices



. If you leave it as **None**, which is the default, or as **Information**, then the message box will only have an **OK** button. The **Title** text will appear on the title bar of message box when it shows.

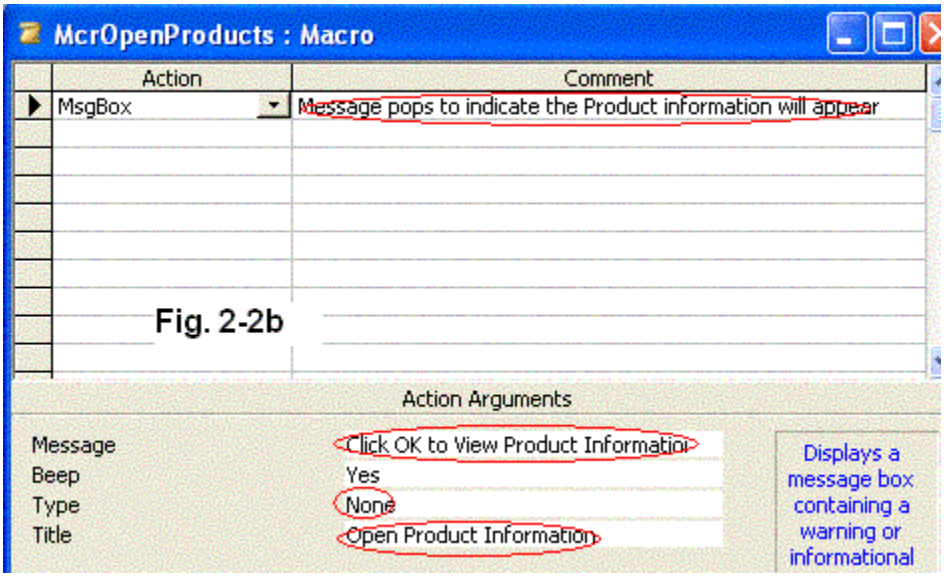


Fig. 2-2b

Move to the next row to select a different **Action**, filling in the **Comment** box along with the **Action Arguments**:

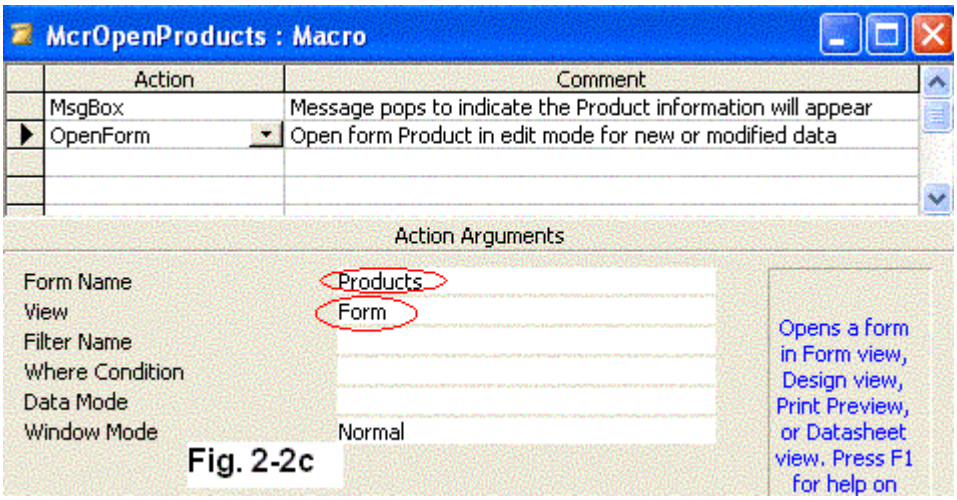


Fig. 2-2c

Close the macro and save it. The first macro is completed. Only four more to go!

Continue to construct more macros using the same procedure, each with the following requirements:

Macro Name: **McrViewCate**
Command button name: **cmdViewCate**
Command button caption: **View Category**
Action: MsgBox its arguments are as shown in Fig. 2-3

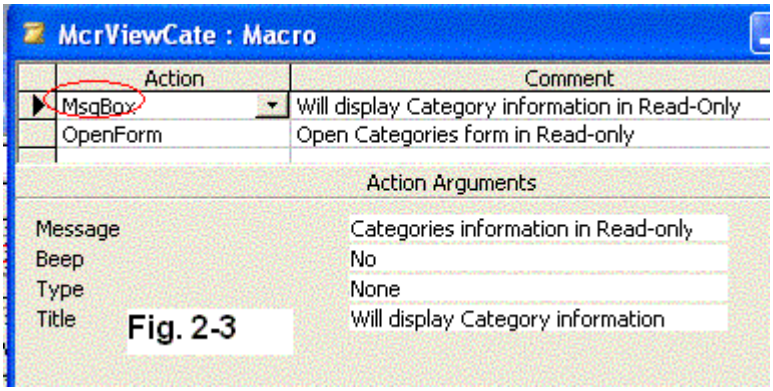


Fig. 2-3

Action: OpenForm its arguments are shown in Fig.2-3a.

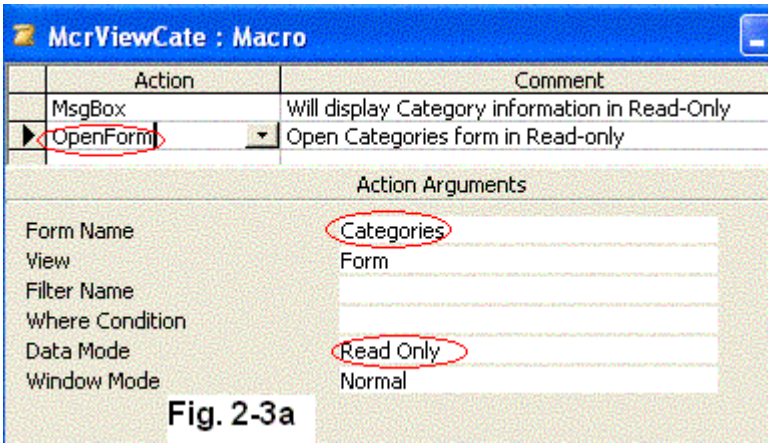


Fig. 2-3a

Macro Name: **McrNewCust**
Command button name: **cmdNewCust**
Command button caption: **Add New Customers**
Action: MsgBox its arguments are as shown in Fig. 2-4

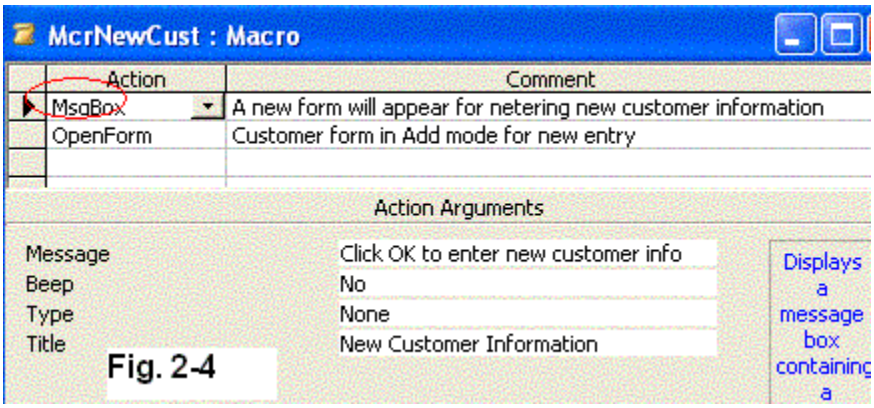


Fig. 2-4

Action: OpenForm its arguments are as shown in Fig.2-4a

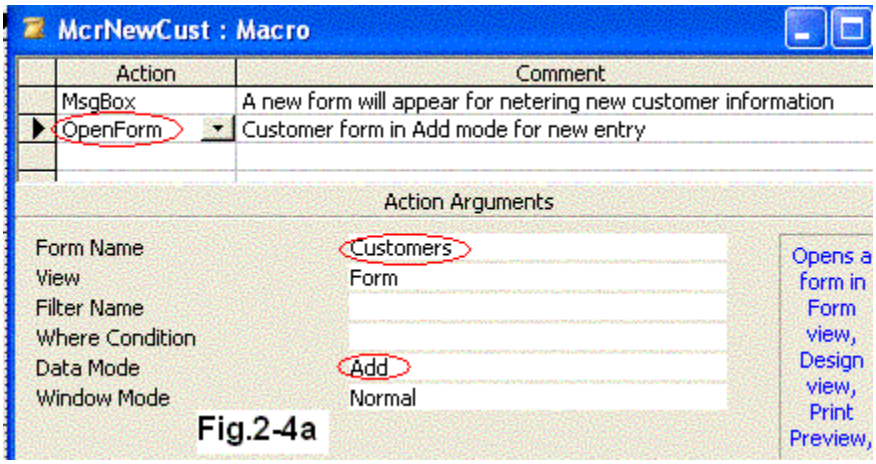


Fig.2-4a

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Macro Name: **McrCloseFrm**
Command button name: **cmdCloseFrm**
Command button caption: **Close this form**
Action: MsgBox its arguments are as shown in Fig. 2-5

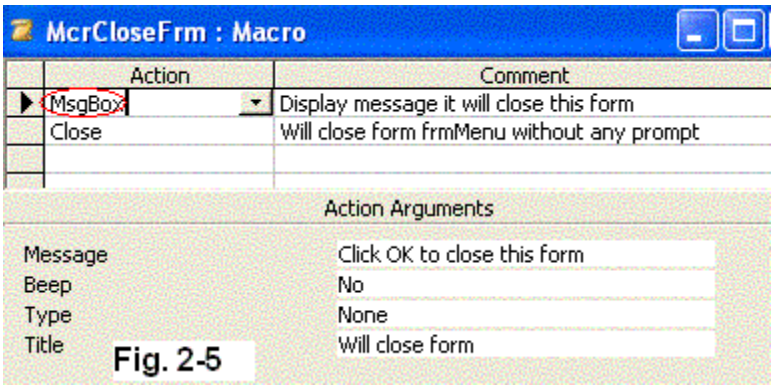


Fig. 2-5

Action: Close its arguments are as shown in Fig.2-5a.

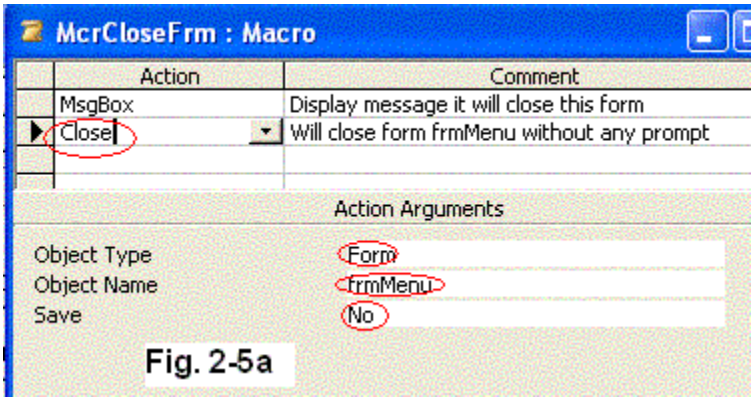


Fig. 2-5a

This **Close** command will close the object specified in the **Object Type** which should be matched with the name of object entered in the **Object Name** box or an error message will appear. Another action is **Quit** which is different from **Close**. Next macro is to use the Quit action to close the database application and Access program all at once.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Macro Name: **McrQuit**
Command button name: **cmdQuit**
Command button caption: **Qut**
Action: MsgBox its arguments are as shown in Fig. 2-6

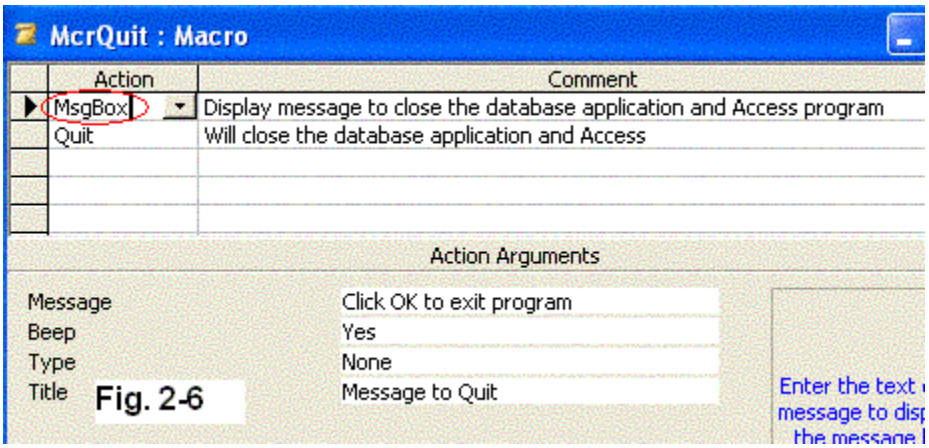


Fig. 2-6

Action: Quit its arguments are shown in Fig. 2-6a.

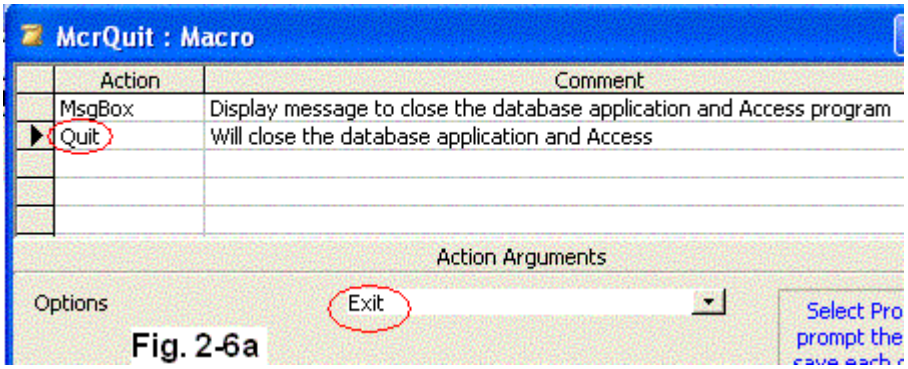


Fig. 2-6a

You need to adjust the size and alignment of the buttons to elevate the visual appeal and quality of the form. Here are some typical adjustments:

1. Make the width and height on all objects (i.e. button) the same
2. Align all five buttons.
3. Display five buttons an equal distance apart, vertically or horizontally.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Adjust the width and height:

Select all five buttons and right-click on any one of them to open the pop-up menu. If you would like to use the widest button as a model, among other possibilities, click **To Widest**, as shown in Fig. 2-7. You can also adjust the height by changing to **Size, To Tallest** on the pop-up menu.

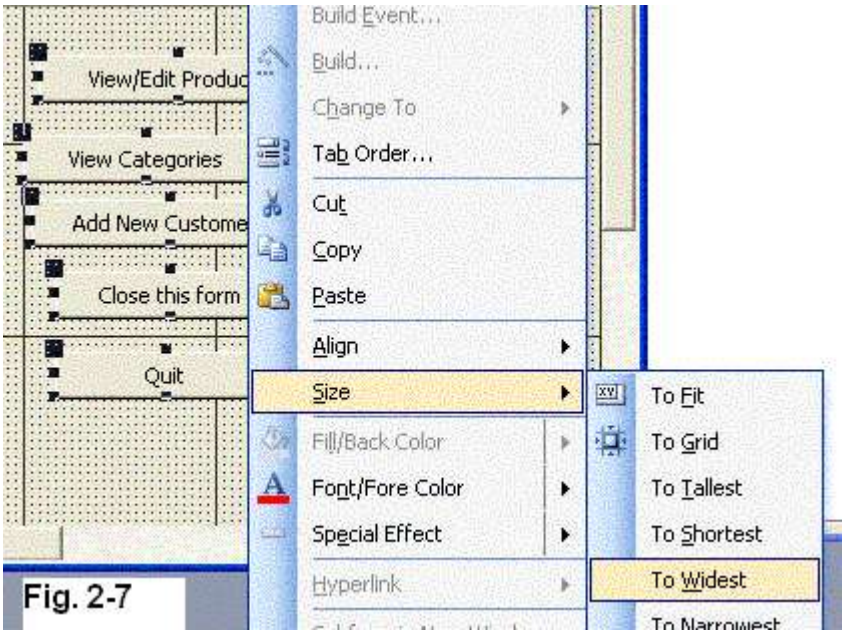


Fig. 2-7

Align the objects horizontally

Now, all five buttons are identical in size. However, on the form, they are not in order, in terms of horizontal position. You need to align them either to the left or right. For this example, as seen in Fig. 2-8, align these objects to the left where the far left object, button **View Category**, is located. All other five buttons will move to the position where the left border of the button **View Category** is located.

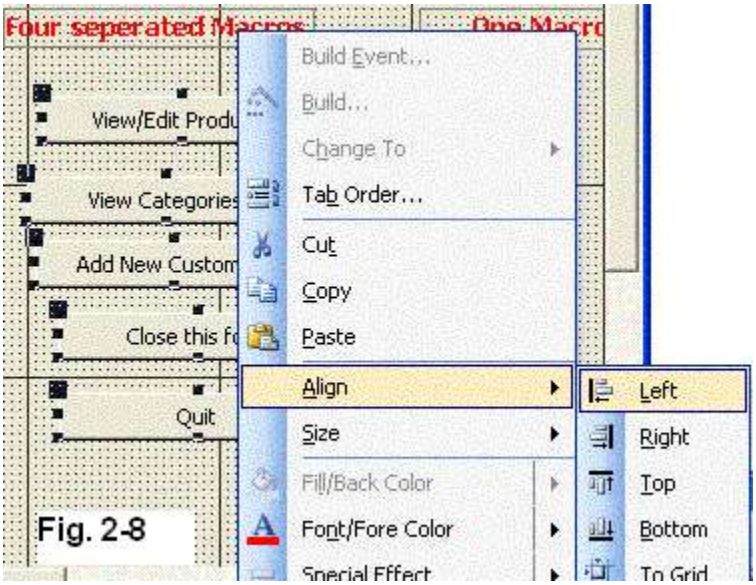


Fig. 2-8

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Equalize the space between objects

Like in Fig. 2-9, adjust the distance between objects by equally dividing the space between the top object and the bottom object in the group.

Fig. 2-9

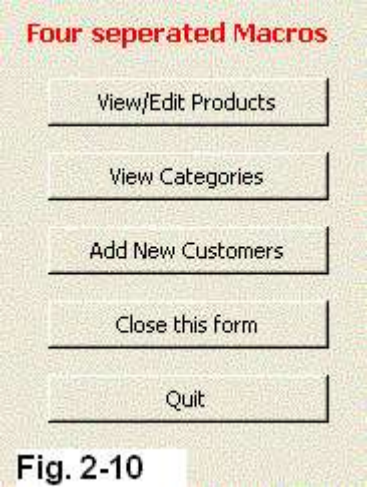


Fig. 2-10

After all the adjustments have been made, your form should look like Fig. 2-10.

Let's view what arguments are needed from the above design. Not all arguments should be filled; some of them can be filled by default if no item is selected. Remember that the following must be filled in the **OpenForm** action.

Object Name: During the execution of the macro, error messages will come up if this is left blank, a common mistake made by beginning users.

View: This argument will display the proper type of objects during execution. The **View** must be consistent with the **Object Name**, or else an error message will pop up when executing the macro.

Data Mode: If no **Data Mode** is selected, it will be the default mode, **Edit**. The **Edit** mode lets users delete, modify, and add records. The **Add** mode only allows the addition of new records. Users cannot view, delete, or modify any existing records in the **Add** mode. You can design one form and for different groups of users, each with different job responsibilities, by setting different data modes for each. The **Read Only** mode is straightforward.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Flavor 2.3 Use Macro Builder to Create a Group Macro table of contents

Creating a group macro is very similar to making a single macro, except you need to add one more column: **Macro Name**. You will then enter the name of each macro you make in the group to this column. The rest, such as the **Action** column, **Comment** column and the **Arguments**, are the same as creating a single macro.

You will make a group macro to include all five individual macros created in **Flavor 2.2** to make it easier to compare them. A group macro needs to be made before you add any command button to the form.

Click **New** to create a new macro in **Macros** objects. A new macro is open. Select **View**, **Macro Names** on the menu to add the **Macro Name** column as shown in Fig. 2-11. A group macro now is available with three columns, **Macro Name**, **Action**, and **Comment** as shown in Fig. 2-12.

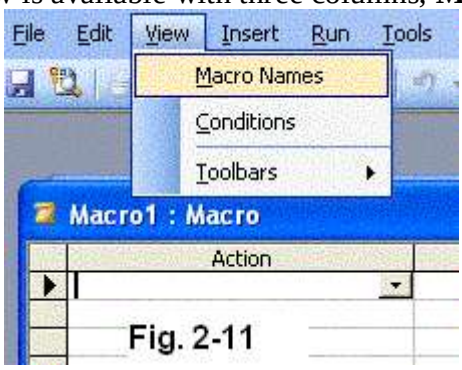


Fig. 2-11

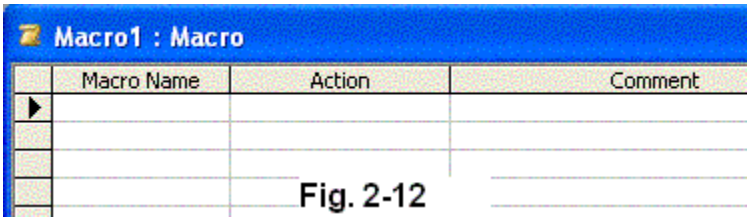


Fig. 2-12

Enter the **Macro Name**, **Actions**, and **Arguments** for all five macros, as used in Flavor 2.2. You can open the individual macro and group macro side-by-side to enter the information, as shown in Fig.2-13. Feel free to copy and paste the text. Continue to build the other four macros, **McrNewCust**, **McrViewCate**, **McrQuit**, and **McrOpenProducts**.

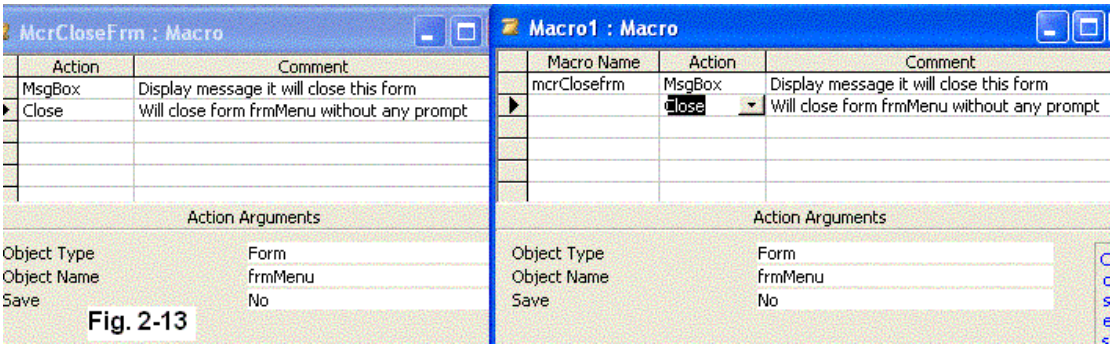


Fig. 2-13

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

The final group macro is shown in Fig. 2-14.

gMcrMenu : Macro **Fig. 2-14**

Macro Name	Action	Comment
mcrClosefrm	MsgBox	Display message it will close this form
	Close	Will close form frmMenu without any prompt
mcrNewCust	MsgBox	A new form will appear for netering new customer information
	OpenForn	Customer form in Add mode for new entry
mcrQuit	MsgBox	Display message to close the database application and Access program
	Quit	Will close the database application and Access
mcrOpenProduct	MsgBox	Message pops to indicate the Product information will appear
	OpenForn	
mcrViewCate	MsgBox	Will display Category information in Read-Only
	OpenForn	Open Categories form in Read-only

There is no need to repeat the **Macro Name** in a group macro. You only need to enter the name of each macro once in the **Macro Name** column. Also, there is no need to have any empty row between different macros in the **Action** column. The main reason for leaving a blank row between macros is to make it easier to read. Access can read and execute them either way with no difficulty or hint of any apparent difference. Save this group macro under the name, **gMcrMenu**, where "g" signifies a group macro.

You need to "connect" these macros in a group to the objects on the form, otherwise, no actions can be performed. First, open the form **frmMenu** in **Design View**. Make a label

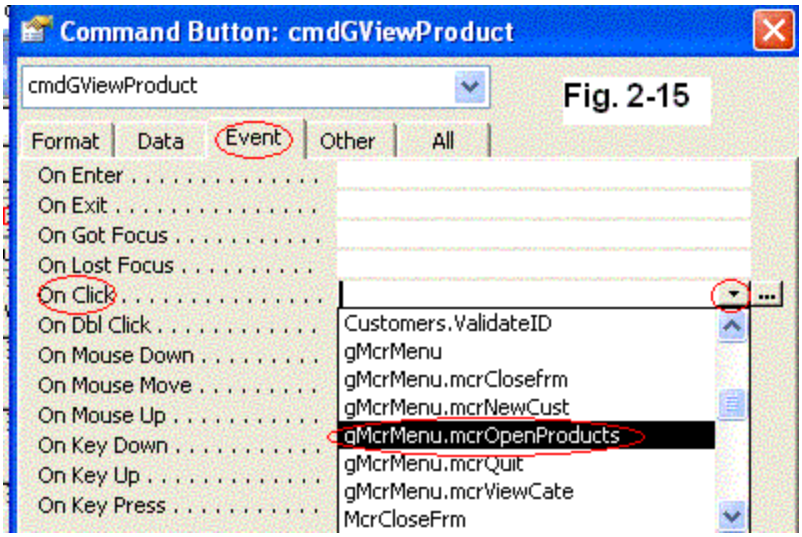
with 10-point size and in red. Add a new command button on form with the following settings:

Name: **cmdGOpenProduct**
Caption: **Open Products Information**

Click the E**vent** tab under **Property** of the command button. Click inside the **On Click** event box and select the macro **gMcrMenu.mcrOpenProducts** from the drop-down list, as shown in Fig. 2-15. The name of the macro shows the group macro name, **gMcrMenu**, and the individual macro name, **mcrOpenProducts**. This one macro is made by two names and separated by a dot.

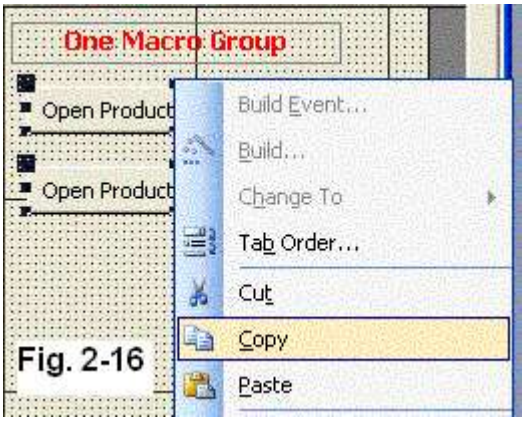
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition



You can follow the same steps to make the other four buttons. Instead of making these buttons four more times, you may want to find a quick way to do it using **Copy** and **Paste**. When the objects are done, you can just edit the properties of each button.

To quickly create the other buttons, select the two buttons and open the pop-up menu by right-clicking on top of the buttons. Select the **Copy** command from the pop-up menu as shown in Fig. 2-16.



When you paste them on the form, you will get two buttons. This is kind of like a "Buy one, get one free" deal. For many objects, this shortcut is ideal. After you paste them, the objects should be in the upper left hand corner in the **Detail** section. Move them to their proper places on the form. Make changes in properties, such as the name, caption, and **On Click** event macro. Double check the actions on all buttons to insure that they will perform what you designate them to.

Finally, you can test these two different macro designs. You will notice they are the same in terms of performance and actions. Why should you make a group macro if the individual macro can perform the same tasks? The main reason is because the group macro is easier to manage and maintain compared to individual macros. Remember back in Flavor 2.2, you had five separate macros, instead of just one group macro. Related individual macros can be put into

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

one group macro, often allowing more organization and easier maintenance. Decide which you are more comfortable with.

Flavor 2.4 Create a Custom Switchboard by Using Option Control table of contents

You can assign a macro to a command button without using VBA code. Here is an example of how to create a switchboard without using Switchboard Manager, which is one of the tools Access provides.

First, create all macros that you will need on both the main switchboard or sub-switchboards. Create a new form without selecting a table or query (leave the "Data Comes From" box blank) as shown in Fig. 2-17, or you can use **frmMenu**.

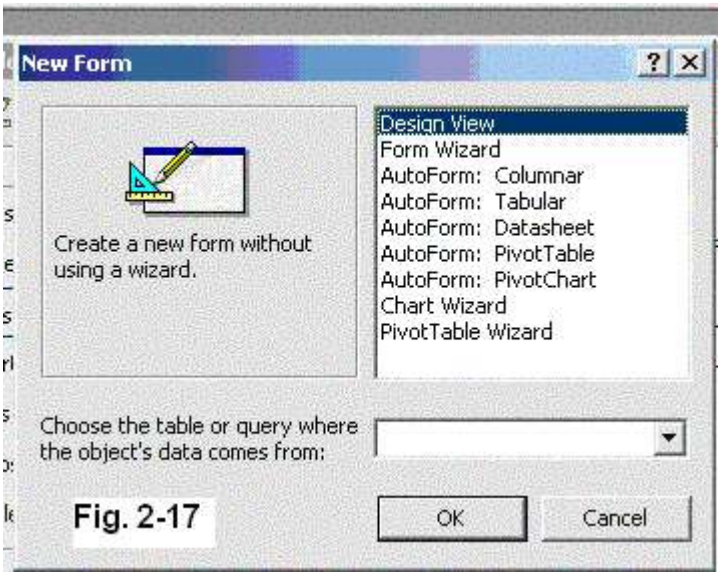


Fig. 2-17

Make sure **Control Wizard** is activated.

Click the **Option Group** as shown in Fig. 2-18.

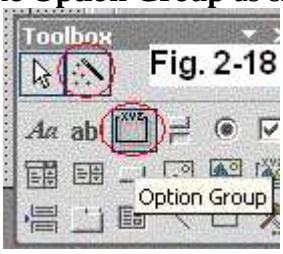


Fig. 2-18

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Follow the wizard and fill proper labels as Fig. 2-19 indicates:
View/Edit Customer Information
View/Edit Customer Orders
View/Edit Categories
Close

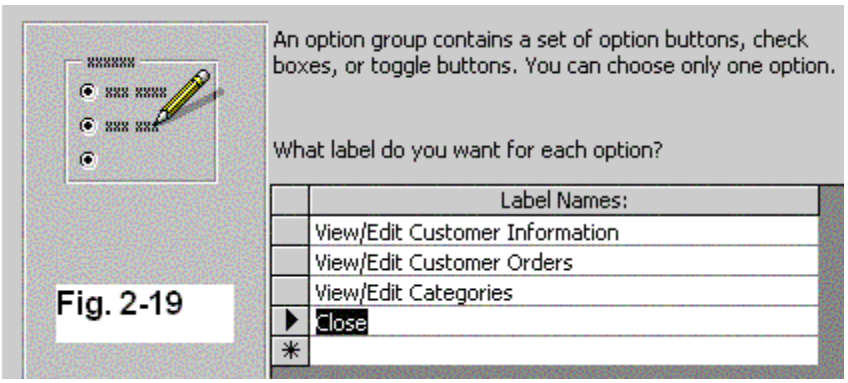


Fig. 2-19

Click NEXT.

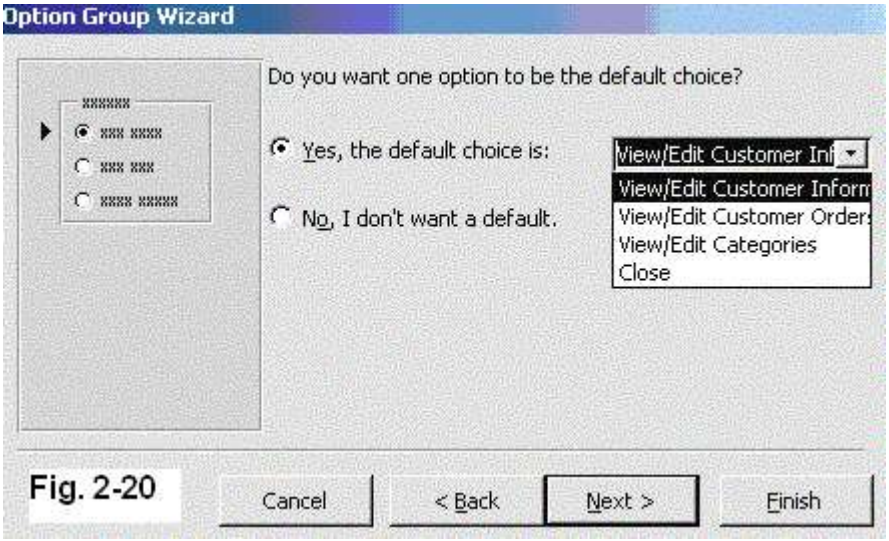


Fig. 2-20

If there are any options you would like to set as a default when the user opens this menu, here is the place to set it up. Select a choice from the drop-down list, as the Fig. 2-20 shows, or select the "**No**" option. Click NEXT**.**

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

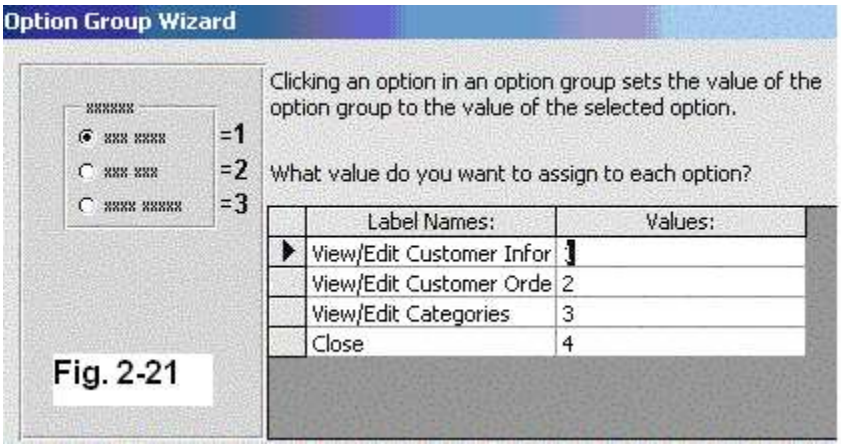


Fig. 2-21

Each option has an index value as shown in Fig. 2-21. The **View/Edit Customer Information**, for instance, has a value of one. You can use these values to control which command should be executed by using **Conditions** in **Macro**. Click NEXT.

This step, shown in Fig.2-22, provides some design choices for the type of controls shown on the form, with a preview of their labels in the left **Sample** section. **Option** buttons and **Toggle** buttons force the user to select one choice from an exclusive set. However, the **Check** buttons are different; the user can select one or more choices from this set of options. Even "None of the selection picked" is one of the possible choices! Yet, the **Check** buttons item is not a good choice if your intention is to force the user to make a decision by selecting one, and only one, choice.

The style of buttons will provide a different view of the options you design, as shown in the figure with **Toggle buttons** and **Raised style chosen in Sample section**. Click **NEXT**.

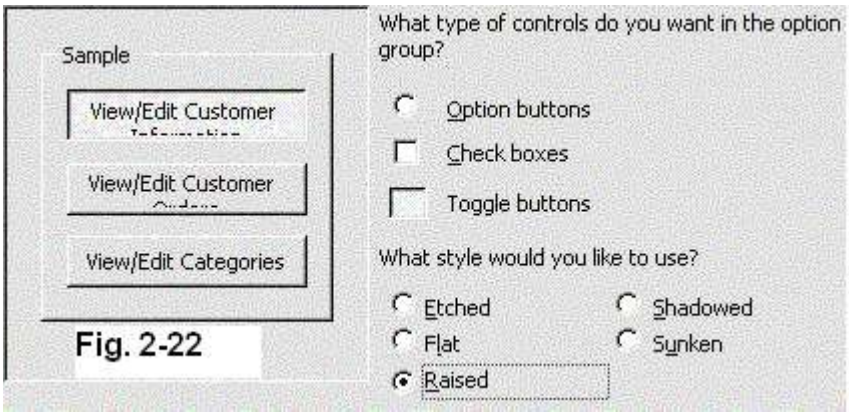


Fig. 2-22

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Enter "**Menu**" as a caption of this option set. Click **Finish** to end the wizard.

Fig. 2-23

At this moment, these buttons, as shown in Fig. 2-24, are only pure objects, having no actions (macros or codes) behind them.

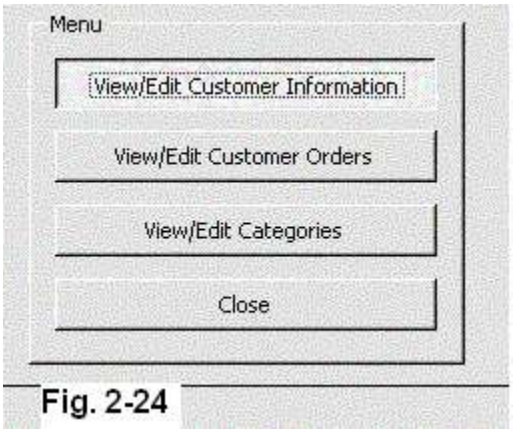
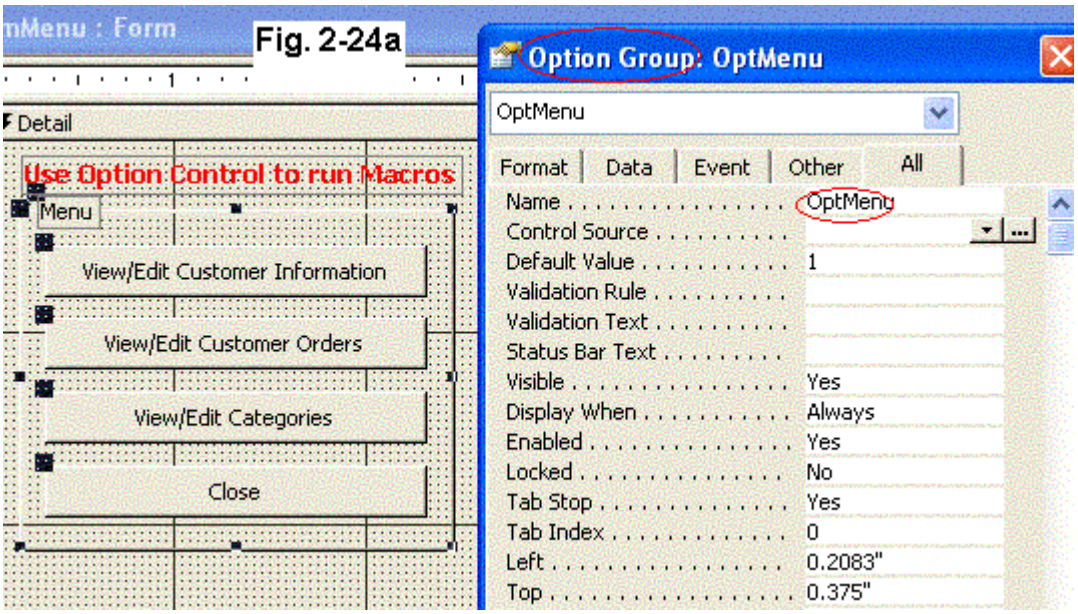


Fig. 2-24

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Right-click on the frame border to open the **Option Group** properties as exhibited in Fig. 2-24a. Enter the name, **optMenu**. Save the form.



Next, you will need to assign the above option group with a group macro with a **Condition** column.

Open a new macro, with the columns **Macro Name** and **Condition,** by opening the **View** menu (see Fig. 2-25).

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Enter **gMcrOptMenu** into the Macro Name column. Right-click on the first row of the **Condition** column, and open the **Expression Builder** by selecting **Build** on the pop-up menu. Click the "+" sign next to **Forms**, click on **All Forms**, then click **frmMenu** to display all objects in the middle column, as displayed in Fig. 2-26. Select the first object, **OptMenu**, in the middle column and paste its value into the expression area. You need to remove **Forms![frmMenu]!** from the expression and add **=1** on the right side of **[OptMenu]**. The final expression should like **[OptMenu]=1**. Of course, if you think the above steps require too much work, you can just enter **[OptMenu]=1** into the **Condition** column. Move to next column, **Action**, and select **OpenForm**. Detailed arguments are shown in the next few figures.

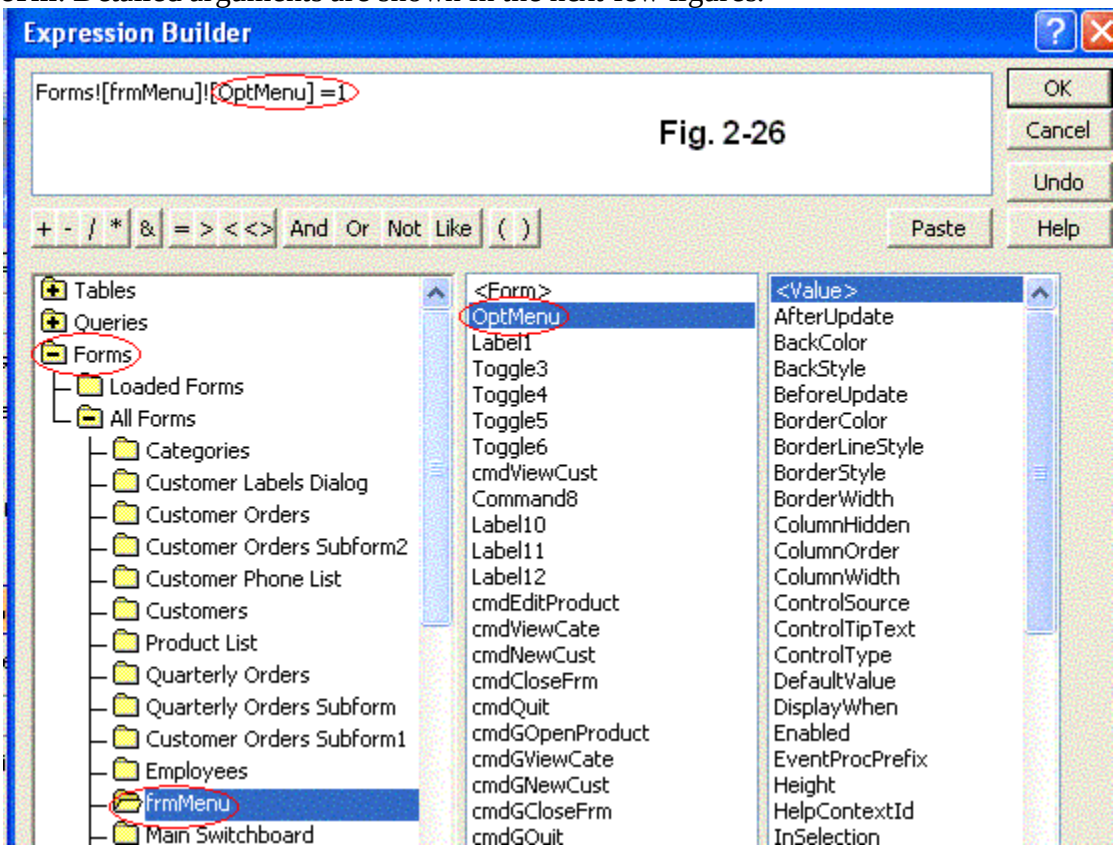


Fig. 2-26

[OptMenu]=1

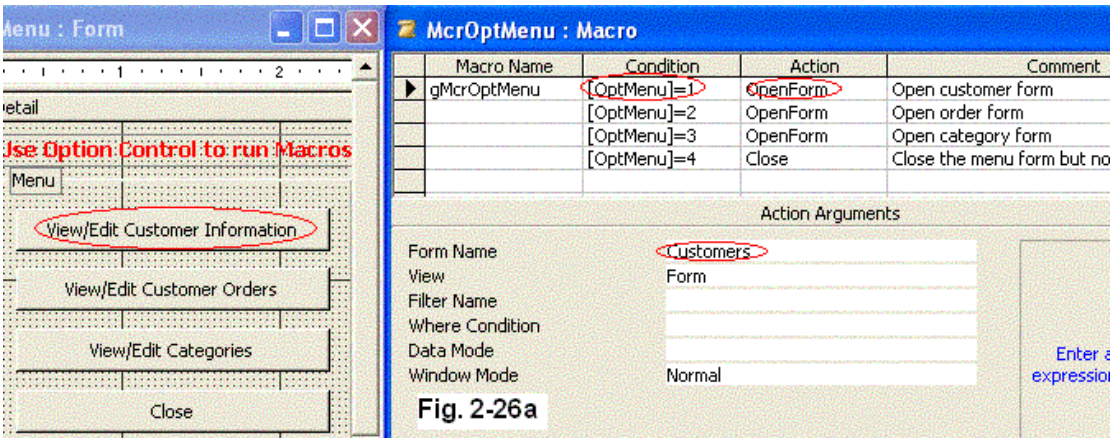


Fig. 2-26a

[OptMenu]=2

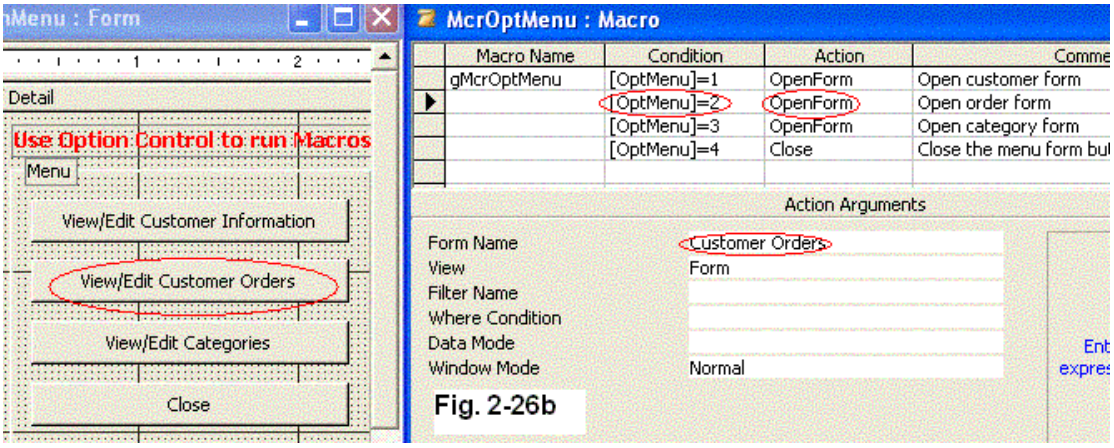


Fig. 2-26b

[OptMenu]=3

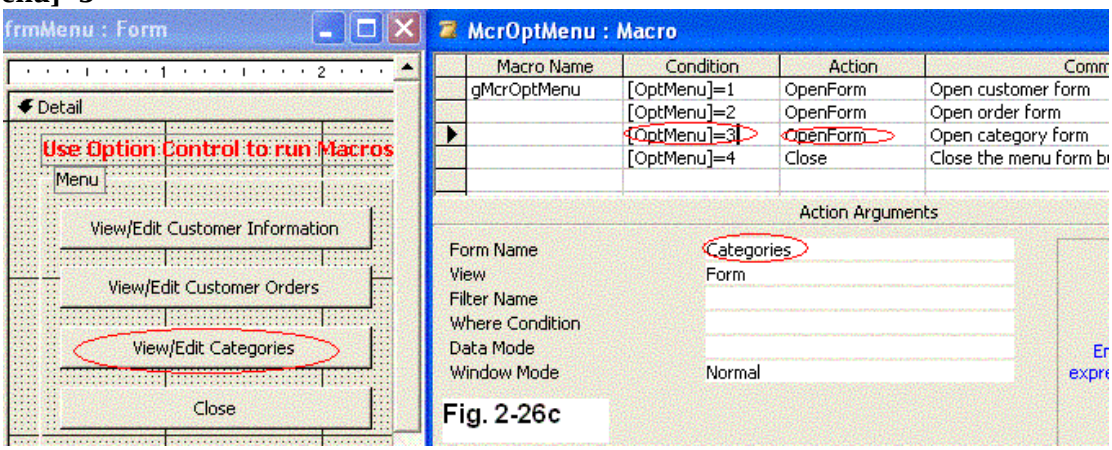


Fig. 2-26c

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

[OptMenu]=4

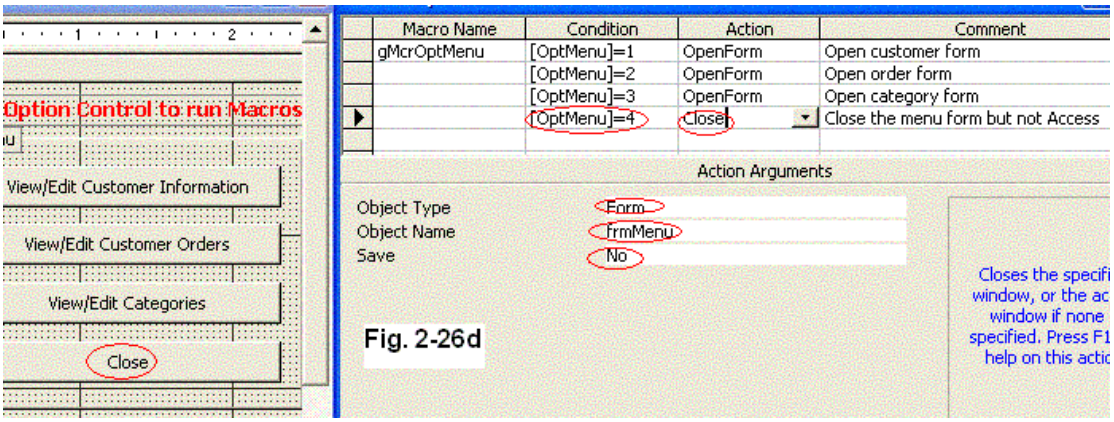


Fig. 2-26d

Save the macro **McrOptMenu**. Open the form **frmMenu**. Open the property of the **Option Group** and click the **Event** tab, as shown in Fig. 2-27. Select the macro **McrOptMenu.gMcrOptMenu** in the event **After Update**. Save the form.

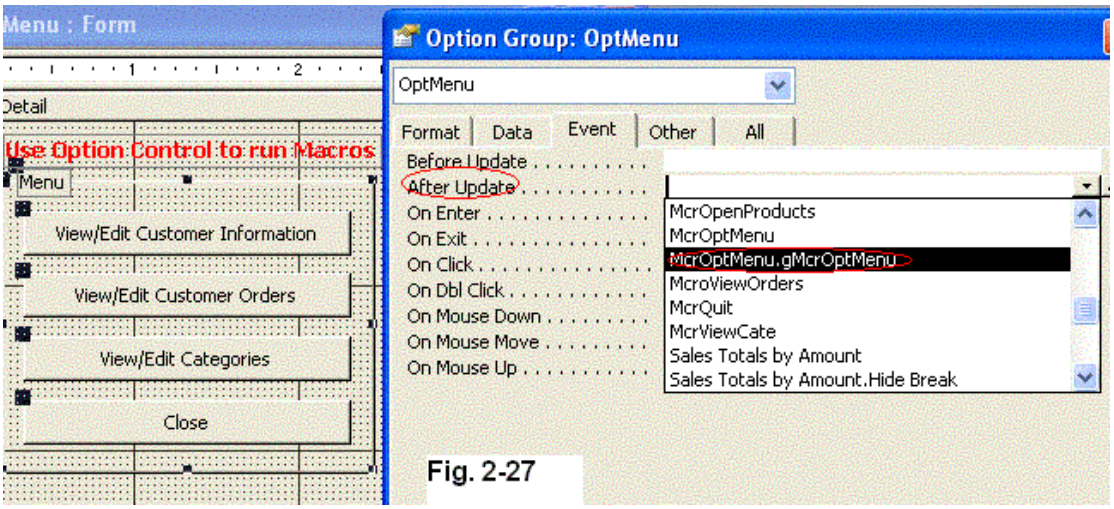


Fig. 2-27

Open the form **frmMenu** and test it. In Fig. 2-28, you can see that the first toggle button is sunk in and the customer information form appears.

Fig. 2-28

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Flavor 2.5 How do you convert a Macro to a VBA procedure? table of contents

For inexperienced VBA code programmers, it is relatively easier to create a Macro than a VBA procedure. However, the macro performance is limited to the list of macro commands. For some projects, it will be more strenuous to write an entire VBA code than to simply modify the macro code. To do so**,** the macro will need to be converted to VBA procedure first. Here are the steps for the conversion.

Select the macro, for example, **McrDemo1**, you want to convert. Click **Tools** on the **Menu** bar, then **Macro**, then **Convert Macros to Visual Basic**, as shown in Fig. 2-29.

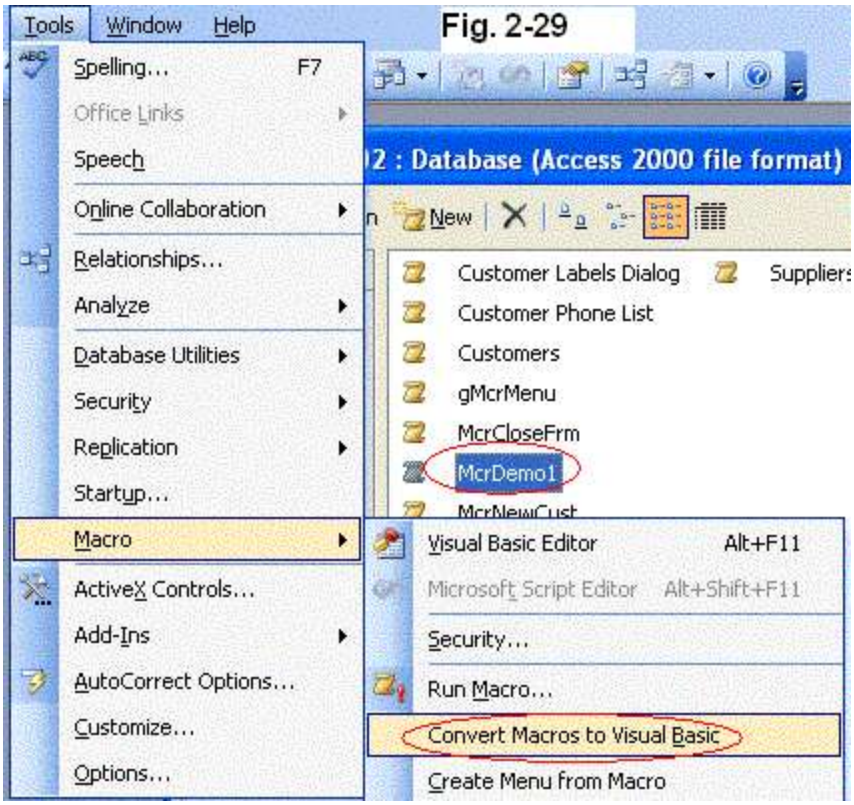


Fig. 2-29

Then Fig. 2-30 shows to check both boxes so the new procedure will have an error handler and display all remarks. Click **Convert**.

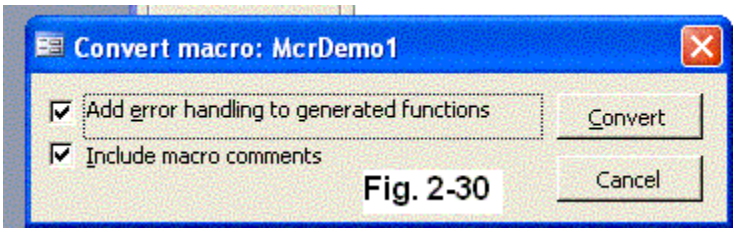


Fig. 2-30

You will see the **VB Editor**, as shown in Fig. 2-31, with a message box. Click **OK** to end the conversion.

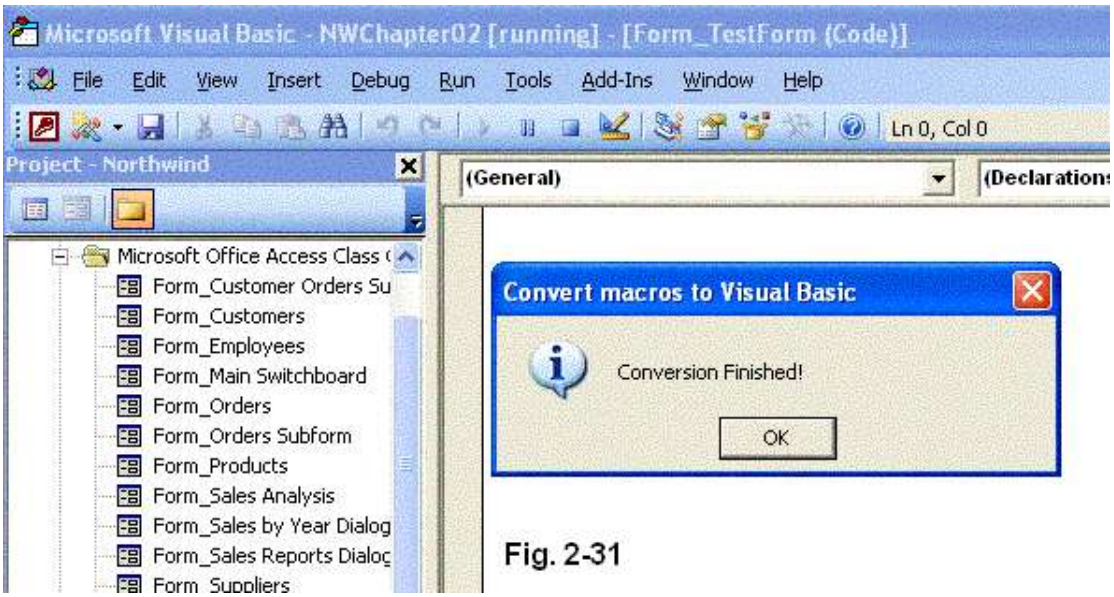


Fig. 2-31

The **VB editor** will then show up to display the procedure, just like Fig. 2-32. If there is no procedure shown in the **VB editor** after the conversion, select the macro name in the **Module** and then the procedure will appear.

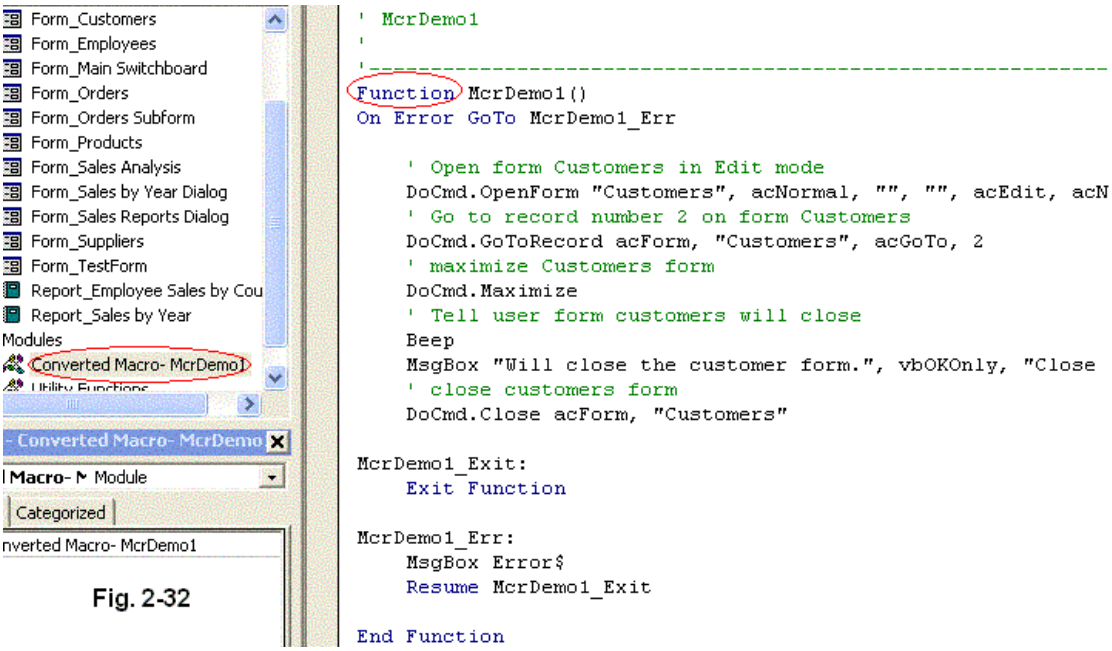


Fig. 2-32

Even if the macro is converted to a module (i.e. a function) and not a regular sub procedure, you still can make it a sub procedure by copying the code and pasting it into a new private procedure. The difference between the function and procedure (private or public) will be

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

discussed in Dish 5. If the sub procedure, rather than the module, is the final destination of the macro conversion, here is the procedure you can consider using.

In the **Objects** window, select **Form_TestForm** to view the code as Fig. 2-33 displays.

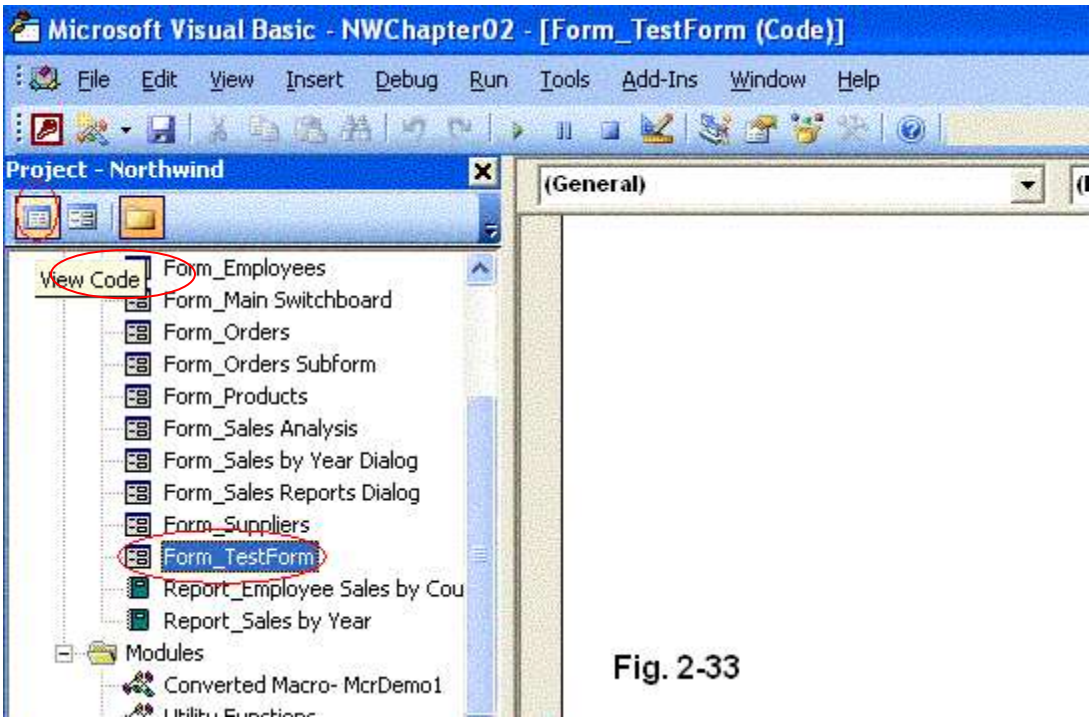


Fig. 2-33

Leave the cursor inside the code area. Go under **Insert** on the menu bar (see Fig. 2-34).

Fig. 2-34

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Type the procedure name, which is "**Customers**" in this case. Select **Sub** as the procedure type. Select **Private** as shown in Fig. 2-35. Click **OK**.

Fig. 2-35

Copy the entire code in the module, except the first (Function Name) and the last (End Function) line of code, as illustrated in Fig. 2-36.

```
'-----------------------------------------------------------
Function McrDemo1()
On Error GoTo McrDemo1_Err

    ' Open form Customers in Edit mode
    DoCmd.OpenForm "Customers", acNormal, "", "", acEdit, acNormal
    ' Go to record number 2 on form Customers
    DoCmd.GoToRecord acForm, "Customers", acGoTo, 2
    ' maximize Customers form
    DoCmd.Maximize
    ' Tell user form customers will close
    Beep
    MsgBox "Will close the customer form.", vbOKOnly, "Close customer
    ' close customers form
    DoCmd.Close acForm, "Customers"

McrDemo1_Exit:
    Exit Function

McrDemo1_Err:
    MsgBox Error$
    Resume McrDemo1_Exit

End Function
```

Fig. 2-36

Paste it into the Private Sub you just made. One line needs to be modified. Change **Exit Function** to **Exit Sub**. Another line, McrDemo1(), needs to be removed, as shown by Fig. 2-37.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

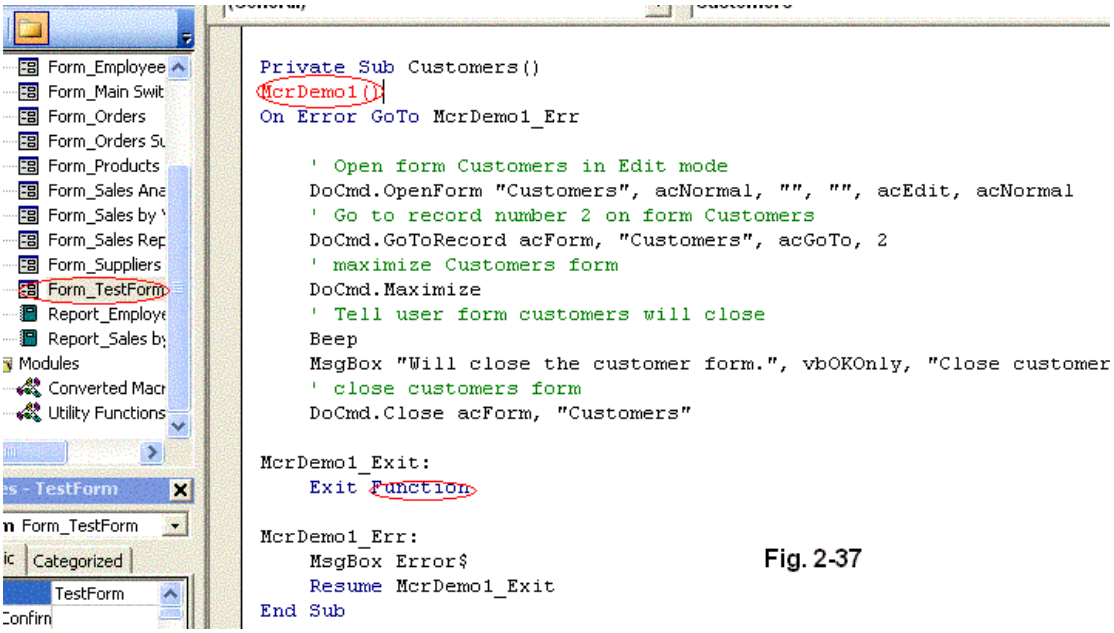


Fig. 2-37

The final code should look like Fig. 2-38.

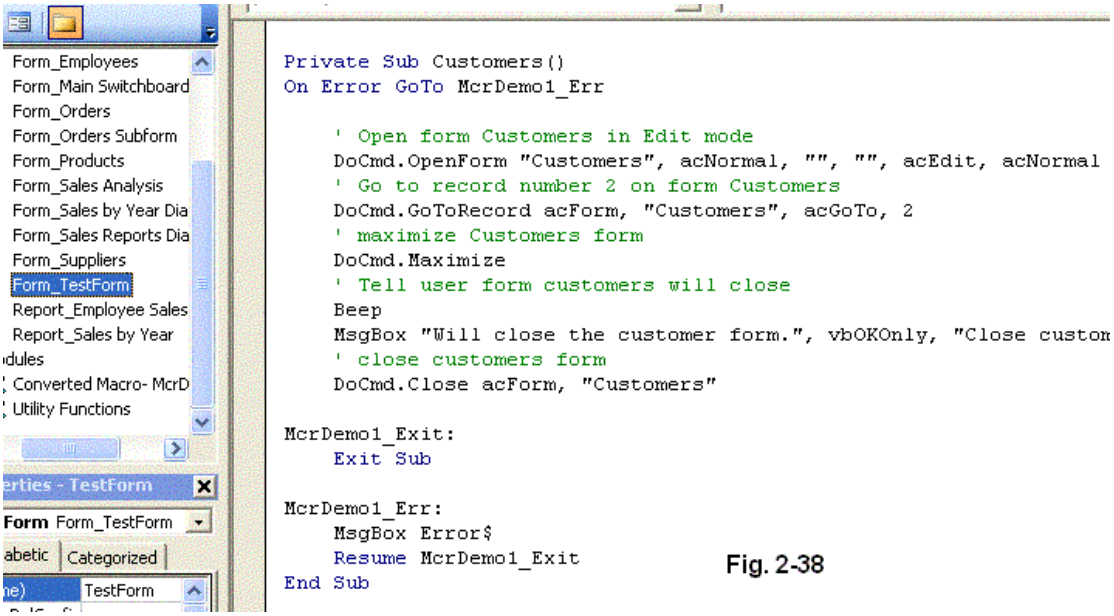


Fig. 2-38

Therefore, the procedure is completed!

What if you want to make this procedure work but there is no command button on the form that can trigger the event? You can still make a connection, even if it makes no sense to open a **Customer form** (with the macro code behind it) from a **Customer form** (existing form **frm_TestForm**).

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Open the **VB Editor** to see the code you just pasted and modified. Add **cmd** before **Customers** and add **_Click** after **Customers**, as shown in Fig. 2-39.

```
(General)                                   cmdCustomers_cli

Private Sub cmdCustomers_click()
On Error GoTo McrDemo1_Err

    ' Open form Customers in Edit mode
    DoCmd.OpenForm "Customers", acNormal, "", "",
    ' Go to record number 2 on form Customers
    DoCmd.GoToRecord acForm, "Customers", acGoTo,
    ' maximize Customers form
    DoCmd.Maximize
    ' Tell user form customers will close
    Beep
    MsgBox "Will close the customer form.", vbOK(
    ' close customers form
    DoCmd.Close acForm, "Customers"

McrDemo1_Exit:
    Exit Sub
```

Fig. 2-39

Close the **VB Editor** and save the code. Open the form **TestForm** in **Design View**. Add a command button with the **Control Wizard** inactivated. Change the Name and Caption of the button, as indicated in Fig. 2-40.

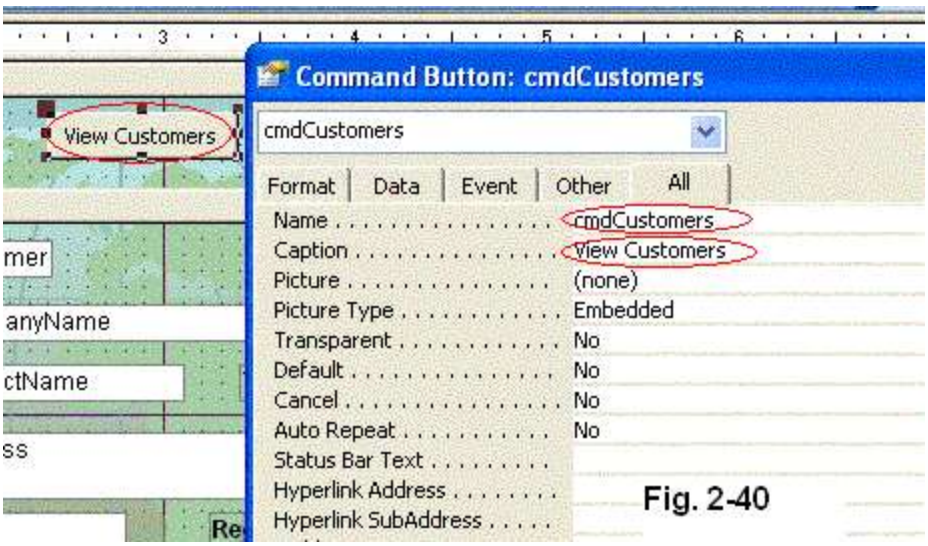


Fig. 2-40

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Click the event **On Click** under the **Event** tab. Click to select **Code Builder** in the **Choose Builder** dialog window (see Fig. 2-41). Click **OK**.

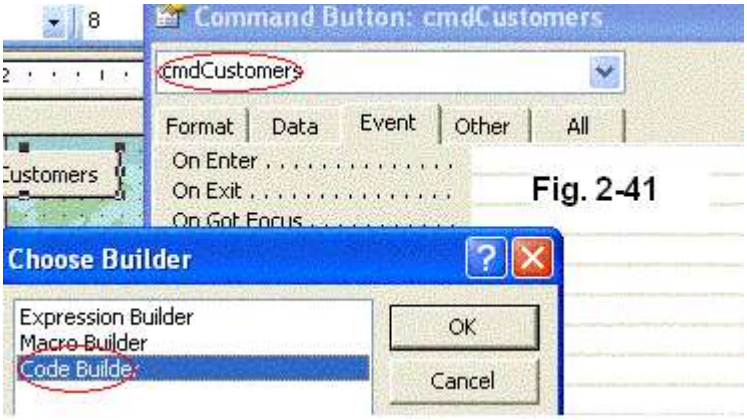


Fig. 2-41

The **VB Editor** appears automatically and your cursor is blinking in the **cmdCustomers_Click()** sub procedure as shown in Fig. 2-42. Bingo! Now, the front object, command button, and the back, sub procedure, are connected.
Save it and close the **VB Editor**.

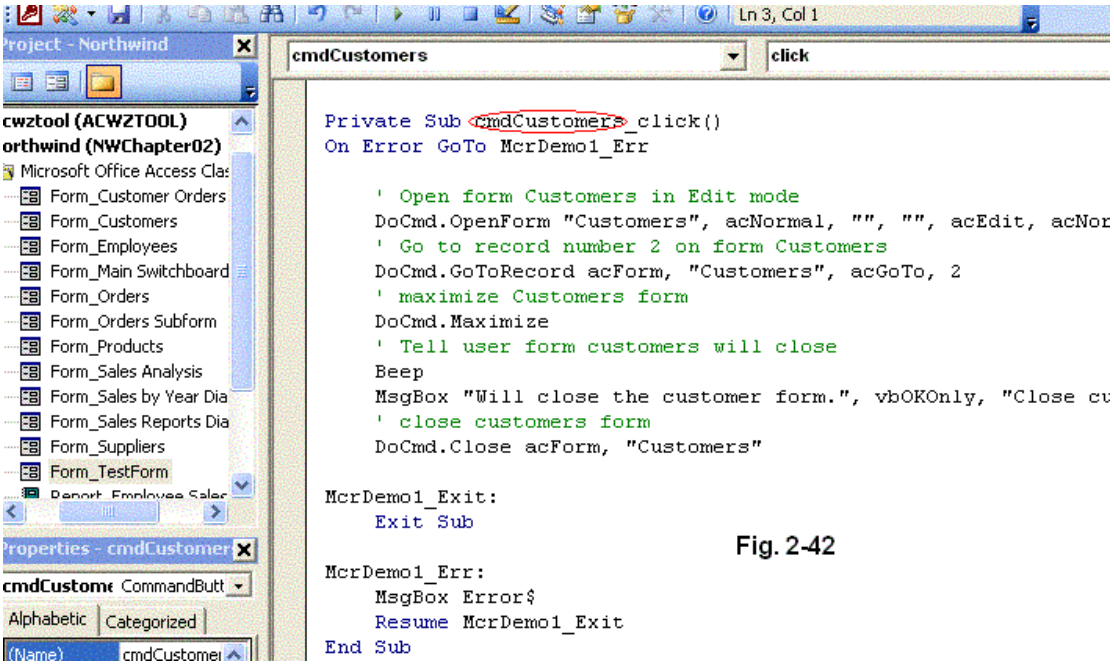


Fig. 2-42

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

The **[Event Procedure]** is in the **On Click** box, rather than the name of procedure. It tells you that the procedure behind this event is not a macro. If you want to view the procedure code you can click ... to see it in **VB Editor**. Close the property and save it.

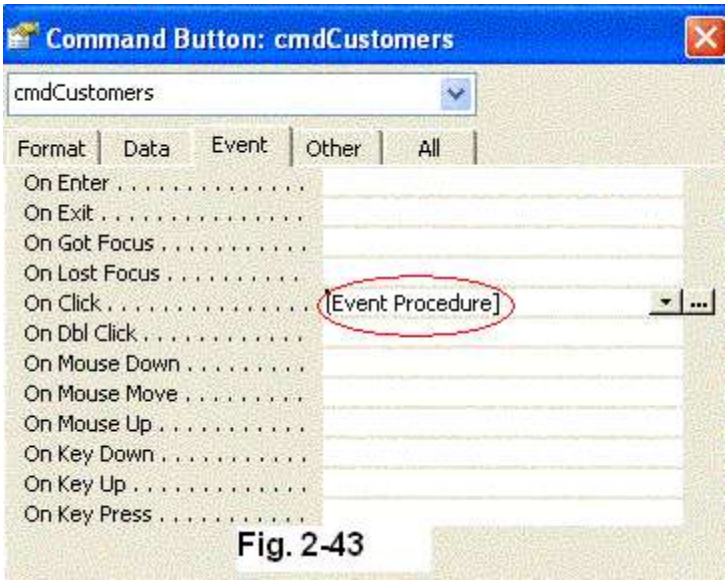


Fig. 2-43

Open the form **TestForm** in **Form View** and click the button, **View Customers**, to test it. You should see all the same actions executed as in the macro, **McrDemo1**, but rather than running the macro, it is run by the sub procedure, **cmdCustomers**. Amazing!

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition

Chapter 3 Custom Menu, Toolbars, Pop-up Menu

table of contents

(Note: This section only works for Access 97-2003. For Access 2007, it needs XML to customize Ribbon and pop-up menu which is beyond the scope of this book.)

Typically, an Access database application should disable all database design features (discussed later) to prevent users from changing the design. For some in-house applications, it is not necessary to put in extra effort to customize the menu, toolbars, or pop-up menu. However, to some degree, the application will still come back to you for solutions if it has problems after being implemented. If you do not know all the changes, in terms of design, made by users, you will have a very difficult time in fixing the problems quickly. It is best to disable all design features, regardless of who the application users will be

Flavor 3.1 Why should you use custom menus and toolbars?

table of contents

The first thing you need to know is what the **Startup** dialog box is, which can be found under **Tools** on the menu (see Fig. 3-1). This is where you can disable all built-in menus, toolbars, and pop-up menus by "un-checking" all the check boxes, preventing possible design changes. To make the form appear on the screen when the application is first opened, you will need to create a startup menu (form) like a switchboard form. This **Startup** dialog box is also the place to add custom menus and shortcut menus if you want to make them available on the application.

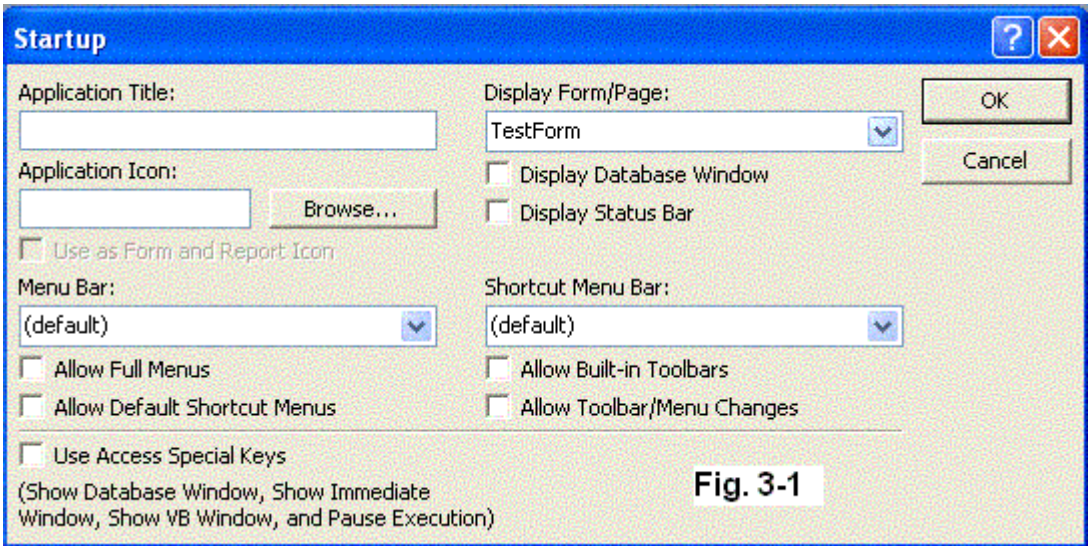


Fig. 3-1

Uncheck all boxes and select form **TestForm** from the drop-down list **Display Form/Page**, as shown in Fig. 3-1. Click the OK button to close the **Startup** dialog window. Exit Access and reopen **NWCh03** to test the startup.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

In Fig. 3-2, you can see that the menu bar cannot be further simplified. You may want to create your own, more useful menu bar to include the needed commands. A detailed instruction on how to bypass the **Startup** can be found in Dish 6.
You can close Access from the menu: **File**, **Exit** or just click the "X" button in the upper right-hand corner of the Access window (not the form **Customers**).

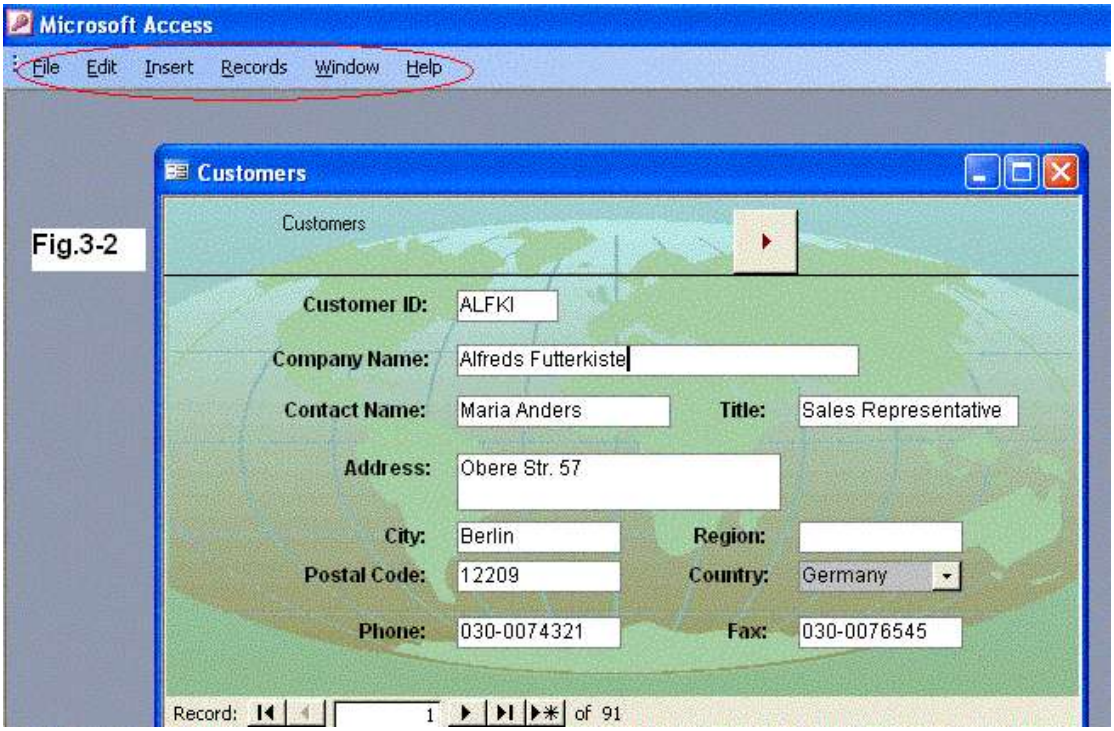


Fig.3-2

Flavor 3.2 Create a custom toolbar

table of contents

You can access the **Custom** dialog box by opening one of following:

From menu: **View**, **Toolbars**, **and Customize** (as shown in Fig. 3-3).

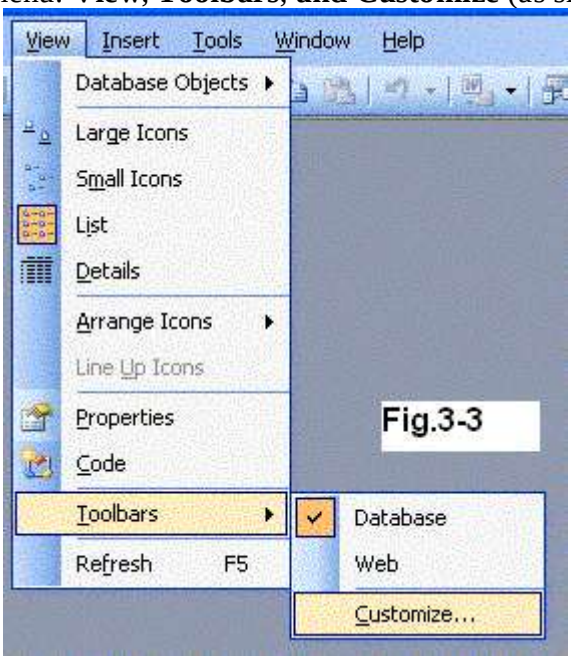


Fig.3-3

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Right-click anywhere on the menu bar or a toolbar. Click **Customize**… as Fig. 3-3a demonstrates.

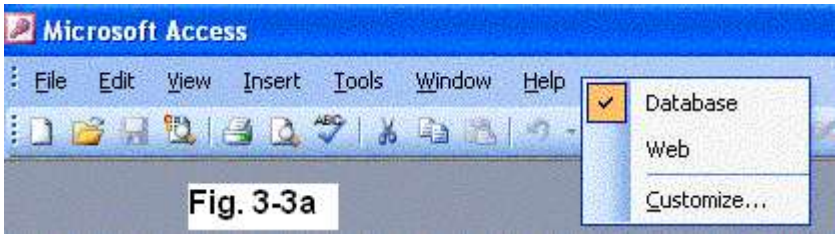


Fig. 3-3a

The **Customize** dialog box appears (see Fig. 3-4). In the **Customize** dialog box, click **New** on the **Toolbars** tab.

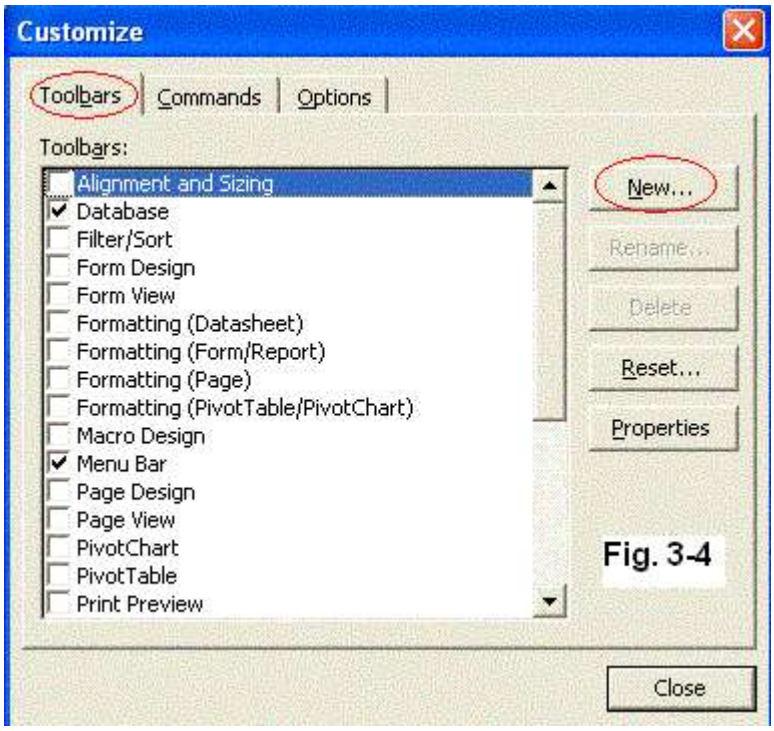


Fig. 3-4

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Type your toolbar's name, for example **MyToolBars**, into the name box, as illustrated in Fig. 3-4. Then click OK to close the **New Toolbar** input box.

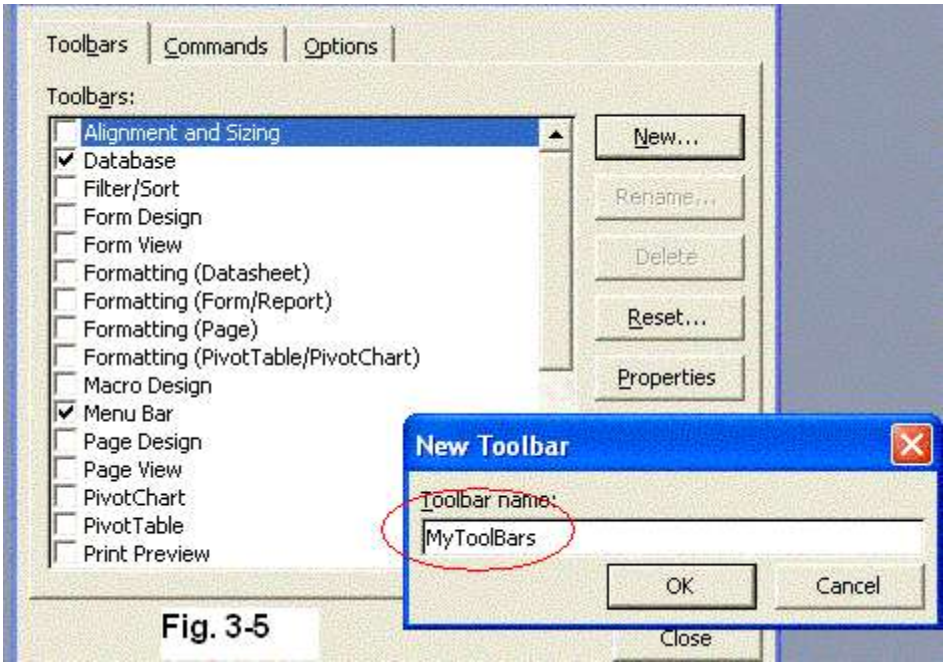


Fig. 3-5

At this moment, there is nothing in your new toolbar (Fig. 3-5a).

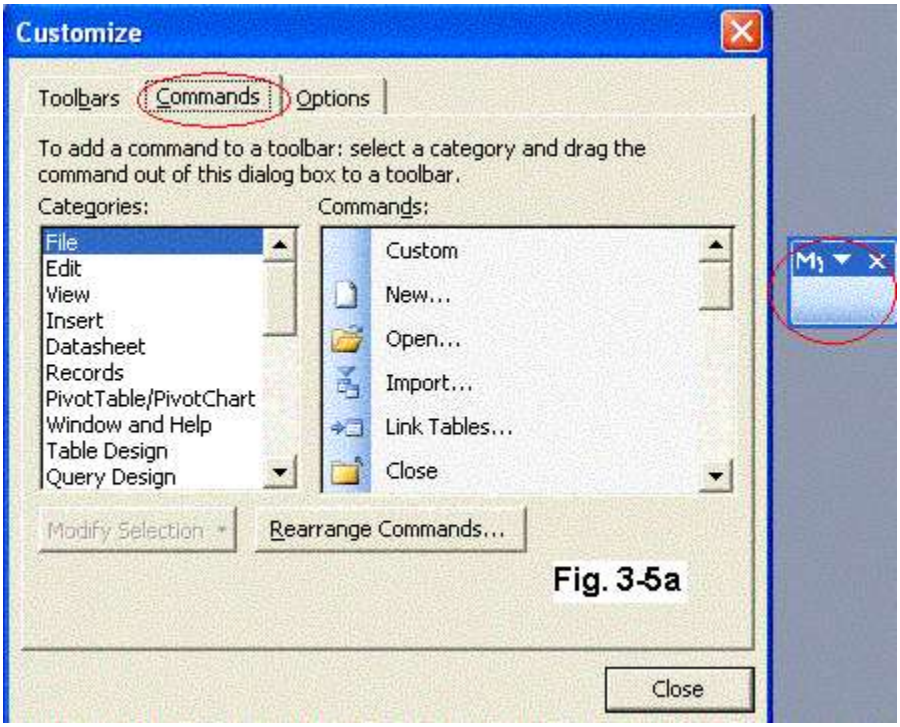


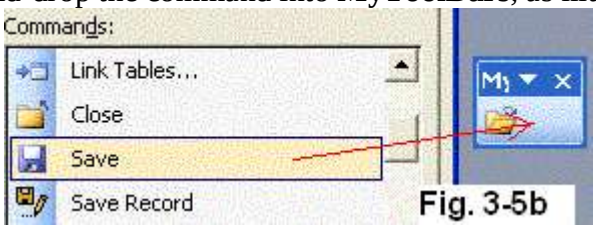
Fig. 3-5a

Now you need to add the proper commands into **MyToolBars**. Select the **Commands** tab in the **Customize** dialog box, like in Fig. 3-5a. Choose the item in the **Category** list and the commands in each category will be displayed in the **Commands** list. Select the command you would like to add to **MyToolBars** from the **Commands** list. You can use the mouse to

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

drag-and-drop the command into **MyToolBars**, as illustrated in Fig. 3-5b.



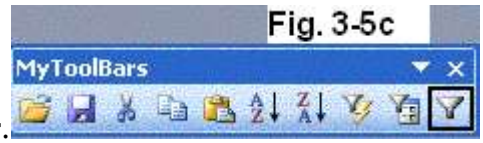
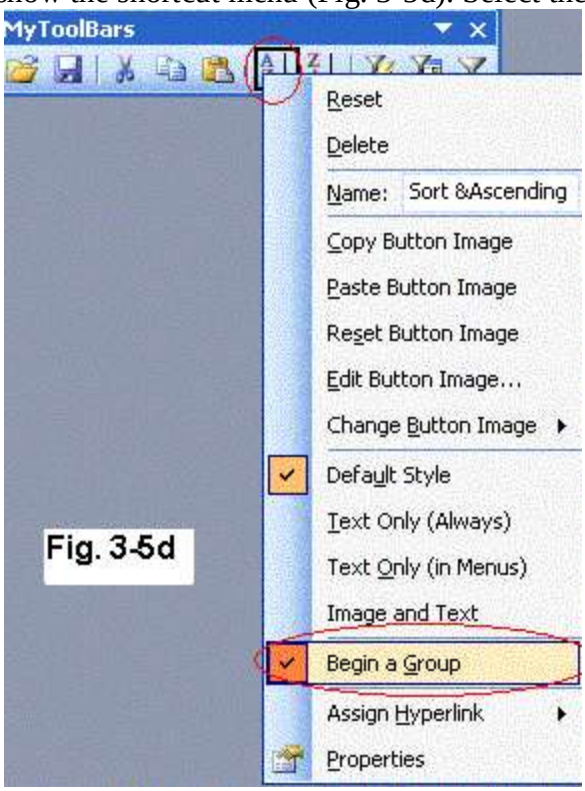
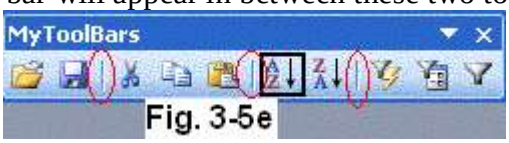


Fig. 3-5c displays the final customized toolbar.

You may want to group the related tools on the toolbars menu, such as **Open** and **Save** as a group and **Cut**, **Copy** and **Paste** as another. In the following figure, the **Open** and **Save** toolbars are grouped by adding a white separator on the right side of the **Save** icon. Here is how you can add such a cosmetic bar to make your custom toolbar more organized. If you, for example, would like to add a separator between and , right-click on toolbar to show the shortcut menu (Fig. 3-5d). Select the command **Begin a Group**.



A white bar will appear in between these two toolbar icons. Fig. 3-5e shows the completed toolbar.



In Fig. 3-5f, select **MyToolBars** on the **Toolbar** tab to view its **Toolbar Properties**.

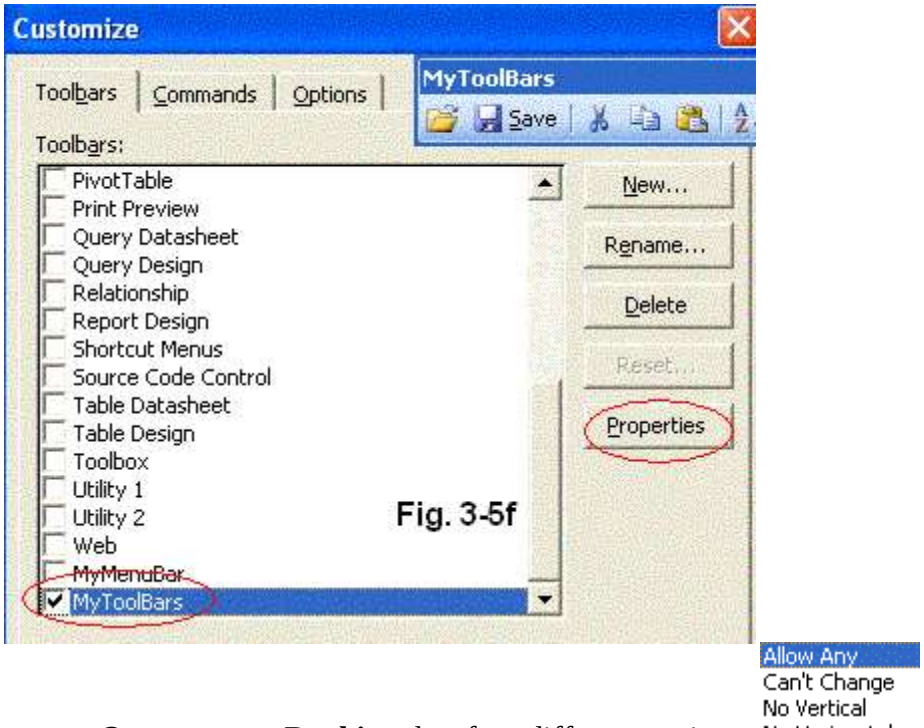


Fig. 3-5f

One property, **Docking**, has four different settings: (also in Fig. 3-5g). The default is **Allow Any**, which allows your custom-made toolbar to be docked anywhere in the Access window, just like a regular toolbar. **Can't Change** is straightforward, meaning that the user cannot dock it on the top, bottom, right or left side of Access screen and cannot change the width or height of the toolbar either. It means that the toolbar can dock on anywhere but not on four sides of window**.**

Fig. 3-5g

No Vertical means the user can dock it horizontally either on the top or bottom of Access screen (Fig.3-5h).

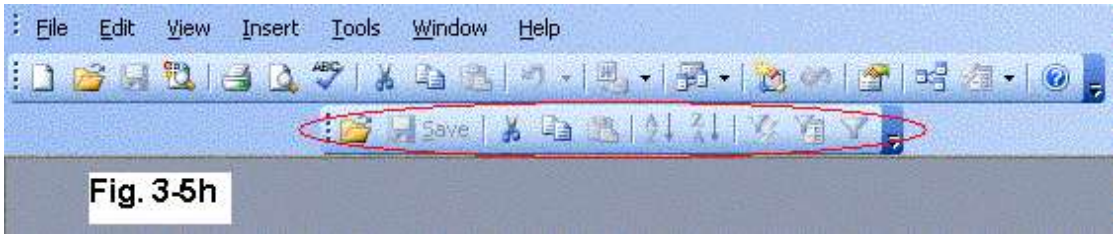
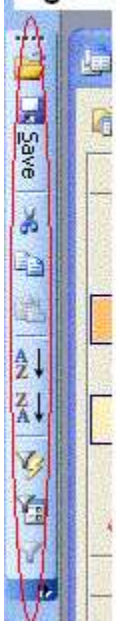


Fig. 3-5h

No Horizontal is the opposite of **No Vertical**, Fig. 3-5i is an example.

Fig.3-5i



Click **Close** to close the **Toolbar Properties** dialog window.

The above procedure is to add built-in toolbars into your custom toolbars. How about the macros you create? You can execute these macros by connecting them to the buttons on the form, and can also add them onto your custom toolbars. This way, your application will provide one more avenue for users to execute your macros. Here is the procedure on how to add the macros to custom toolbars.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

First, the **Customize** toolbars dialog window should be open, as shown in Fig. 3-5j. Select **All Macros** in the **Categories** list and choose one of the macros you want to add: for example **McrQuit** to the **MyToolBars**. Drag-and-drop it into **MyToolBars**, as was done in Fig. 3-5j. You may want to change the settings, such as changing or only displaying the image.

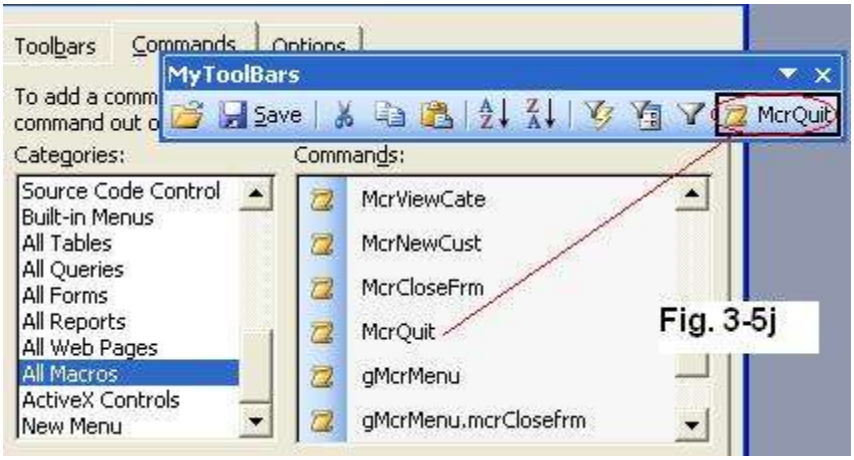


Fig. 3-5j

You can open the shortcut menu by right-clicking the mouse on top of the **McrQuit**. Click the **Default Style** to display only the image. Then click **Change Button Image** to choose one of the existing images; in this case, the image is selected to represent the **Exit**.

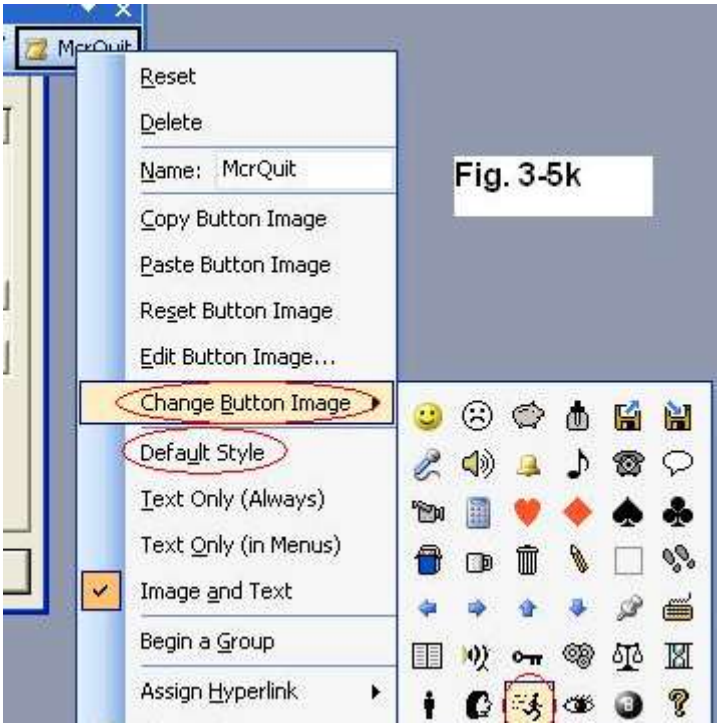


Fig. 3-5k

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Also in its properties, you need to change the **ScreenTip** to **Exit Database application** as shown in Fig. 3-5l.

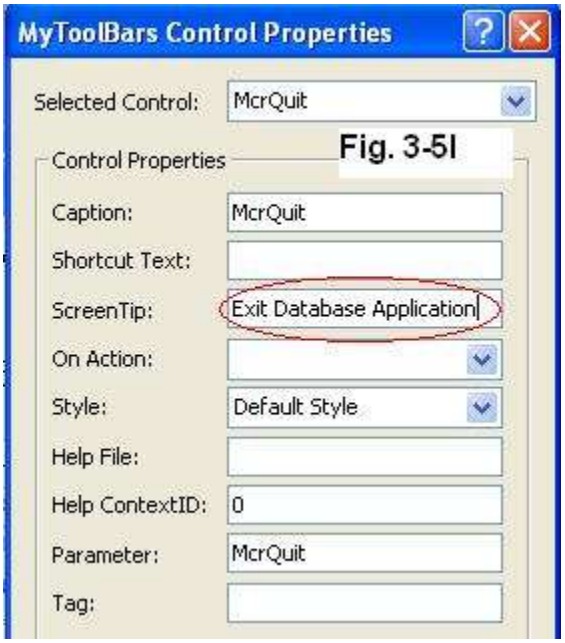


Fig. 3-5m shows the completed **MyToolBars**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

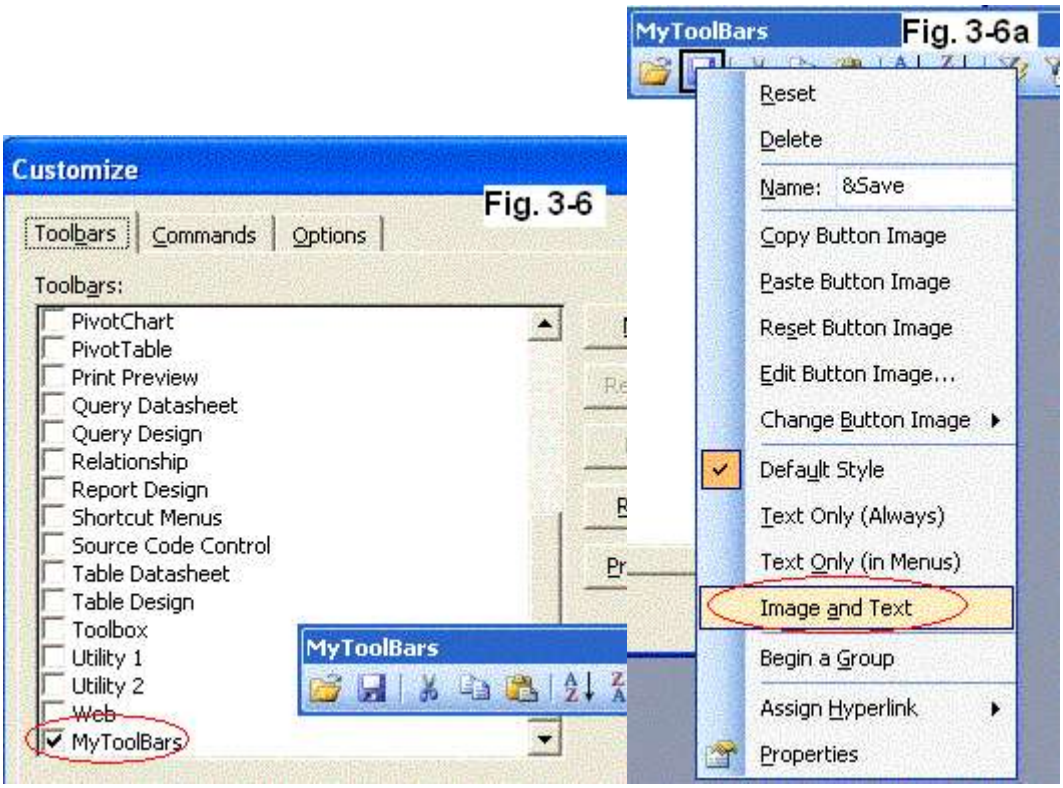
ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 3.3 Create a custom menu bar **table of contents**

Before you create a custom menu bar, let's take a look at the difference between the toolbar and menu bar. The biggest difference is that the former displays icon images as a default, while the menu bar may display both text and images. However, if you want, it is possible to add both images and text onto your toolbar. To do so, open the **Customize** dialog window and check the box in front of **MyToolBars** in the list of toolbars under the **Toolbars** tab. In order to add, remove or delete toolbars, you will need to have the toolbar **Customize** dialog window open.

Fig. 3-6 shows the custom toolbar that is to be modified. Let's change the image icon to display both text and image. On top of the icon, right-click the mouse to display the shortcut menu. Select **Image and Text** as shown in Fig. 3-6a. The on the toolbar is changed to

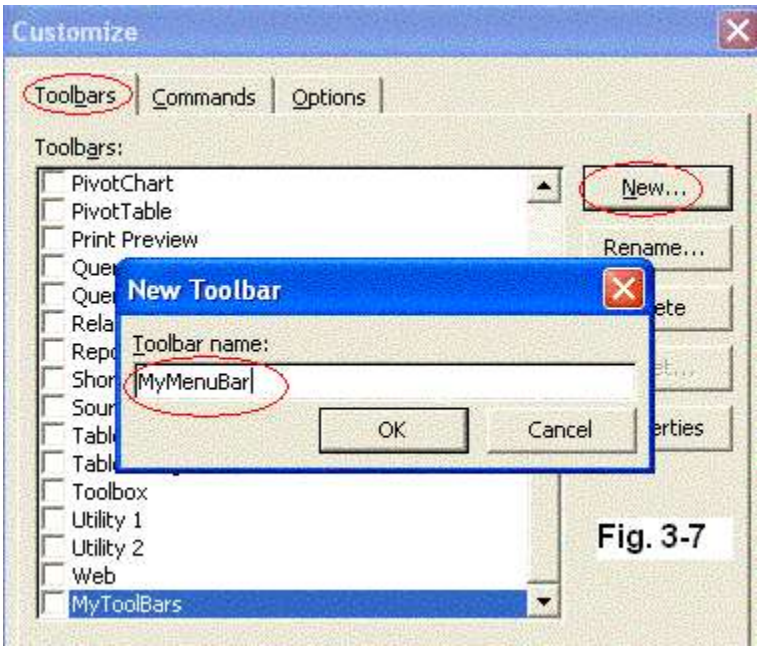
. However, this is not a completed menu bar. It is just another toolbar design. If you need to create a custom menu bar, you will need to change all icon images to text instead. You will also need to change the property **Toolbar** to **Menu Bar**.



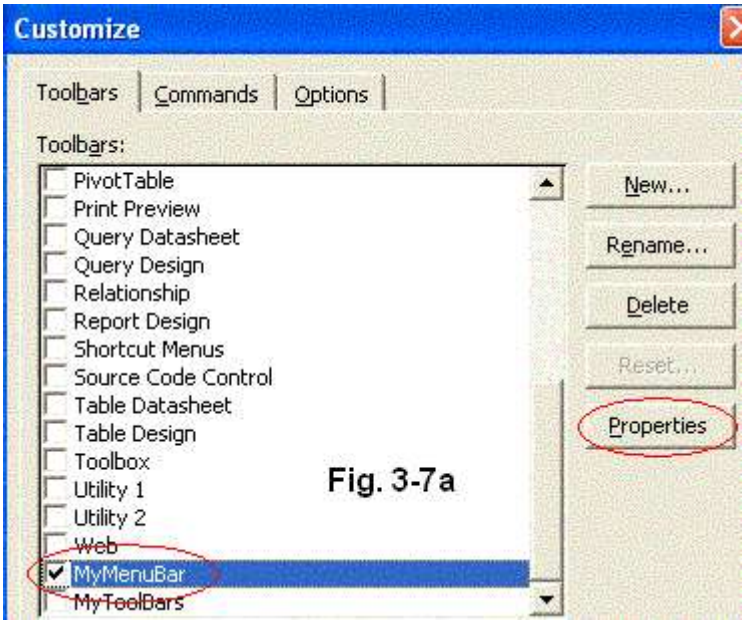
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

1. Making a menu bar is the same as making a toolbar. You need to have the toolbars **Customize** dialog window open. In the **Customize** dialog window, click **New** under the **Toolbars** tab. Let's give this custom menu bar a name, **MyMenuBar**, as shown in Fig. 3-7. Click OK to close the **New Toolbar** input box.



2. You need to change the type of toolbar in the property. Make sure the **MyMenuBar** check box is checked in the **Toolbars** tab. Click **Properties** (Fig. 3-7a).



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

In the **Properties** dialog window, click the **Type** drop-down list to change **Toolbar** to **Menu Bar**, as shown in Fig. 3-7b. Make sure the name in the **Selected Toolbar** is the name of the toolbar you want to change. After changing it to a **Menu Bar**, you will see that the **Show on Toolbars Menu** check box is automatically unchecked. Because this menu bar will be independent from any other toolbars, it should not be shown on the toolbars menu. If you want to use this custom menu bar as the main menu bar when the application is opened, you will then make changes in the **Startup** dialog window. This will be discussed at the end of this dish. Click **Close** to close the **Toolbar Properties** window.

Fig. 3-7b

3. Select the **Commands** tab in the **Customize** dialog box. On the bottom of the **Categories** list, select **New Menu** as Fig. 3-7c shows. There is only one item to be selected in the **Commands** list. Drag-and-drop this item into the blank toolbar you created.

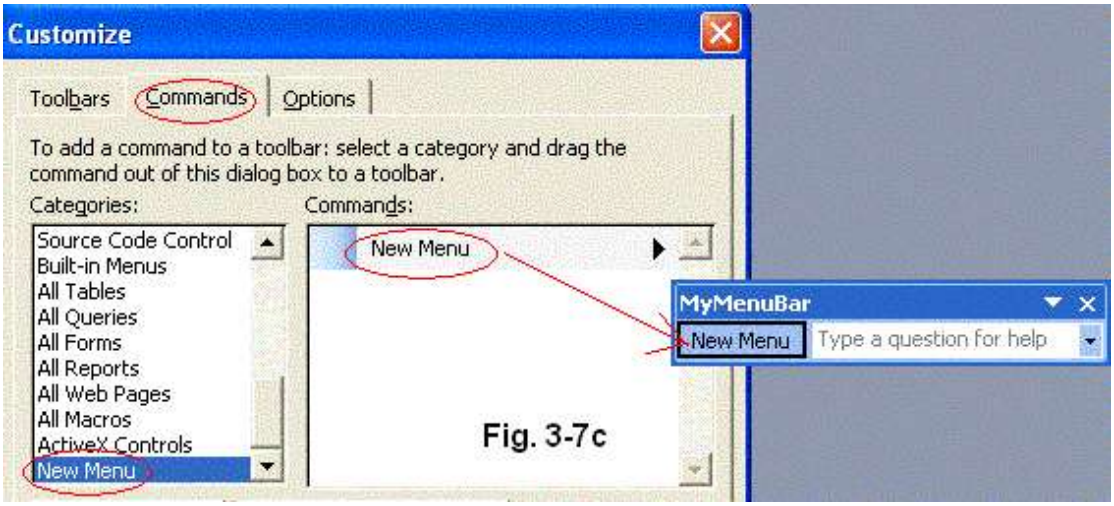


Fig. 3-7c

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

4. Open the shortcut menu by right-clicking on the **New Menu** of **MyMenuBar**. Change the name from **New Menu** to **Rec&ords** Name: Rec&ords . An ampersand (**&**) assigns "o" as a hot key. The properties associated with this menu command can be found by opening the **Properties** on the shortcut menu. Open the **Properties** (see 3-7d). In the **ScreenTip** property, enter **Records Operation**. Fig. 3-7e displays the screen tip text as the mouse hovers over the **Records** toolbar.

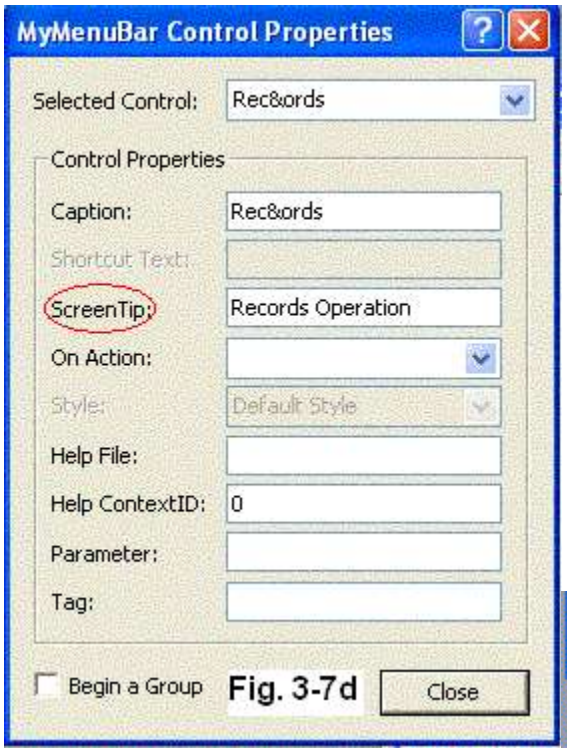


Fig. 3-7d

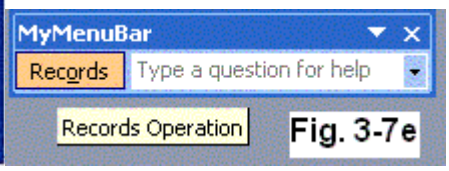


Fig. 3-7e

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

5. Add more commands to the menu bar **Records** by drag-and-dropping commands from the **Commands** tab into the blank square below the **Records** menu, as shown in Fig. 3-8. The I-beam indicates where the command will land once you release the mouse. For aesthetic purposes, it would be nice to group related commands on the menu bar by adding a separator, the same way you did in the toolbars, except you will right-click on the command below the separator bar as shown in Fig. 3-8b. A separator will be added between these two commands

by right-clicking on

. This will open the shortcut menu, where you select the **Group** command. If you would like to remove the separator from the menu, uncheck the **Group** command on the shortcut menu. Fig. 3-8a shows a menu list with separator bars.

Fig. 3-8

Fig. 3-8a

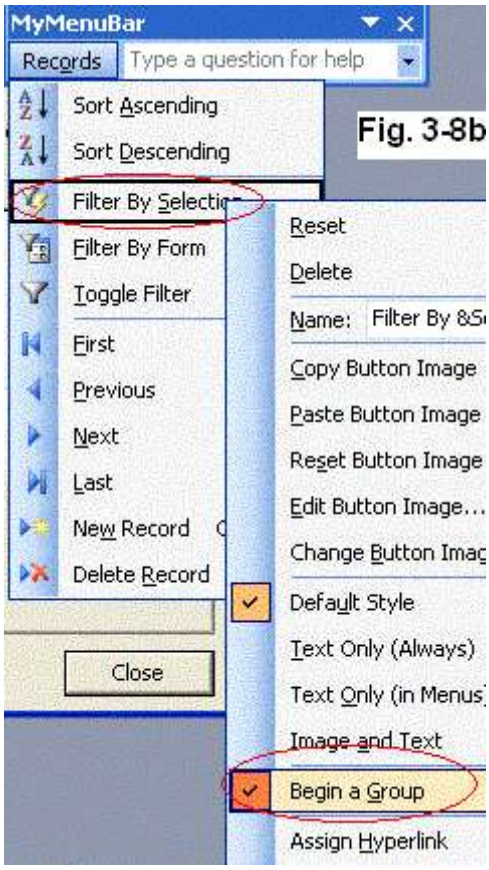


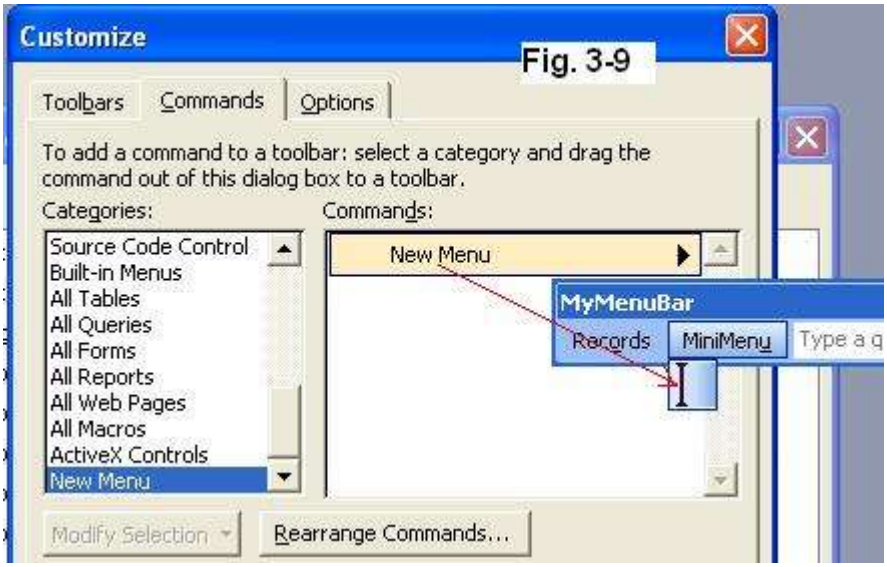
Fig. 3-8b

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 3.4 How do you make a submenu? **table of contents**

Another difference between menu bars and toolbars is that the former can have submenus, but the latter cannot. In this example, you will add another menu item **MiniMenu** with the hot key "u", and a submenu under **MiniMenu** (refer to Fig. 3-9). The procedure is the same as creating the first menu bar **Records**. You will add **New Menu** to the **Commands** list from the **Categories** list, as shown in Fig. 3-9.



Under the **Commands** tab, select the **New Menu** from the **Categories** list. Drag the **New Menu** from the **Commands** list to the block with the I-beam, below **MiniMenu** (see Fig. 3-9). Release the mouse if that is the location you would like **New Menu** to be at. Make sure you hold down the left-click mouse button, move over to **MiniMenu**, and go to the bottom of the menu list before releasing the mouse. It is a little tricky to insert this submenu, **New Menu**, into **MiniMenu**. If you miss the right entry, the **New Menu** will be on the same level as **MiniMenu** instead of being a submenu of **MiniMenu**. If this happens, you can either drag the **New Menu** back to the **Customize** dialog box (this is the way to remove any menu or toolbar item from a menu bar or toolbar) or drag it to the top of **MiniMenu** and go down to the bottom list of **MiniMenu** and release the mouse. Practice it a couple times and you will naturally know how to control the menu item and when and where to release the mouse.
After adding menu items into **MyMenuBar**, you may want to rename them. Here is the list of names you need to use on **MiniMenu** for its submenus.

Data Input
View Reports
Search
Exit

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Except for **Exit**, which you will connect a macro to, the first three items are submenus, as displayed in Fig. 3-9a.

Fig. 3-9a

You will add the menu items into each submenu. Here are the item lists for each submenu:

Data Input
- Products
- Customers
- Categories

View Reports
- Catalog
- Invoice
- Sale by Category

Search
- Current Product List
- Ten Most Expensive Products

First, add a **New Menu** named **Data Input**. Then select **All Forms** in the **Categories** list, and **Products** from the **Commands** list. Drag-and-drop **Products** to the **Data Input**'s block, as shown in Fig. 3-9b. Use the I-beam to place it, as shown. Add two more items, **Customers** and **Categories**, under the **Data Input** submenu. Use the same procedure to finish other submenus, **View Reports** and **Search**, by selecting **All Reports** in the **Categories** list for the **View Reports** submenu, and **All Queries** for **Search**.

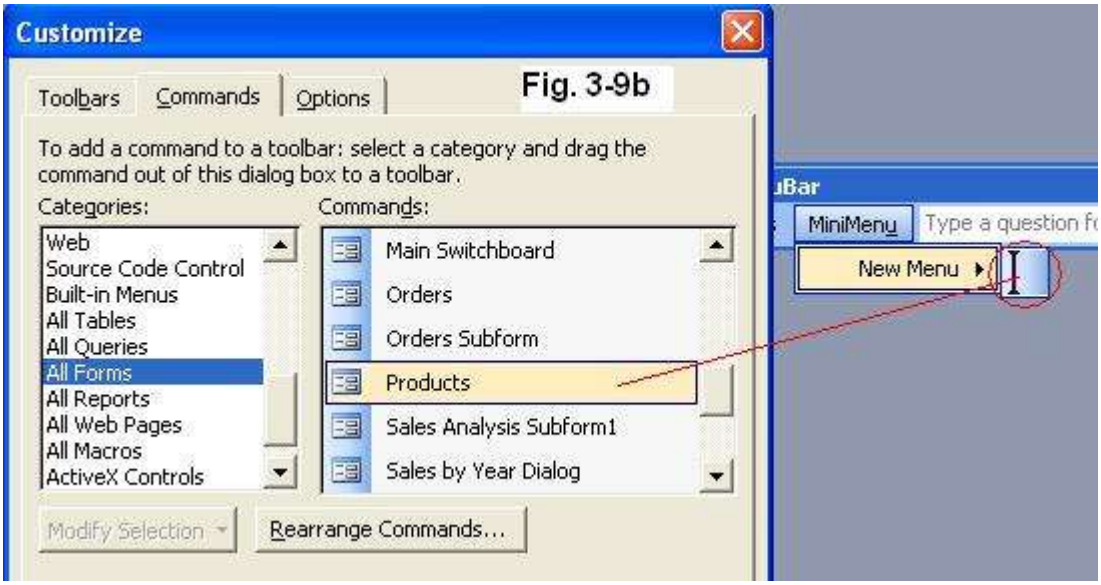


Fig. 3-9b

Fig. 3-9c shows the submenu items of **Data Input**.

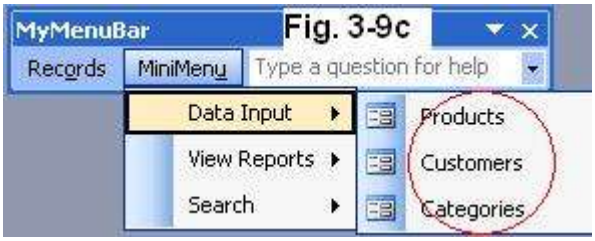
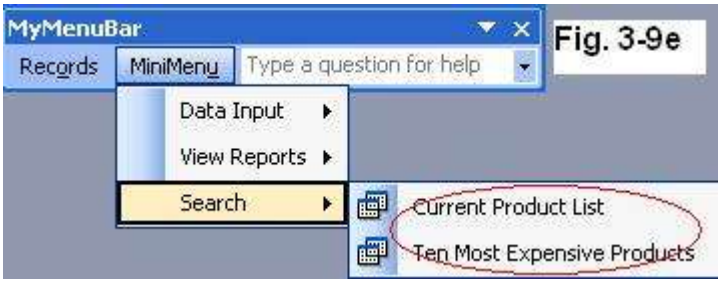


Fig. 3-9d shows the submenu items of **View Reports**.

Fig. 3-9e shows the submenu items of **Search**.



You may wonder if you can display only the text, rather than both image and text, for submenu items. Open the menu's shortcut menu by right-clicking the mouse, as was done in Fig. 3-10. Click the **Text Only (in Menus)**. Or, if you want to keep the image but not use the default standard image, click **Change Button Image** (see Fig. 3-10a).

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

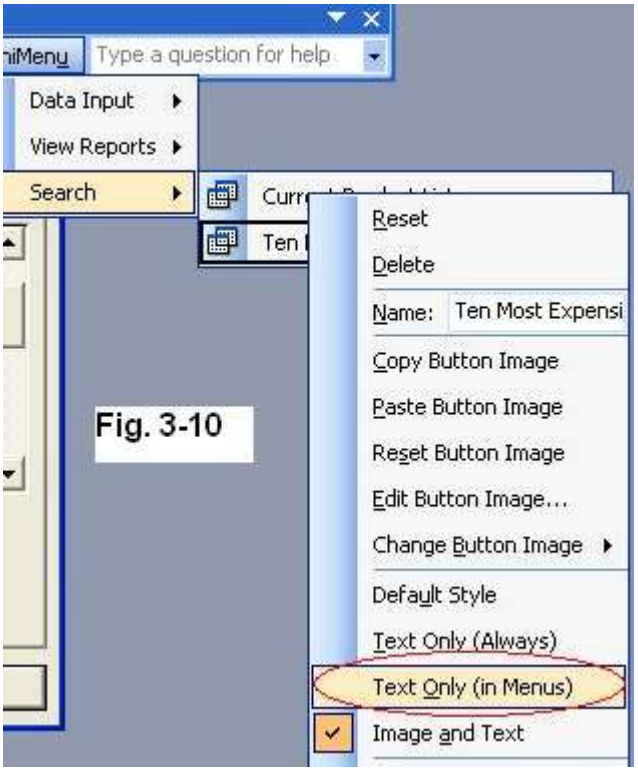


Fig. 3-10

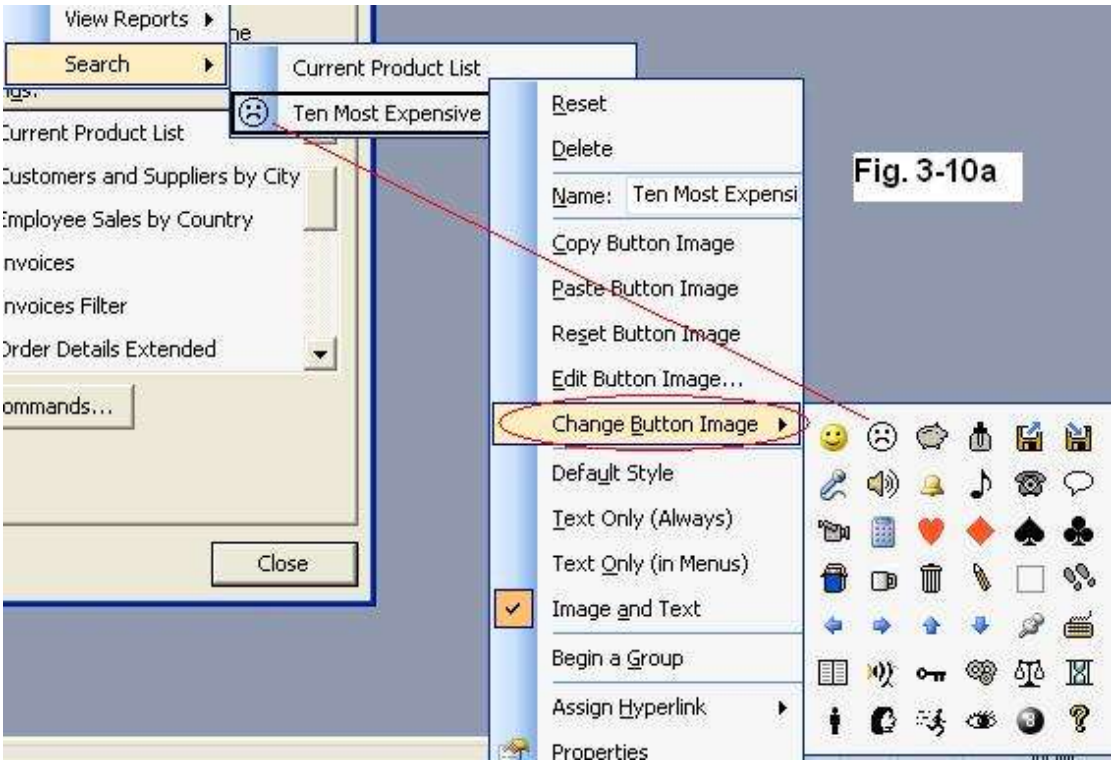
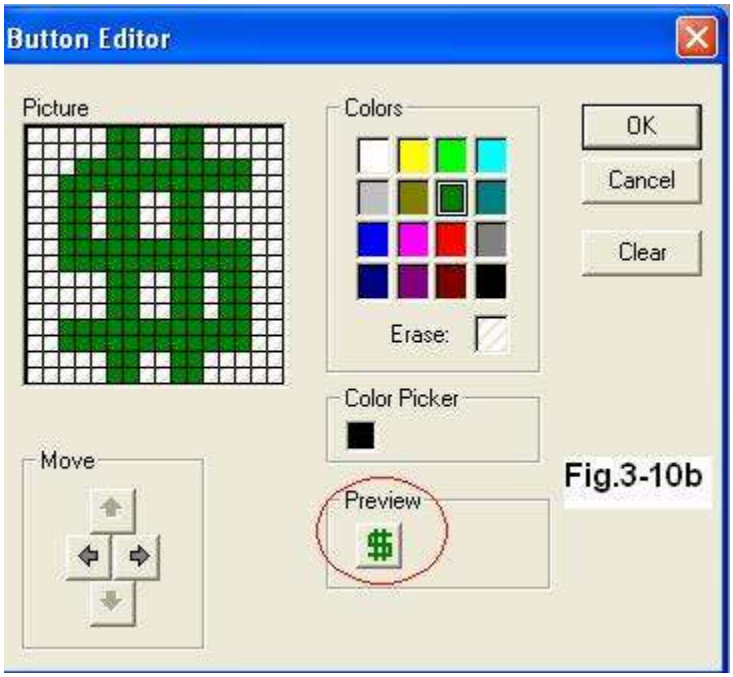


Fig. 3-10a

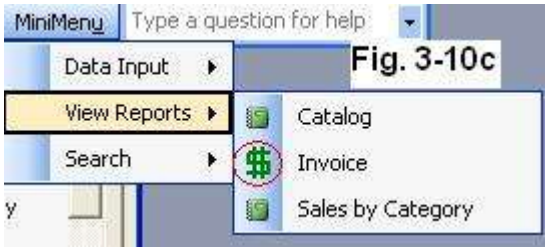
If you would like to make your own image, select **Edit Button Image** on the shortcut menu. The **Button Editor** appears and you can choose different colors to use on the **Picture** grid, as shown in Fig. 3-10b. The **Preview** is the image that will be shown on the menu.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

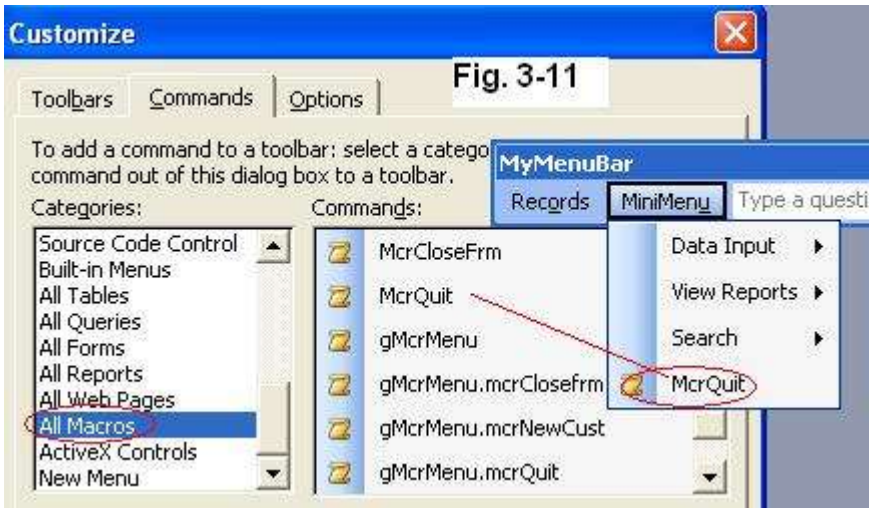
ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.



Click OK to close the **Editor**. You will then see the image you created on the menu. Fig. 3-10c shows the changed image.



One more menu item of **MiniMenu** needs to be added. However, this menu is not in the submenu list. You will connect one of the macros built in Dish 2 to end the application. Here is the procedure:



1. Select **All Macros** in the **Categories** list and **McrQuit** in the **Commands** list. Drag-and-drop **McrQuit** to the bottom of **MiniMenu**. Make sure to watch the horizontal I-beam (not

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

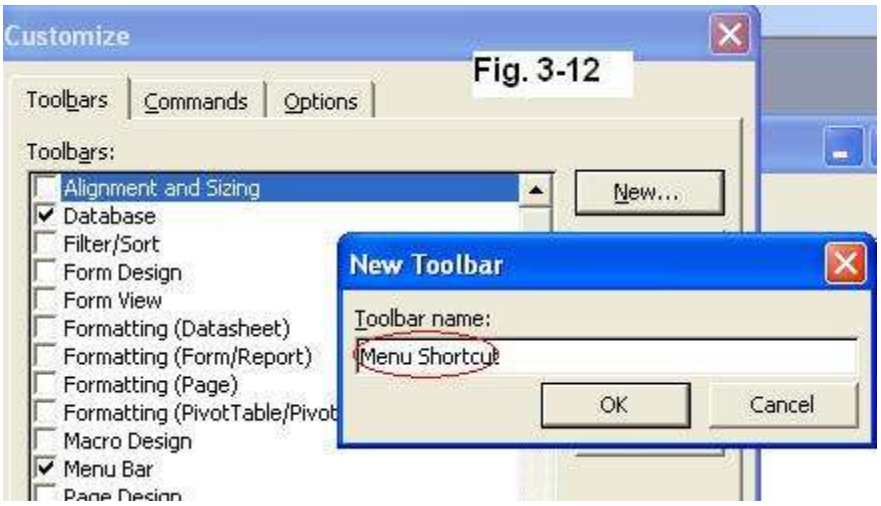
ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

the vertical I-beam) move to the bottom of the **Search** submenu. You should rename the **McrQuit** to **Exit** by opening the shortcut menu of **McrQuit**. Also, it is not a bad idea to keep text only on the menu, rather than both image and text, as illustrated in Fig. 3-11. Fig. 3-11a shows the modified menu.

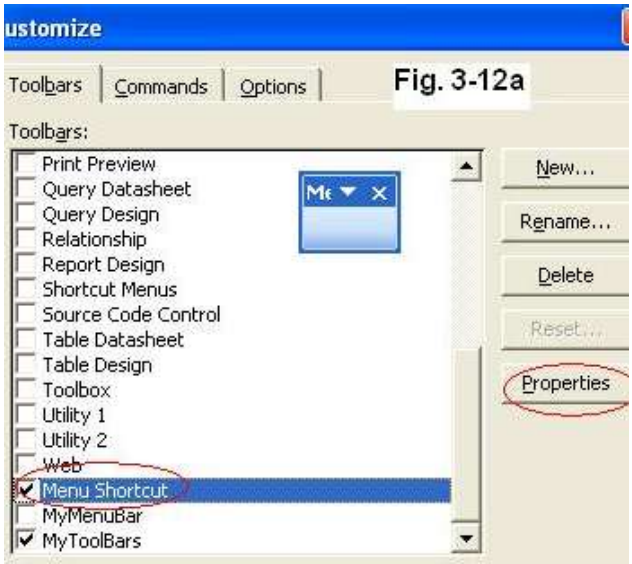
Flavor 3.5 Create custom shortcut menu (popup menu) table of contents

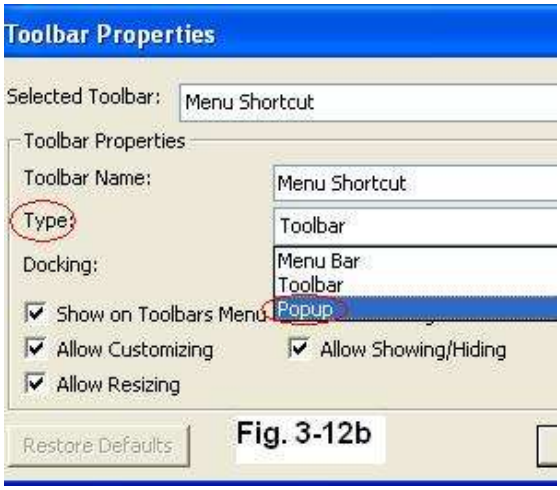
You may want to create your own shortcut menu to attach to forms instead of using the Access default. This shortcut menu is also called a pop-up menu or right-click menu. In this section, we will call it the pop-up menu. The procedure to create a custom pop-up menu is almost the same as creating a custom toolbar.

1. Open the **Customize** dialog box. Click **New** under the **Toolbars** tab to create a new toolbar. Enter a descriptive name, like **Menu Shortcut**, in the **New Toolbar** input box, as displayed in Fig. 3-12. Click OK to close the **New Toolbar** input box.



2. This step is highly important, as it distinguishes the procedure for creating menu and toolbars from pop-up menus. Make sure the item you select in the **Toolbars** tab is the one you want to make a pop-up menu (see Fig. 3-12a). Click the **Properties** button to open the **Properties** dialog window, as shown in Fig. 3-12b.





Select **Popup** from the **Type** drop-down list. Click **Close** to close the **Toolbar Properties** window. You will then see this message box:

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Click OK to close it. You should see **shortcut menu** under the **Toolbars** tab, because the pop-up menu is no longer an ordinary toolbar.

In the Toolbar Properties window, in Fig. 3-12c, the **Type** is **Popup**. Click **Close** on the **Toolbars Properties** window.

Fig. 3-12c

3. In the **Customize** dialog box, which is in Fig. 3-12d, look for **Shortcut Menus** and check the box. Next, you will see the shortcut menu bar pop up on the screen, as shown in Fig. 3-12e. Find the menu item named **Custom**, and click it to open the submenu. There is no menu item inside it.

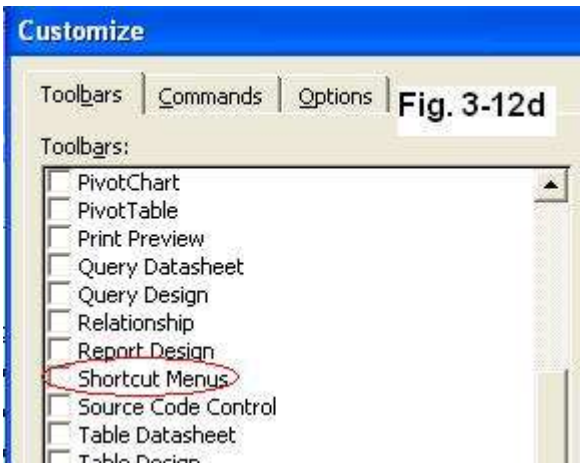
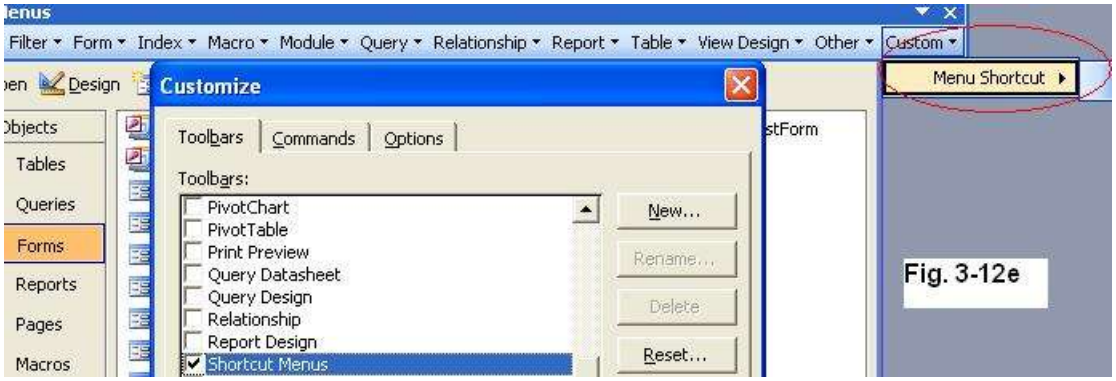


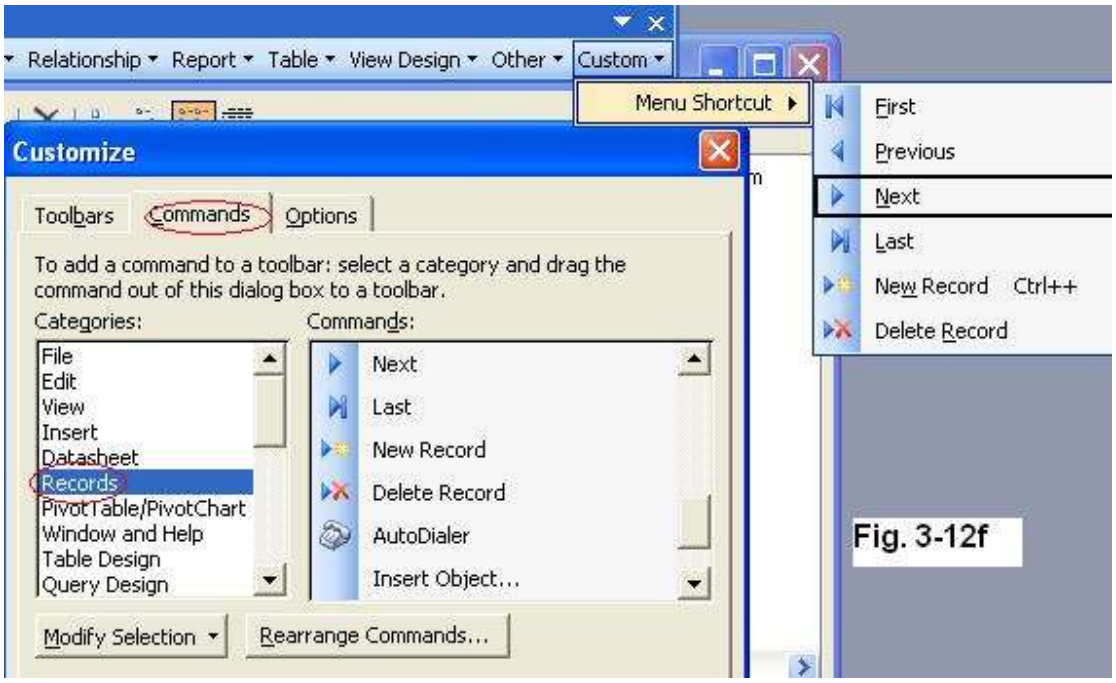
Fig. 3-12d

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.



4. The procedure to add commands into shortcut menus is the same as adding menu items. Select the **Commands** tab in the **Customize** dialog box, then **Records** in the **Categories** list. Choose the commands seen in Fig. 3-12f from the **Commands** list.

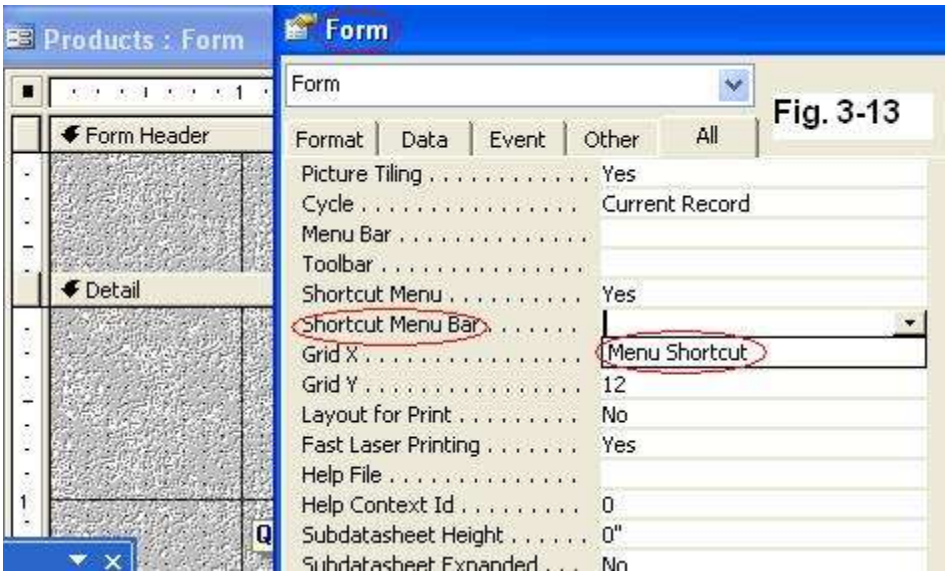


Close the **Customize** dialog window. Next, assign this shortcut menu to the form you want. When users right-click, this shortcut menu will appear, instead of the default shortcut menu.

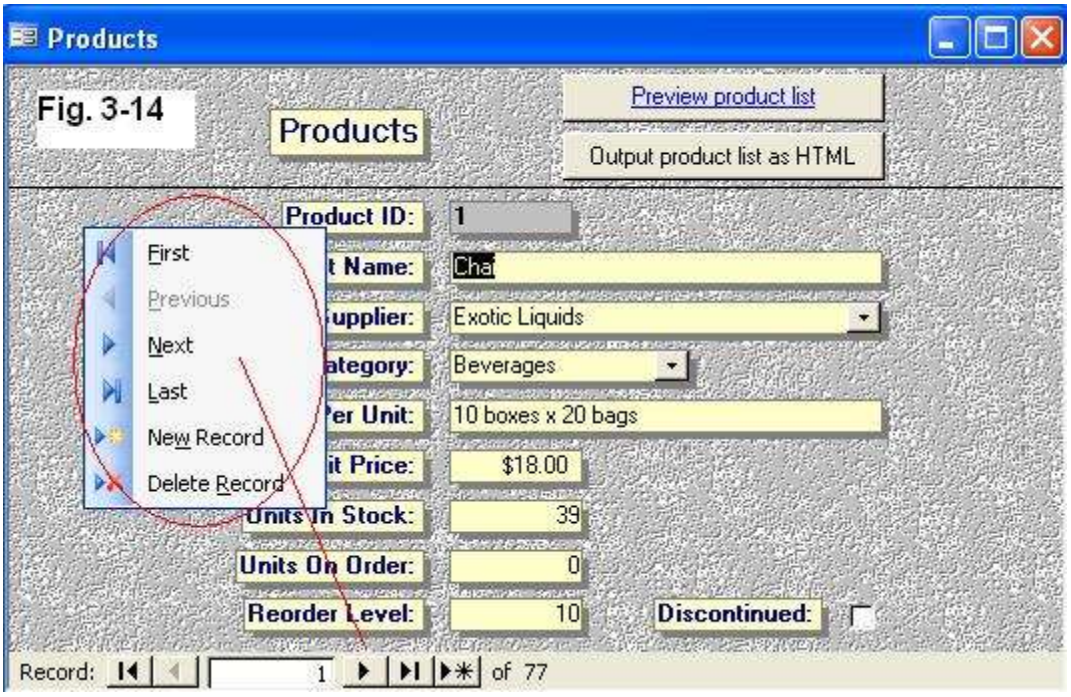
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

5. Open the form **Products** in **Design View** and open the **Form Property** (see Fig. 3-13). Click the **All** tab in the **Form** property and look for **Shortcut Menu Bar**. Click inside the drop-down list; you will see your custom pop-up menu, **Menu Shortcut**, in the list. Select it and close the F**orm** property and save the **Products** form.



6. Open the form **Products** to test the pop-up menu. Fig. 3-14 shows the pop-up menu circled in red. The commands on the pop-up menu will work the same as the Record Navigator.



In terms of adding commands, making a shortcut menu is the same as making a menu bar. If you want to remove some items from the shortcut menu, open the **Customize** dialog window, as shown in Fig. 3-12e. Drag unwanted items out of the custom shortcut menu. However, deleting an entire shortcut menu is quite different compared to deleting a menu bar. To delete a menu bar, select the name of the toolbar or menu bar and click **Delete** in the

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Toolbars tab of the **Customize** dialog window. To delete a shortcut menu, follow these steps:

1. Open the **Customize** dialog box. Click **Properties**.
2. In the **Toolbar Properties** dialog box, select the name of the shortcut menu you want to delete, for example **Menu Shortcut**, as shown in Fig. 3-15. The **Type** box shows **Popup**. Change the **Type** from **Popup** to **Toolbar** in the drop-down list.

Fig. 3-15

3. Fig. 3-16 shows the **Type** as **Toolbar**, and the custom toolbar, **Menu Shortcut**, appears on the screen.

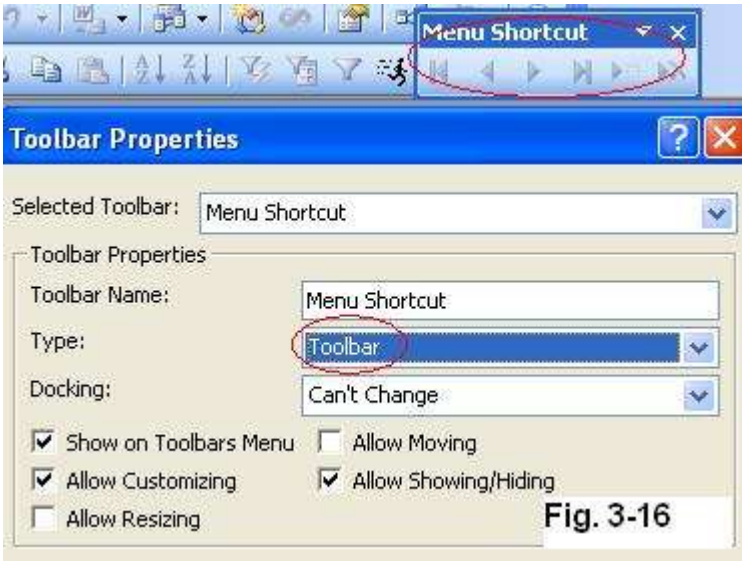


Fig. 3-16

4. Now, the shortcut menu is an ordinary toolbar. You can delete it by selecting this toolbar in the **Toolbars** tab of the **Customize** dialog box. Click **Delete** (see Fig. 3-17).

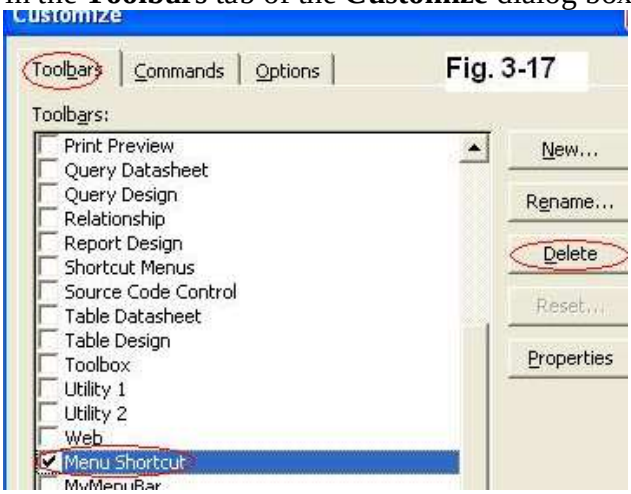


Fig. 3-17

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 3.6 Change the Startup settings

table of contents

How do you replace the Access default menu bar with your custom menu bar **MyMenuBar**? You need to change the circled settings in the **Startu**p window (refer to Fig. 3-18). Under **Tools** on the menu, select the **Startup** command to open the **Startup** dialog window. In the dialog window, click the **Menu Bar** drop-down list and select your custom menu bar **MyMenuBar**.

In the **Shortcut Menu Bar** drop-down list, you can see that the custom pop-up menu is available for selection. If you use **Menu Shortcut** instead of the default one, you will only see this custom shortcut menu every time you right-click the mouse. It is not a good idea to replace the default because default pop-up menus can vary from object to object. A list of pop-up menus is shown at the top of Fig.3-12e. You may want to design different pop-up menus for each object, according to your specific needs, and then add the custom pop-up menu to individual objects' properties. If the shortcut menu bar property is blank, then the default shortcut menu will be used.

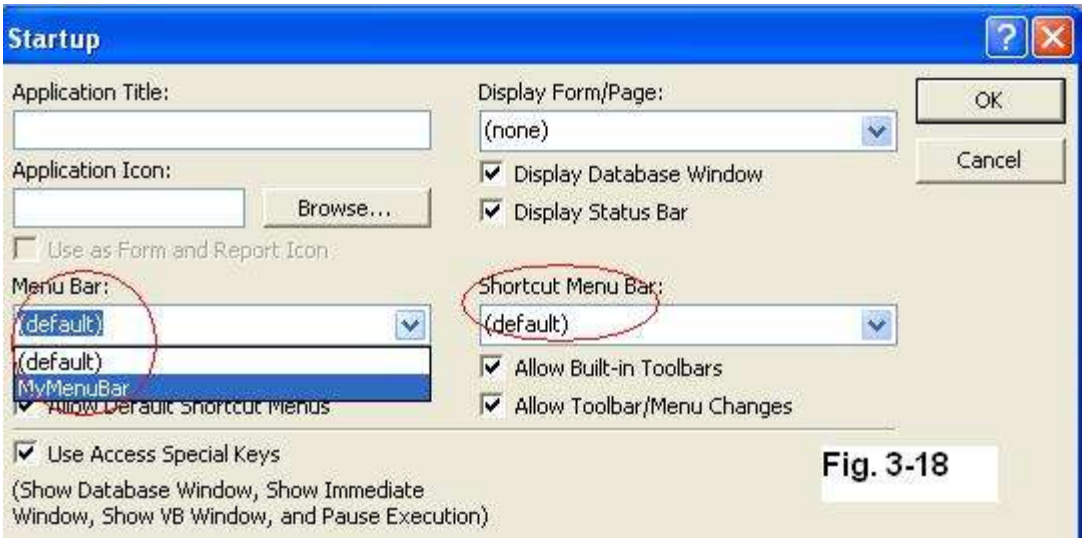
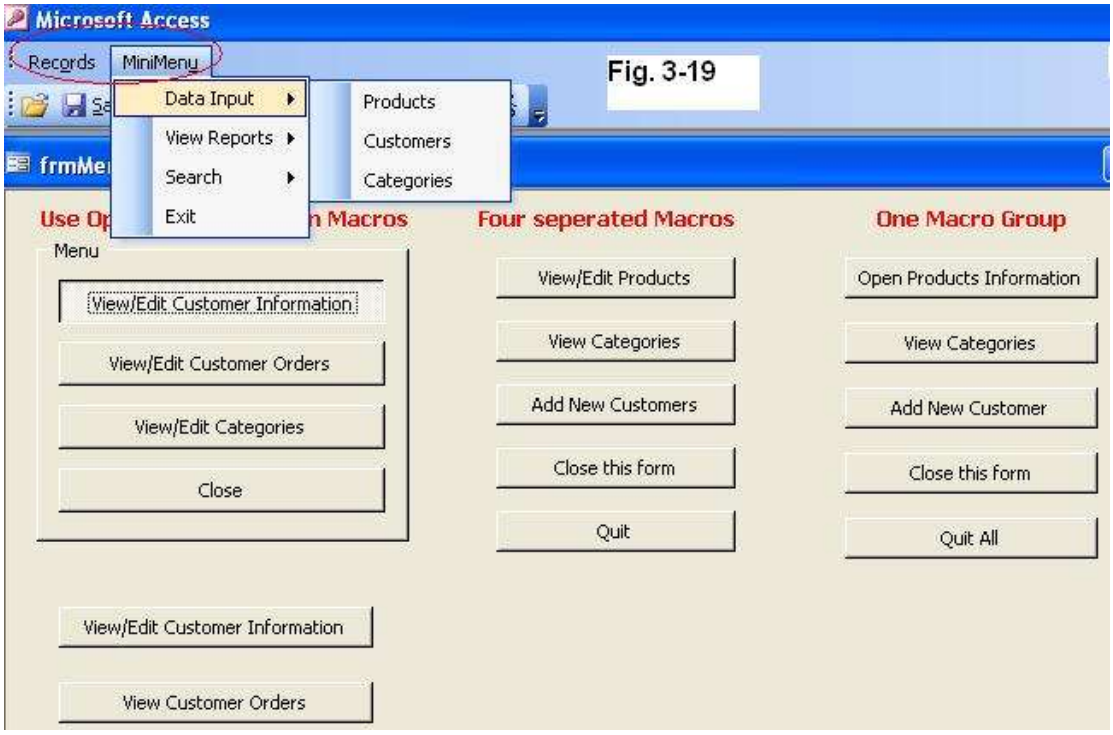


Fig. 3-18

You will keep the **Shortcut Menu Bar** as default on the **Startup** dialog window. Uncheck all checkboxes. Select the form **frmMenu** from **Display Form/Page**. Click OK to close the **Startup** dialog window. Close Access and **NWCh03**, then reopen **NWCh03** to test it. Fig. 3-19 shows the custom menu bar and toolbar. You can hold the **Shift** key when you open **NWCh03** if you want to be in the design environment.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Chapter 4 Using Code Builder

table of contents

Before getting to the last builder in Access, **Code Builder,** you need to know common control structures used in Access Visual Basic for Application (VBA) codes. VBA provides two basic types of control structures, decision structures and iteration structures.

Flavor 4.1 Decision Structures: IF

table of contents

From the name decision structures, you know this structure executes statements depending on specific conditions. Two basic structures, **If…Then** and **If…Then…Else**, will be discussed.

A complete syntax structure of **If…Then…Else** is as follows:

If *logical-test* **Then**
    statement-A if logical-test is true
Else
    (*statement-B if logical-test is false)*
End If

For example:

```
If State="IL" Then
    TaxRate=0.075
Else
    TaxRate=0
End If
```

The above code will change the field **TaxRate** value to 0.075 if the field **State** contains **IL** otherwise the field **TaxRate** has a value of 0 for other states.

If the *logical-test* is true, you will not need *statement-B*; if it is false, no action will be taken, making *statement-B* unnecessary. This kind of **IF** syntax structure is:

If *logical-test* **Then**
    statement-A if logical-test is true
End If

For example:

```
If State="IL" Then
    lblmessage.Caption="Illinois"
End If
```

If the field **State** contains **IL**, the above code will display "**Illinois**" in the caption of label box **lblmessage**. For other states, there will be no change in the caption of the label box.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 4.2 Decision Structure: nested IF

table of contents

The concept of a nested **IF** structure puts an **IF** structure into another **IF** structure. You use the nested **IF** structure, **If…Then…ElseIf**, when you want to evaluate multiple conditions. The syntax of **If…Then…ElseIf** is as follows:

```
If logical-test1 Then
        statement-A if logical-test1 is true
ElseIf logical-test2 Then
        statement-B if logical-test2 is true
ElseIf logical-test3 Then
        statement-C if logical-test3 is true
.       .       .       .       .
Else
        last-statement
End If
```

For example:

```
If State="IL" Then
        TaxRate=0.075
ElseIf State="IN" Then
        TaxRate=0.065
ElseIf Stae="WI" Then
        TaxRate=0.055
Else
        TaxRate=0
End If
```

The above code will assign the variable **TaxRate** with the value 0.075 if the field **State** contains **IL**, with the value 0.065 if the field **State** contains IN**,** and 0.055 if the field **State** is **WI**. For other states, the **TaxRate** will be 0.

Flavor 4.3 Select Case structures

table of contents

The nested **IF** structure may not read easily for a lot of **ElseIf** structures. The **Select Case** statement is an alternate. It executes different codes depending on one variable's value. The syntax of a **Select Case** statement is as follows:

```
Select Case variable
        Case expression1
                statement-A if expression1=variable
        Case expression2
                statement-B if expression2=variable
        Case expression3
                statement-C if expression3=variable
        Case Else
                statement-else if none of the above is true
```

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

End Select

The keywords of the above given syntax are explained as:

Select Case: states the beginning of the structure
***Variable*:** stores a value used within the **Select Case** structure
***Expression*:** denotes the condition that tests the value stored in the variable
Case Else: represents the code that will be executed, if none of above cases is true
Statement: represents the code that will be executed, if the condition is true.
End Select: states the end of the construct.

For each expression, the statement will run if the condition is true (i.e. the variable and the specific expression match). Technically, the **Select Case** structure is not only easier to read and maintain, but is also executed faster than the **If…Then…ElseIf** structure when giving multiple tests for a single variable.

Let's rewrite the example shown in 4.2 in **Select Case** structure:

```
Select Case State
        Case "IL"
                TaxRate=0.075
        Case "IN"
                TaxRate=0.065
        Case "WI"
                TaxRate=0.055
        Case Else
                TaxRate=0
End Select
```

You can use a range of values by using two keywords: **IS** and **TO**:

```
Select Case StudentScore
        Case Is < 59
                ClassGrade="F"
        Case 60 To 69
                ClassGrade="D"
        Case 70 To 79
                ClassGrade="C"
        Case 80 To 89
                ClassGrade="B"
        Case Is > 90
                ClassGrade="A"
End Select
```

Based on the value of **StudentScore**, only one of the above expressions will be true and only one grade will be assigned to **ClassGrade**.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The above structures cannot be satisfied if you want to repeatedly execute a specific block of code. The next flavor provides structures for repeated executions for cases where the logical-test condition is true. There are two types of repeated structures: fixed iteration and indefinite iteration.

Flavor 4.4 Iteration structures: Fixed iteration

table of contents

Iteration is also called looping. It is a complete cycle of execution of statements in a loop. The first type of iterations, fixed iteration, executes predetermined blocks of codes for a certain number of times. In order to control the number of repeats, a counter will be used. Such a structure is called *counter-controlled* repetition.

For…Next

Use the **For…Next** loop when you want to execute a block of code for a set number of times.

The syntax of the **For…Next** loop is:

```
For counter = StartValue To EndValue  [Step increment]
        statements
    Next counter
```

The keywords of the above syntax are defined as:

Counter*:** denotes a variable that keeps track of the number of iterations. After each iteration, the value of ***counter increases by ***increment*** value.
***StartValue*:** represents the initial value of the counter, required.
***Increment*:** represents any fixed number specified for the iteration of the loop. If this step value is not specified, then the default value is one.
***EndValue*:** represents the end value of the counter,reauired
Statements: Represents the code that is executed in each iteration.
Next: increases the value of the counter and sends the control to the beginning of the loop. The counter after Next is optional.

For example:

```
For intCounter = 10   To 1   Step   -1
        Debug.Print intCounter
Next intCounter
```

The result shown in the window will be ten numbers run from 10 in descending order because of the negative increment. If you switch the start and end values and leave the increment step value out of the structure, you will still get ten numbers, but in ascending order from 1 because of the positive step value.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The **For Each…Next** loop is another similar fixed iteration structure you can use to execute on objects inside another object of a collection. The general syntax of the **For Each…Next** loop is as follows:

For Each object **In** collection
    statements
Next

For example, you might need to hide all buttons on a form. The number of iterations depends on the number of objects in the collection. **Dim** declares the variable and will be discussed later.

```
Dim ctl As Control
For Each ctl In Controls
    If TypeOf ctl Is CommandButton Then
        ctl.Visible = False
    End If
Next
```

Another example is that you might need to clear all textboxes on a form. To do this, you can use the **For Each…Next** loop in the **Activate** event of the form:

Dim ctl **As Control**
For Each ctl **In Controls**
    If Typeof ctl **Is TextBox Then**
        ctl.**SetFocus**
        ctl.**Text** = “”
    End If
Next

In this code, **ctl** is the collection object in the **Controls** collection. The **For Each…Next** loop iterates for each control in the form and clears the text if the control is a textbox.

Flavor 4.5 Iteration structures: Indefinite iteration

table of contents

The indefinite iteration structure executes a set of statements until a specific condition occurs (i.e. equals something). It is also called *sentinel-controlled* repetition. Typically, there are two different structures: pretest type **Do Loop** (i.e. **Do While**, **Do Until)** and posttest type **Loop (**i.e. **Loop While**, **Loop Until)**.

Pretest type **Do Loop** structure:

Do While…Loop

The **Do While…Loop** will evaluate the specified *condition* first before running the *statements*. While the evaluation is true, the *statements* will be executed again and again. It will not stop execution until the evaluation becomes false.
The syntax of the loop is as follows:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Do While *condition*
statements
Loop

The **Do** keyword starts the loop, the **While** keyword evaluates the *condition* to determine if the statements should be executed. The statements will be executed if the evaluation is true; otherwise, it will skip the entire block of statements and execute the code after the **Loop** keyword. The **Loop** keyword starts another cycle. You can use variables in the *condition*.
For example:

```
x=1
Do While x <= 10
        Debug.Print x
        x=x+1
Loop
```

In this loop, it will only display an x value from 1 to 10. When x becomes 11, the loop will stop execution because the condition is false.

Do Until…Loop

The **Do Until…Loop** is also a pretest type loop structure. It will also evaluate the specific *condition* first before running the *statements*. However, in terms of the evaluation of *condition* it is different from **Do While… Loop** structure. It uses reverse logic determination. As long as the evaluation is false, the statements will be executed repeatedly. It will not stop execution until the evaluation becomes true.
The syntax of the **Do Until…Loop** is as follows:

Do Until *condition*
statements
Loop

The **Do** keyword starts the loop. The **Until** keyword evaluates the *condition* to determine if the statements should be executed. If the evaluation is false, the statements will be executed. If true, it will skip the entire block of statements and execute the code after the **Loop** keyword. The **Loop** keyword turns the flow to the beginning of the loop to start another cycle. You can use variables in the *condition*.

For example:

```
x=1
Do Until x > 10
        Debug.Print x
        x=x+1
Loop
```

This example will display the same result, 1 to 10, like the example in **Do While….Loop**. From the above examples, notice that as long as you use the correct *condition* in the structure, you will get the same outcome, regardless of which **Do Loop** structure you use.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Posttest type: **Loop** structure

Do….Loop While

The **Do….Loop While** structure places the *condition* at the end of the loop, allowing statements to be executed at least once. The syntax of the loop is:

```
Do
        statements
    Loop While condition
```

The *statements* will be executed as long as the *condition* is true.

For example:

```
x=1
Do
        Debug.Print x
        x=x+1
Loop While x < 1
```

Since the loop's condition is executed last, the statement inside the loop is guaranteed to be executed at least once.

Do…..Loop Until

Using reverse logic from the **Do…..Loop While** loop, the **Do …Loop Until** loop will execute the *statements* at least once regardless of the *condition*. It will stop execution when the *condition* becomes true. The syntax of the loop is as follows:

```
Do
        statements
Loop Until condition
```

For example:

```
x=1
Do
        Debug.Print x
        x=x+1
Loop Until x > 1
```

Using an increment statement, this loop's *statements* will be executed only once. More specifically, the loop will iterate until x is greater than 1, which meets the loop's exit condition of true.

Generally, for pretest type loops, you need to initialize the variable before the loop because the condition is evaluated first. On the other hand, if you specify the condition at the end

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

(posttest typed loop), along with the **Loop** keyword, the variable needs to be initialized within the loop.

Later on, you will learn how to test the above codes by using the **Code Builder** in the **On Click** event of a command button. You can step through it, one iteration at a time, with each click of the command button.

Flavor 4.6 With,,,End With

table of contents

This structure executes a series of codes on a single object. For example, you want to format a textbox with a list of properties, including font, size, BackColor, and ForeColor. The syntax is as follows:

With *object*
statements
End With

The components of **With…End With** are:
object: required. Name of an object or a user-defined type.
statements: optional. One or more statements to be executed on *object*

For example:

```
With MyLabel
                    .Height = 2000
                    .Width = 2000
                    .Caption = "This is MyLabel"
End With
```

There is a dot character (period) before each property. This dot character is required. Once a **With** block is entered, the *object* can't be changed. A block of codes between **With** and **End With** only affect this single object. As a result, you can't use a single **With** statement to affect multiple objects. A possible solution is to nest **With** statements by placing one **With** block within another. However, you should be careful because members of outer **With** blocks are masked within the inner **With** blocks. You must provide a fully qualified object reference in an inner **With** block for any member of an object in an outer **With** block. Here is an example of the nested **With**:

```
With MyObject
        .Height = 100        ' Same as MyObject.Height = 100.
        .Caption = "Hello World"    ' Same as MyObject.Caption = "Hello World".
   With .Font
        .Color = Red        ' Same as MyObject.Font.Color = Red.
         .Bold = True       ' Same as MyObject.Font.Bold = True.
   End With
End With
```

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

There are at least two benefits from using the above structure. First, there is less typing because you don't need to retype the object name for each task. The second benefit is that this kind of code will be executed more efficiently, because the object is referred to once rather than multiple times.

Flavor 4.7 Common Built-in String-Based Functions

table of contents

UCase

The **UCase** function converts a string parameter to a string in uppercase. Here is the syntax:

UCase(*string*)

The required *string* argument is any valid string expression. If the *string* contains **Null**, **Null** is returned. Only lowercase letters are converted to uppercase; all uppercase letters and non-letter characters remain unchanged.

For example:
txtOutput.Value = **UCase**(txtInput.Value)

The string in the **txtInput** textbox will be converted to uppercase and be displayed in the textbox **txtOutput**.

LCase

The **LCase** function is a reverse function of **UCase**. It takes a string parameter and outputs the string in lowercase. The syntax is as follows:

LCase(*string*)

Like the **UCase**, the required *string* argument is any valid string expression and when the *string* contains **Null**, **Null** is returned. Only uppercase letters are converted to lowercase; all lowercase letters and non-letter characters remain the same.

For example:

txtOutput.Value = **LCase**(txtInput.Value)

The string in the **txtInput** textbox will be converted into lowercase and displayed in the textbox **txtOutput**.

Len

The **Len** function will determine the length of a string and return the number of bytes required to store a variable. This value indicates the number of characters present in the string parameter, which includes spaces. The syntax is:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Len(*string* | *varname*)
The **Len** function syntax has these parts:
String: any valid string expression. If *string* contains **Null**, **Null** is returned.
Varname **:** any valid variable name. If *varname* contains **Null**, **Null** is returned. If *varname* is a **Variant**, **Len** treats it the same as a **String** and always returns the number of characters it contains.

For example:
txtLastName text box contains **Smith**.

Len(txtLastName)=6

Right

The **Right** function has two parameters and returns a string containing the number of characters from the right side of a string. The first parameter is the string to be evaluated, the second is a number that indicates how many characters to return from the right side of the string. Its syntax is:

Right(*string*, *n*)

Below are explanations of the arguments in the **Right** function syntax:
string: required. String expression from which the rightmost characters are returned. If ***string*** contains **Null**, **Null** is returned.
n: required; **Variant** (**Long**). The numeric expression indicates how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in ***string***, the entire string is returned.

For example:
A textbox **txtInput** contains **Smith**.

Right(txtInput.**Value**, 3) will return **ith**.
Right(txtInput.**Value**, 7) will return **Smith**.
Right(txtInput.**Value**, 0) will return zero-length string if you use Len function you will get 0.

Left

The **Left** function works like the **Right** function, except the direction of return of the string is opposite. It extracts a predetermined number of characters from the left side of a string. The first parameter is the string to be evaluated. The second parameter is a number, which indicates how many characters to return from the left side of the string. Its syntax is as follows:

Left(*string*, *n***)**

The **Left** function syntax has these named arguments**:**
string**:** required. String expression from which the leftmost characters are returned. If *string* contains **Null**, **Null** is returned.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

n: required. A numeric expression indicates how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *string*, the entire string is returned.

For example:
A textbox **txtInput** contains **Smith**.

Left(txtInput.**Value**, 3) will return **Smi**.
Left(txtInput.**Value**, 7) will return **Smith**.
Left(txtInput.**Value**, 0) will return zero-length string if you use **Len** function you will get 0.

Mid

The **Mid** function takes three parameters, of which the first two are required. The first parameter is the string to be evaluated. The second indicates where the function will begin counting the characters for its return value. The last parameter, which is optional, is the number of characters to be returned. If the last parameter is omitted, all characters from the starting position to the end of the string are returned. Its syntax is:

Mid(*string, start[, n]*)

These named arguments are explained as:

string: required. String expression from which characters are returned. If ***string*** contains **Null**, **Null** is returned.
start: required. The character position in ***string*** at which the part to be taken begins. If ***start*** is greater than the number of characters in ***string***, **Mid** returns a zero-length string ("").
n: optional. It indicates how many characters to return. If omitted or if there are fewer than ***n*** characters in the text (including the character at ***start***), all characters from the ***start*** position to the end of the string are returned.

For example:
A textbox **txtInput** contains **Smith**.

Mid(txtInput.**Value**, 3, 2) will return **it**.
Mid(txtInput.**Value**, 3, 4) will return **ith**.
Mid(txtInput.**Value**, 7) will return zero-length string if you use **Len** function you will get 0.

Str

The **Str** converts any numeric value into a **Variant** (**String**) representation of a number. Its syntax is:

Str(*number*)

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The required *number* argument is any valid, numeric expression. When numbers are converted to strings, a leading space is always reserved for the sign of the *number*. If the *number* is positive, the returned string contains a leading space and the plus sign is implied.

Use the **Format** function to convert numeric values you want formatted as dates, times, currency, or in other user-defined formats. Unlike **Str**, the **Format** function doesn't include a leading space for the sign of the *number*. The **Str** function recognizes only the period (**.**) as a valid decimal separator.

For example:
MyString = **Str(**459**)** ' Returns " 459", four characters as string data type.
MyString = **Str(**-459.65**)** ' Returns "-459.65", 7 characters as string data type.
MyString = **Str(**459.001**)** ' Returns " 459.001", 8 characters as string data type.

StrReverse

The **StrReverse** returns a string in which the character order of a specific string is reversed. The syntax is as follows:

StrReverse(*expression***)**

The *expression* argument is the string whose characters are to be reversed. If the *expression* is a zero-length string (""), a zero-length string will be returned. If the *expression* is **Null**, an error occurs.

For example:
StrReverse(“Good Morning”)
It returns gninroM dooG.

InStr

The **InStr** function can take up to four parameters, and returns a number specifying the starting position of the first occurrence of a string within another string. The required parameters are two strings, where the first is the string being searched, the second is the string that is sought.

The optional parameters determine the starting position of the search and the type of string comparison to be made. The syntax for the **InStr** is:

InStr([*start*,]*string1*, *string2*[, *compare*])

The above arguments used in the **InStr** function are defined as:

start: optional. Numeric expression that sets the starting position for each search from left side of string. If omitted, search begins at the first character position. If ***start*** contains Null, an error occurs. The ***start*** argument is required if ***compare*** is specified.
string1: required. String expression being searched.
string2: required. String expression sought.
compare: optional. Specifies the type of string comparison. If ***compare*** is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

For example:

```
SearchString ="XXpXXpXXPXXP"    ' String to search in.
SearchChar = "P"    ' Search for "P".

' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)

' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)

' Comparison is binary by default (last argument is omitted).
MyPos = Instr(SearchString, SearchChar)    ' Returns 9.

MyPos = Instr(1, SearchString, "W")    ' Returns 0.
```

InStrRev

This function, **InStrRev**, starts searching from the right side of a string and returns to the position where one string is found within another string. This position value is counted from the left side of a string. Its syntax is as follows:

InStrRev(*stringcheck, stringmatch[, start[, compare*]]**)**

The **InStrRev** function syntax has these named arguments:

stringcheck*:* Required. String expression being searched.
stringmatch*:* Required. String expression being searched for.
start: Optional. It is a numeric expression that sets the starting position for each search from the left side of *stringcheck*. If omitted, –1 is used, which means that the search begins at the last character position. If ***start*** contains Null, an error occurs.
compare: Optional. A numeric value indicates the kind of comparison to use when evaluating substrings. If omitted, a binary comparison is performed.

The ***compare*** argument can have the following values:

Constant	Value	Description
vbUseCompareOption	–1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Return Values

InStrRev returns the following values:

If	InStrRev returns
stringcheck is zero-length	0
stringcheck is **Null**	**Null**

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

stringmatch is zero-length	*start*
stringmatch is **Null**	**Null**
stringmatch is not found	0
stringmatch is found within ***stringcheck***	Position at which match is found
start > **Len(*stringmatch*)**	0

For example:

```
SearchString = "XXpXXpXXPXXP"   ' String to search in.
SearchChar = "P"    ' Search for "P".

' A textual comparison starting at position 12. Returns 12.
'MyPos = InStrRev(SearchString, SearchChar)
```

Replace

Simply speaking, the above function replaces one string with another. Its syntax is:

Replace(*expression, find, replace*[*, start*[*, count*[*, compare*]]])

The **Replace** function syntax names these arguments:
expression Required. String expression containing substring to replace.
find Required. Substring being searched for.
replaceRequired. Replacement substring.
start Optional. Position within ***expression*** where substring search is to begin. If omitted, 1 is assumed.
count Optional. Number of substring substitutions to perform. If omitted, the default value is –1, which means make all possible substitutions.
compare Optional. Numeric value indicating the kind of comparison to use when evaluating substrings.
The ***compare*** argument can have the following values:

Constant	Value	Description
vbUseCompareOption	–1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Replace returns the following values:

If	Replace returns
expression is zero-length	Zero-length string ("")
expression is **Null**	An error.
find is zero-length	Copy of ***expression***.
replace is zero-length	Copy of ***expression*** with all occurrences of *find* removed.
start > **Len(*expression*)**	Zero-length string.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

count is 0	Copy of ***expression***.

The return value of the **Replace** function is a string, with substitutions made, that begins at the position specified by ***start*** and concludes at the end of the ***expression*** string. It is not an identical copy of the original string.

For example:

Replace("This is lesson one.", "one", "two")

This function will return "This is lesson two".

Replace("This is lesson one.", " ", "") 'remove all spaces

This function will return "Thisislessonone.". All spaces are removed.

StrComp

The **StrComp** function is used to compare the sequence of characters in two strings. The **Option Compare** statement is used to determine whether a binary or textual comparison has been performed. If a binary comparison is done, characters are case-sensitive. Textual comparisons are not case-sensitive. The synax for the **StrComp** is:

StrComp(*string1, string2[, compare*])

These arguments are:
string1: required. Any valid string expression.
string2: required. Any valid string expression.
compare: optional. Specifies the type of string comparison. If the ***compare*** argument is **Null**, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

The **compare** argument settings are:

Constant	Value	Description
vbUseCompareOption	-1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

The **StrComp** function has the following return values:

If	StrComp returns
string1 is less than ***string2***	-1
string1 is equal to ***string2***	0

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

string1 is greater than ***string2*** 1
string1 or ***string2*** is **Null** **Null**

The follwoing example uses the **StrComp** function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```
MyStr1 = "ABCD"
MyStr2 = "abcd"    ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)    ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)    ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)    ' Returns 1.
```

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 4.8 Common built-in date and time functions

table of contents

DateAdd

This function returns a date to which a specified time interval has been added. Its syntax is as follows:

DateAdd(*interval, number, date***)**

The **DateAdd** function syntax has the following named arguments:

interval: required. String expression that is the interval of time you want to add.

number: required. Numeric expression that is the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).

date: required. **Date** or literal representing date to which the interval is added.

The ***interval*** argument has these settings:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

Use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 10 days from today, or a time 30 minutes from now. To add days to ***date***, you can use Day of **Year ("y")**, **Day ("d")**, or **Weekday ("w")**.

The **DateAdd** function won't return an invalid date. The following example adds one month to January 31, 2006:

DateAdd("m", 1, "31-Jan-06")

In this case, **DateAdd** returns 28-Feb-06, not 31-Feb-06. The format of the return value for **DateAdd** is determined by control panel settings, not by the format that is passed in the ***date*** argument.

DatePart

This function returns a **Variant** (**Integer**) containing the specified part of a given date. Here is its syntax:

DatePart(*interval, date[,firstdayofweek[, firstweekofyear]])*

The **DatePart** function syntax has these named arguments:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

interval: required. String expression that is the interval of time you want to return.
date: required. **Variant** (**Date**) value that you want to evaluate.
firstdayofweek: optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
firstweekofyear: optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

The ***interval*** argument is the same as the interval in **DateAdd** function.

The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The ***firstweekofyear*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.
vbFirstFullWeek	3	Start with first full week of the year.

You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour. The ***firstdayofweek*** argument affects calculations that use "w" and "ww" interval symbols. If ***date*** is a date literal, the specified year becomes a permanent part of that date. However, if ***date*** is enclosed in double quotation marks (" "), and you omit the year, the current year is inserted in your code each time the ***date*** expression is evaluated. This makes it possible to write codes that can be used in different years.

DateDiff

This function returns a **Variant** (**Long**) specifying the number of time intervals between two specified dates. Its syntax is:

DateDiff(*interval, date1, date2[, firstdayofweek[, firstweekofyear*]])

The **DateDiff** function syntax has these named arguments:

interval: required. String expression that is the interval of time you use to calculate the difference between ***date1*** and ***date2***.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

date1, ***date2***: required; **Variant** (**Date**). Two dates you want to use in the calculation.
firstdayofweek: optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
firstweekofyear: optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

The ***interval*** argument is the same as in the function **DateAdd**.

The ***firstdayofweek*** argument is the same as in the the function **DatePart**.

The ***firstweekofyear*** argument is the same as in the function **DatePart**.

The **DateDiff** function is used to determine how many specified time intervals exist between two dates. For example, you could use **DateDiff** to calculate the number of weeks between today and the end of the year.
If ***date1*** or ***date2*** is a date literal, the specified year becomes a permanent part of that date. However, if ***date1*** or ***date2*** is enclosed in double quotation marks (" "), and you omit the year, the current year is inserted in your code each time the ***date1*** or ***date2*** expression is evaluated. This way, your code can be used in multiple years. When comparing December 31 to January 1 of the immediately succeeding year, **DateDiff** for Year ("yyyy") returns 1 even though only a day has elapsed.

Date/Time

Access VBA contains numerous date/time functions such as **Date**, **Time**, and **Now** for accessing your system's date and time. Specifically, the following VBA date/time functions will be discussed:
Date
Day
Weekday
Month
Year
Time
Second
Minute
Hour
Now
With these functions, you can create date/time stamps, stopwatches, clocks, or custom timers.

Date

The **Date** function requires no parameter when executed and returns a **Variant** data type containing your system's current date.

For example:
MyDate = **Date** ' MyDate contains the current system date.

Day

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The **Day** function takes a required date argument (output of the **Date** function) and returns a whole number between 1 and 31, which represents a day within the current month. For example:

```
MyDate = #February 12, 1969#    ' Assign a date.
MyDay = Day(MyDate)    ' MyDay contains 12.
```

WeekDay

The **WeekDay** function takes two parameters and returns a whole number containing the current day of the week. The first parameter, which is optional, determines the first day of the week. The default first day of the week is Sunday. The syntax is:
Weekday(*date*, [*firstdayofweek*])
The arguments are described as:

date: required. Variant, numeric expression, string expression, or any combination, that can represent a date. If ***date*** contains Null, **Null** is returned.
firstdayofweek: optional. A constant that specifies the first day of the week. If not specified, **vbSunday** is assumed.
The ***firstdayofweek*** argument is the same as in **DatePart**.
The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

```
MyDate = #February 12, 2006#   ' Assign a date.
MyWeekDay = Weekday(MyDate)            ' MyWeekDay contains 1 because
                                       ' MyDate represents a Sunday.
```

Month

The **Month** function takes a single parameter, which signifies the current date, and returns a whole number representing the current month in the year. The following is its syntax:

Month(Date) ‘ Returns a number between 1 and 12.

For example:

```
MyDate = #February 12, 2006#    ' Assign a date.
MyMonth = Month(MyDate)    ' MyMonth contains 2.
```

Year

Similar to the preceding date-based functions, the **Year** function takes a date parameter and returns a whole number representing the current year. The syntax is as follows:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Year(Date)

For example:

```
MyDate = #February 12, 2006#    ' Assign a date.
MyMonth = Year(MyDate)    ' MyMonth contains 2006.
```

Time

The **Time** function is another function that requires no parameters as input. It returns a **Variant** data type with your system's current time.

```
MyTime = Time    ' Return current system time.
```

When used with other functions and events, the **Time** function can be quite useful in building applications. For example, you can use a form's **Timer** event and **TimerInterval** property to display a splash screen while your application is loading.

Second

The **Second** function requires a time parameter (the output from the **Time** function) and returns a whole number from 0 to 59 indicating the current second of the current minute. Its syntax is as follows:

Second(Time) ' Returns a number from 0 to 59.

For example:

```
MyTime = #4:35:17 PM#    ' Assign a time.
MySecond = Second(MyTime)    ' MySecond contains 17.
```

Minute

Similar to the **Second** function, the **Minute** function requires a time parameter (the output from the **Time** function) and returns a whole number from 0 to 59, indicating the current minute of the current hour. The syntax for the **Minute** function is:

Minute(Time) ' Returns a number from 0 to 59.

For example:

```
MyTime = #4:35:17 PM#    ' Assign a time.
MyMinute = Minute(MyTime)    ' MyMinute contains 35.
```

Hour

Similar to the **Second** and **Minute** function, the **Hour** function takes a required time parameter and returns a whole number between 0 and 23, which represents the current hour, according to your system's time. Its syntax is as follows:

Hour(Time) ' Returns a number from 0 to 23.

For example:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

MyTime = #4:35:17 PM# ' Assign a time.
MyHour = **Hour(**MyTime**)** ' MyHour contains 16.

Now

The **Now** function incorporates results from both **Date** and **Time** functions. It takes no parameters and returns a **Variant** data type indicating the system's current date and time, respectively.

For example:
Today = **Now** ' Assign current system date and time.

Flavor 4.9 Common built-in conversion functions

table of contents

Conversion functions are used to convert data from one type to another. Common uses for data conversion involve converting strings to numbers and numbers to strings.

Val

The **Val** function takes a string as input and coverts recognizable numeric characters to a numeric data type. The following are some sample return values:

Val(" 123 ") ' Returns 123
Val(" a123 ") ' Returns 0
Val(" 123a ") ' Returns 123

Chr

Character codes are numbers, which represent a single character. For example, the character "A" is represented by the character code 65, and its lowercase character, "a", is represented by the character code 97.
To convert a character code to its corresponding character, VBA programmers use the **Chr** function. The **Chr** function takes a single character code as a parameter and returns its corresponding character. Note that character codes also represent numeric characters and nonprintable characters such as space, tab, and linefeed. Its syntax is as follows:

Chr(*Charcode*)

For example:
MyChar = **Chr(**65**)** ' Returns A.
MyChar = **Chr(**97**)** ' Returns a.
MyChar = **Chr(**62**)** ' Returns >.
MyChar = **Chr(**37**)** ' Returns %.

Asc

The **Asc** function is the inverse of the **Chr** function. It takes a single character as input and converts it to its corresponding character code.

Asc(*string*)

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

For example:

```
MyNumber = Asc("A")    ' Returns 65.
MyNumber = Asc("a")    ' Returns 97.
MyNumber = Asc("Apple")    ' Returns 65.
```

Formatting

Data may need to be formatted so it will be displayed in a specific way. For example, you may want to display an eight-digit number with thousands-place commas, or numbers as a currency or percentage. Each of these scenarios, among others, can be accomplished with a single VBA function called **Format**. Its syntax is as follows:

Format(*expression, format, firstDayOfWeek, firstWeekOfYear*)

The parameters in the syntax are:

expression: required. Any valid expression.
format: optional. A valid named or user-defined format expression.
firstdayofweek: optional. A constant that specifies the first day of the week.
firstweekofyear: optional. A constant that specifies the first week of the year.

The ***firstdayofweek*** argument settings are the same as in function **DatePart**.
The ***firstweekofyear*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the year.
vbFirstFullWeek	3	Start with the first full week of the year.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The following is a list of format symbols used in the date, time, and year:

Format Character	Description
:	Separates time in hours, minutes and seconds
/	Separates dates in day, month, and year
d	Day displayed as a number without a leading zero
dd	Day displayed as a number with a leading zero
ddd	Day displayed as an abbreviation
dddd	Day displayed with the full name
ddddd	Complete date displayed in short format (m/d/yy)
dddddd	Complete date displayed in long format (mmm, dd, yyyy)
w	Day of the week displayed as a number (1 starts on Sunday)
ww	Week of the year displayed as a number
m	Month displayed as a number without a leading zero
mm	Month displayed as a number with a leading zero
mmm	Month displayed as an abbreviation
mmmm	Month displayed with full name
q	Quarters in year displayed as number
y	Day of the year is displayed as a number
yy	Year displayed in 2-digit format
yyyy	Year displayed in 4-digit format
h	Hour displayed without leading zeros
Hh	Hour displayed with leading zeros
N	Minute displayed without leading zeros
Nn	Minute displayed with leading zeros
S	Second displayed without leading zeros
Ss	Second displayed with leading zeros
ttttt	Time displayed with hour, minute, and second
AM/PM	Displays uppercase AM or PM using 12-hour clock
am/pm	Displays lowercase AM or PM using 12-hour clock
A/P	Displays an uppercase A or P using 12-hour clock
a/p	Displays a lowercase A or P using 12-hour clock
c	Displays the date as ddddd and time as ttttt

For example:

```
MyTime = #17:04:23#
MyDate = #January 27, 1993#

' Returns current system time in the system-defined long time format.
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MyStr = Format(Date, "Long Date")

MyStr = Format(MyTime, "h:m:s")    ' Returns "17:4:23".
```

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

```
MyStr = Format(MyTime, "hh:mm:ss AMPM")    ' Returns "05:04:23 PM".
MyStr = Format(MyDate, "dddd, mmm d yyyy")    ' Returns "Wednesday,
                                                ' Jan 27 1993".
' If format is not supplied, a string is returned.
MyStr = Format(23)    ' Returns "23".
```

Formatting Strings

There are five characters that can be used to build user-defined strings using the **Format** function. Each format character listed in table below must be enclosed in quotes when passed as a format expression argument in the **Format** function.
Note that placeholders are displayed from right to left unless an exclamation mark character is present in the format expression. Consult **Microsoft Visual Basic Help** for more information on the **Format** function and expressions.

Format Character	Description
@	Displays a character or space as a placeholder.
&	Displays a character or space as a placeholder.
<	Formats all characters in lowercase.
>	Formats all characters in uppercase.
!	Placeholders are filled left to right.

For example:

```
' User-defined formats.
MyStr = Format("HELLO", "<")    ' Returns "hello". Change to lower case.
MyStr = Format("This is it", ">")    ' Returns "THIS IS IT". Upper case.
```

Formatting Numbers

Numbers can be displayed with user-defined formatting expressions, as shown in the following table. More detailed examples can be found in **Microsoft Access Help**.

Format Character	Description
0	Displays a digit or zero as a placeholder.
#	Displays a digit or nothing as a placeholder.
.	A placeholder that determines how many digits are displayed to the left and right of the decimal.
%	Places the percentage character at the location where it appears in the format expression. Multiplies the number by 100.
,	Separates thousands from hundreds using the comma character.
E- E+ e- e+	Specifies scientific formatting.
-+$( )	Specifies a literal character.
\\	Displays a single backslash.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

For example:

```
' User-defined formats.
MyStr = Format(5459.4, "##,##0.00")    ' Returns "5,459.40".
MyStr = Format(334.9, "###0.00")    ' Returns "334.90".
MyStr = Format(5, "0.00%")    ' Returns "500.00%".
```

Flavor 4.10 Common built-in input validation functions

table of contents

IS Functions: IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject

For your application, you should build a way of validating the quality and accuracy of data entered by users. Prevention is a key to avoiding input error. The **IS** series function, a tool to inspect input values, returns a **Boolean** value of **True** if the variable or expression matches a type of validation, and **False** if it does not. Remember that VBA treats an empty textbox as an empty string value (“”), which is not **Null**, nor is it a number. The following are examples of some **IS** functions:

IsArray

```
YourArray = Array(1, 2, 3)    ' Use Array function.
MyCheck = IsArray(YourArray)    ' Returns True.
```

IsNumeric

```
MyVar = "53"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns True.
MyVar = "459.95"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns True.
MyVar = "45 Help"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns False.
```

IsDate

```
MyDate = "February 12, 1969"
YourDate = #2/12/69#
NoDate = "Hello"
MyCheck = IsDate(MyDate)    ' Returns True.
MyCheck = IsDate(YourDate)    ' Returns True.
MyCheck = IsDate(NoDate)    ' Returns False.
```

IsEmpty

```
MyCheck = IsEmpty(MyVar)    ' Returns True.
MyVar = Null    ' Assign Null.
MyCheck = IsEmpty(MyVar)    ' Returns False.
MyVar = Empty    ' Assign Empty.
MyCheck = IsEmpty(MyVar)    ' Returns True.
```

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

IsNull function to determine if a variable contains a **Null**.

```
MyCheck = IsNull(MyVar)    ' Returns False.
MyVar = ""
MyCheck = IsNull(MyVar)    ' Returns False.
MyVar = Null
MyCheck = IsNull(MyVar)    ' Returns True.
```

Flavor 4.11 Code Builder

table of contents

Common control naming conventions

A proper naming system can improve the readability and consistency of a code. Based on the names, you can easily identify the control type. In Access, you are suggested to use a naming convention called *Hungarian notation* (named after a computer scientist). Type the lowercase, three-letter prefix, followed by a meaningful description into the **Name** property of each control. Each word following the prefix should have its first letter capitalized. If you or your team are uncomfortable with using these notation rules, it is fine as long as there is a systemic naming rule in coding.

Here is a list of prefixes and examples for your reference:

Control	Prefix	Example
Check Box	chk	chkVisa
Combo Box	cbo	cboType
Command Button	cmd	cmdOpen
Form	frm	frmProduct
Image	img	imgLogo
Label	lbl	lblCompanyName
List Box	lst	lstCateory
Option Button	opt	optFirstPick
Text Box	txt	txtAddress

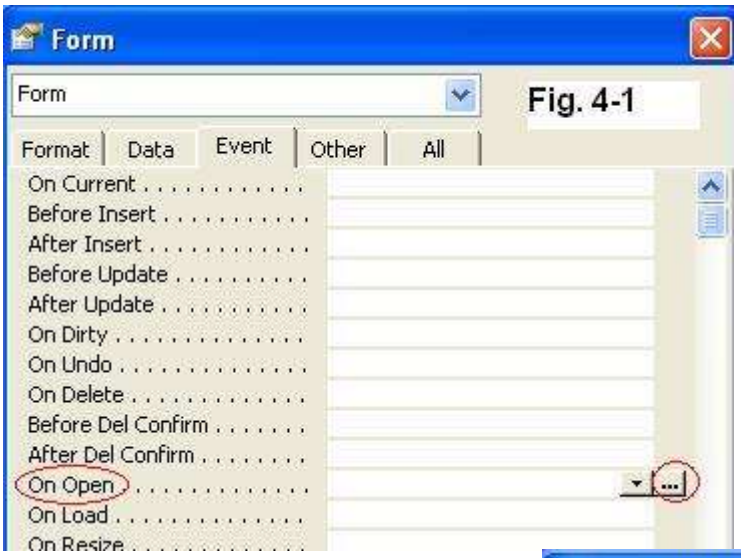
Attaching code with events

All Windows applications are event-driven, which means any activity is related to users' actions. You might create a series response to run in reply to a specific user action, such as clicking a command button on a form. You can do this by associating the code with an event of an object. The following example demonstrates how to create a series code to hide all command buttons when the object, **frmMenu,** opens**.**

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

There are several ways to code the above event. The first is to open the property dialog box of a specific object. In this example, the object is the form, **frmMenu,** and the event is **On Open**, as shown in Fig. 4-1.



Click to open the **Choose Builder** Windows.

Select **Code Builder** and click OK. A block of procedure is automatically created. The keyword **Dim** is used to declare the name and type of a variable. You may want to type keywords in lowercase. If these keywords don't become uppercase then it indicates that these are not correct keywords. This minimizes the chance of bugs in your codes. The **ctl** is a variable. Here are some naming rules you may want to use when naming a variable or constant in codes.

Must start with a letter
Cannot contain spaces, a period (.), or special symbols (%, &, !, #, @, $, +, -, or *)
Cannot be more than 255 characters long
Cannot be the same as a VBA keyword
Must be unique within a procedure or a module

Continue to type the keyword **AS** and a space. A list of objects/methods appears, like in Fig. 4-2. Access can complete the word you're typing by using the **Complete Word** feature. In order to apply this feature, you need to type a few characters for Visual Basic to recognize the word you want.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

```
Private Sub Form_Open(Cancel As Integer)
dim ctl as con
End Sub
```

Connection
Connections
Container
Containers
Control
Controls
Currency

Fig. 4-2

The list is displayed in alphabetical order. You can either right-click to select **Compete Word** from the pop-up menu; press Ctrl+Spacebar to complete the word; or select the word you want from the list and press the spacebar to complete it. Press enter to go to the next line. You will see the keywords capitalized. Enter the following code in lowercase (see Fig. 4-3):

For Each ctl In Controls
If TypeOf ctl Is CommandButton Then

```
Private Sub Form_Open(Cancel As Integer)
Dim ctl As Control
For Each ctl In Controls
    if typeof ctl is commandbutt
End Sub
```

CommandButton
Connection
Connections
Container
Containers
Control

Fig. 4-3

Press enter when the line of code is done. You should see the keywords, circled in red in Fig. 4-3, capitalized. Enter next line of code:

ctl.Visible = False
End If
Next

The complete codes are shown in Fig.4-4.

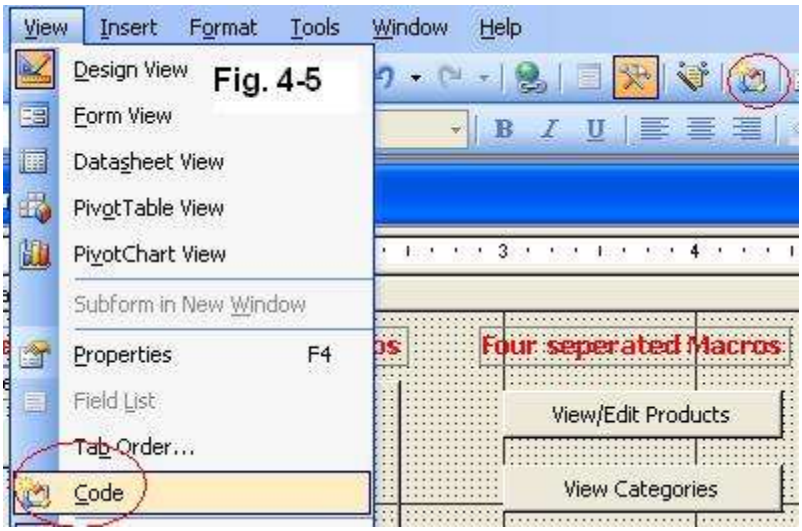
```
Private Sub Form_Open(Cancel As Integer)
Dim ctl As Control
For Each ctl In Controls
    If TypeOf ctl Is CommandButton Then
        ctl.Visible = False
    End If
Next

End Sub
```

Fig. 4-4

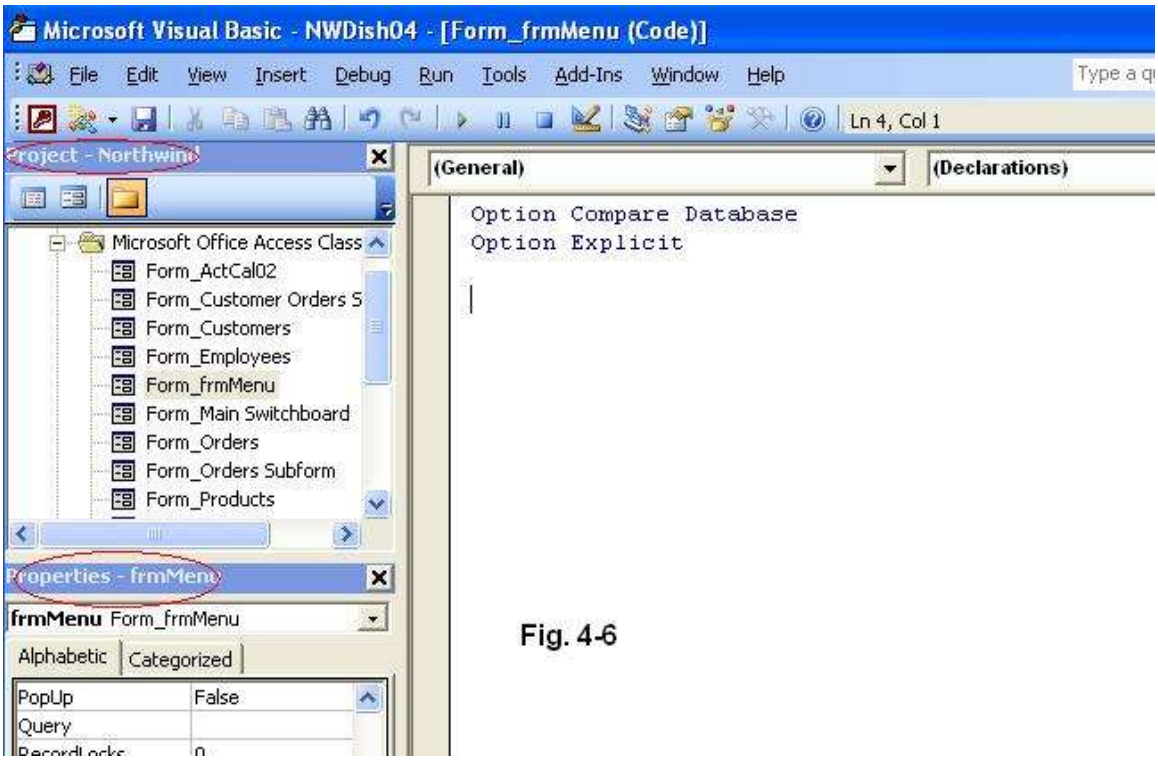
2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The second way to code the event is to open the form in **Design View**. From menu **View**, **Code** or click the toolbar, as shown in Fig. 4-5.

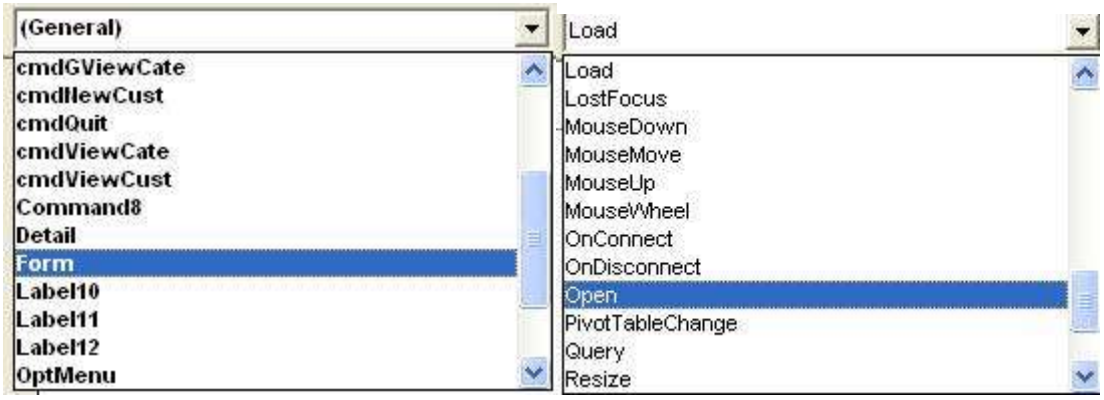


Next you will see a window called **Visual Basic Editor** (VBE) (Fig. 4-6).
The upper-left side of the window is the **Project** window, which lists all class objects (forms and reports) and modules. The lower-left side of the window is the **Properties** window, which lists the properties associated to the control objects on either a form or a report selected in the **Project** windows. The blank area is the place to write codes.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association
ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Click the drop-down list, which is **General** to start with, and select the object **Form** as illustrated below. You will get a block of procedure: **Form_load**. You will need to select the event **Open** from the procedure drop-down list.



You may want to delete the empty procedure **Form_load( )** and enter the codes in Fig. 4-4 into the procedure **Form_Open**. Close the VBE, then save and close the form.

Test the code by opening the form **frmMenu**. You will see only the toggle menu appear, as shown in Fig. 4-7. In this source code, you only made the command buttons invisible, so other objects, like labels and toggle buttons, are not affected.

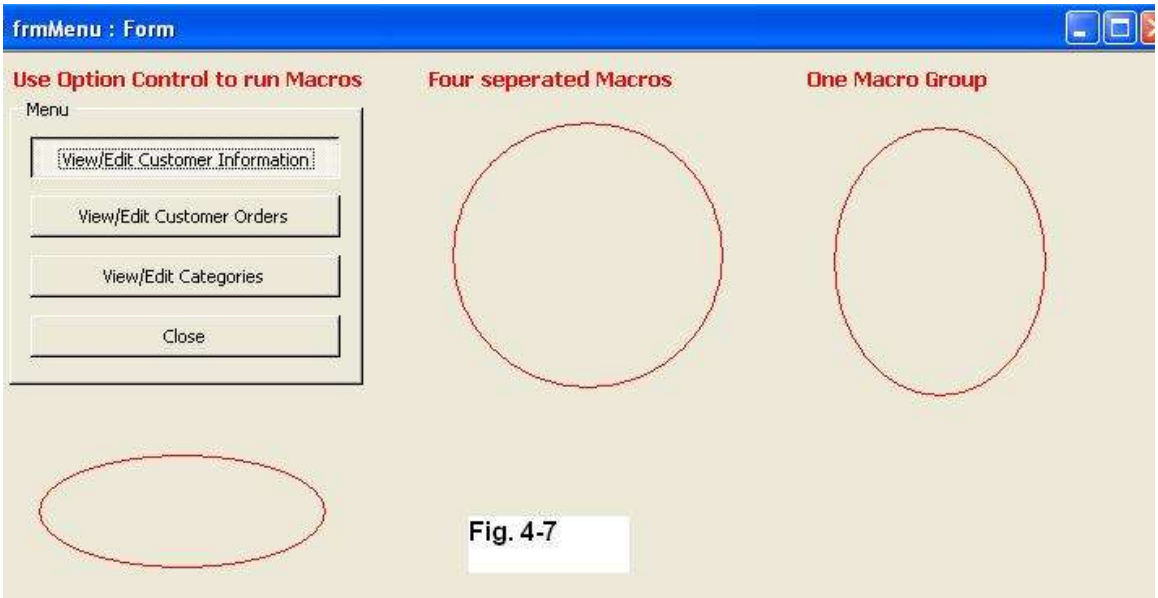


Fig. 4-7

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

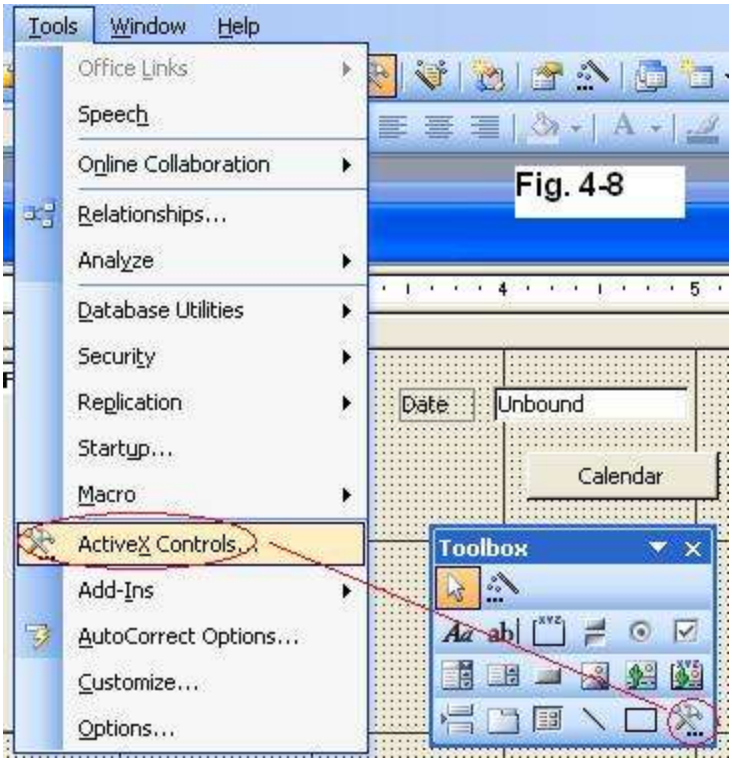
Flavor 4.12 How to use an ActiveX Control

table of contents

Microsoft renamed OLE controls as ActiveX controls. An ActiveX control can be identified by the file extension **.OCX**. Sometimes you will see people refer to ActiveX controls as OCXs. The following example uses a very common ActiveX control: calendar control.

Here is how to add an ActiveX control (OLE custom control) to a form:
Open a form (or report or data access page) in **Design View**.

Click the **More Controls** tool in the toolbox as shown in Fig. 4-8.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

If you need to register or unregister the ActiveX controls built in by a third-party, you can do so by opening the dialog window from **Tools** on the menu, then **ActiveX Controls** (see Fig. 4-8a).

Fig. 4-8a

From the toolbox, **More Controls**, click the calendar control (Fig. 4-8b).

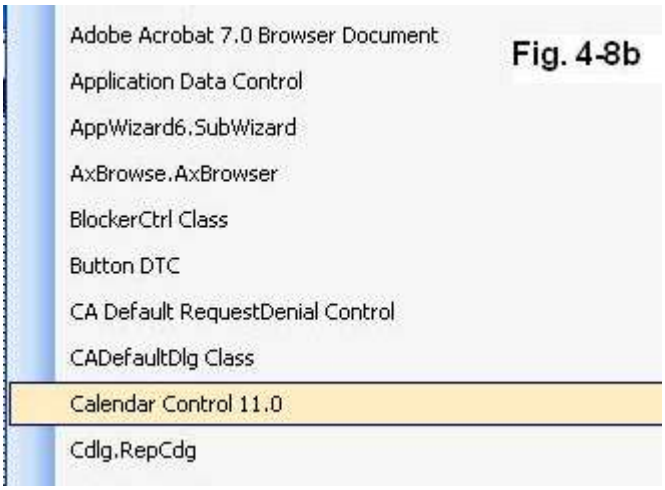
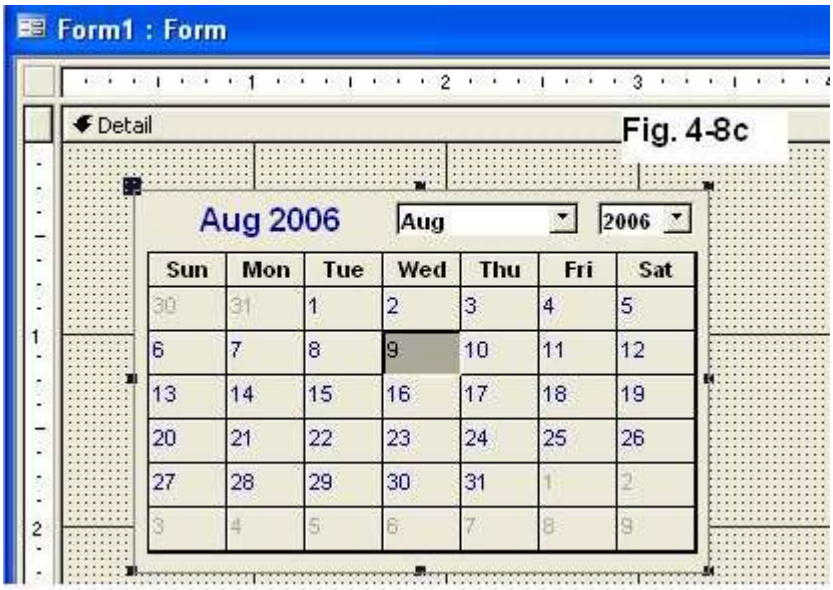


Fig. 4-8b

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Move the mouse pointer (you should see the pointer changed to an ax) to the **Detail** section and left-click the mouse to insert the calendar, like in Fig. 4-8c.



If ActiveX controls are installed in other applications, such as Microsoft Excel, they should appear if you click either the **More Controls** in the toolbox, or the ActiveX control command under **Insert** (Fig. 4-8d).

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Fig. 4-8e

You may want to consider making the calendar visible only when you need it. In order to make the above task work, you will code it on two different events. Add a textbox to the form's **Detail** section and name it **txtDate**, which will display the date selected from the calendar **ActCal**.

The properties of the calendar are:
Name: ActCal
Visible: No
When the form is first opened, the calendar will be hidden. Make sure you select **ActCal** from the object list before the **OnClick** event from the VBE window:

A blank procedure appears. Type the following line of codes:

```
Private Sub ActCal_Click()
txtDate = ActCal
txtDate.SetFocus
ActCal.Visible = False
End Sub
```

The **SetFocus** method will leave the blinking cursor in the textbox when the calendar is clicked. The date is transferred into the textbox and the calendar is hidden again.

The second event makes the calendar either shown or hidden. You will use a command button with the following properties:

Name: cmdCal
Event: On Click

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The codes behind the button are:

```
Private Sub cmdCal_Click()
If ActCal.Visible Then
    ActCal.Visible = False
Else
    ActCal.Visible = True
End If
End Sub
```

This command button is very similar to a toggle, which turns the calendar on (visible) or off (invisible) when you click the button. It uses the **IF** structure to determine the status of calendar.

For example, the calendar should be hidden when the form opens because the calendar visible property is set as **No**. Click the command button, **cmdCal**, to see the calendar. The calendar appears because the first **ActCal.Visible** (after **IF**) **Boolean** value is **False**. Then the code will execute the line after **Else** to make the calendar **ActCal** visible.

If the first **ActCal.Visible** (after **IF**) **Boolean** value is **True**, which means the calendar appears, then the second **ActCal.Visible** = **False** will be executed to hide the calendar when you click the command button.

Name this form **ActCal01** and save it.

Summary of events for objects
Click. Activated when a user presses and releases a left mouse button over the object.
DblClick. Triggered when a user double-clicks the object
Enter. Occurs before the object actually receives the focus
Exit. Occurs before the object actually loses the focus
GotFocus. Triggered when the object receives the focus, i.e. cursor stays
LostFocus. Triggered when the object loses the focus, i.e. cursor left
AfterUpdate. Occurs after data is changed in the control
BeforeUpdate. Occurs before data in the control is updated
MouseDown. Occurs when the user presses a mouse button on the object
MouseMove. Starts when a user moves the mouse while over the object
MouseUp. Occurs when the user releases a mouse button on the object from which a mouse button was clicked.

Textboxes

There are numerous means for retrieving input from users. The most common control used on the form is the textbox control. Textboxes can easily be added to Access forms. After you add a textbox in **Design View**, Access will automatically add a label box to the left of the textbox. Usually, the textbox needs an explanation as to what its content should be. The label box is often the perfect solution. Yet, if you wish to remove the label, simply click on the textbox's label once and delete it (press the **Del** key or right-click and choose **Delete**).

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Common Textbox Properties

Property	Description
BackColor	Determines the back color of the label
BackStyle	Determines normal back style or transparent
DecimalPlaces	Specifies the number of decimal places to display numbers
Enabled	Determines if the control is available for user interaction
EnterKeyBehavior	Specifies what happens when the Enter key is pressed within a text box
FontName	The font's name
FontSize	The size of the font displayed in the caption
ForeColor	Determines the forecolor of the label
Name	Internal control name
ScrollBars	Determines if the text box has scroll bars
TabIndex	Determines the control's tab order on the form
ValidationRule	Determines requirements used for data entered into the text box
ValidationText	Specifies message displayed to user if Validation Rule is not followed
Visible	Determines if the user can see the text box

InputBox function

The **InputBox** displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a string containing the contents of the textbox. Its syntax is:

InputBox(*prompt*[, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The **InputBox** function syntax has these named arguments:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return–linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
default	Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit ***default***, the text box is displayed empty.
xpos	Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If ***xpos*** is omitted, the dialog box is horizontally centered.
ypos	Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If ***ypos*** is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Below are some examples of the **InputBox**:

```
Message = "Enter a value between 1 and 3"    ' Set prompt.
Title = "InputBox Demo"    ' Set title.
Default = "1"    ' Set default.
' Display message, title, and default value.
MyValue = InputBox(Message, Title, Default)

' Use Helpfile and context. The Help button is added automatically.
MyValue = InputBox(Message, Title, , , , "DEMO.HLP", 10)

' Display dialog box at position 100, 100.
MyValue = InputBox(Message, Title, Default, 100, 100)
```

The next example uses textboxes, option control, tab control, combo boxes, checkboxes, and command boxes to integrate the structure from the beginning of this dish. Open the form **frmTestVBA** for reference.

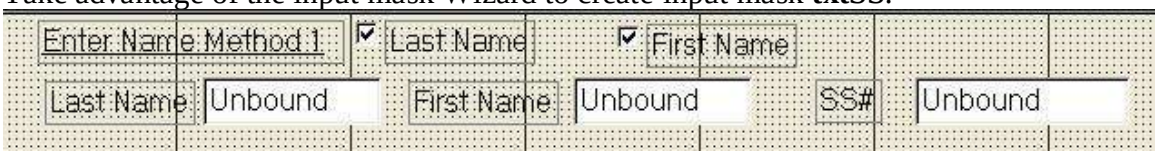
2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

1. Use two check boxes and input box to take the user's input and display it on text box. Here are the properties and values of each object:

Control Types	**Properties**	**Values**
Label	Caption	Enter Name Method 1
Checkbox	Name	chkFName
	Default value	False
	Event	On Click
	Visible	Yes
Label	Name	lblFName
	Caption	First Name
Check box	Name	chkLName
	Default value	False
	Event	On Click
	Visible	Yes
Label	Name	lblLName
	Caption	Last Name
Text box	Name	txtLName
Label	Caption	Last Name
Label	Caption	First Name
Text box	Name	txtFName
Label	Caption	SS#
Text box	Name	txtSS
	Input mask	000\-00\-0000;;_

Take advantage of the input mask Wizard to create input mask **txtSS**.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The source code of **chkLName**'s **On Click** event is as follows:

```
Private Sub chkLName_Click()
    If [chkLName] = True Then
        [txtLName] = InputBox("Please enter your Last Name here.", _
        "Last Name")
        If [txtLName] = "" Then
            txtLName.Visible = False
            txtSS.Visible = False
        Else
            txtLName.Visible = True
        End If
    Else
        txtLName.Visible = False
        txtSS.Visible = False
    End If
        If txtLName.Visible Then
            If txtFName.Visible Then
            txtSS.Visible = True
            [txtSS] = ""
            txtSS.SetFocus
            End If
        End If
End Sub
```

The [ ] enclosed the name of the object **txtLName** in the above code. Access uses [ ] to enclose an object name with a space in it. Otherwise, as the above code shows, there is no difference whether or not the [ ] is enclosed.

The above code for InputBox has been simplified to use only Prompt and Title. The **SetFocus** method leaves the cursor in the **SS#** box and lets the user enter the **SS#** directly into it, rather than into the **InputBox**. It is another option for retrieving data from a user.

You learned the property, **Visible**, in the previous example. By integrating **IF** structures, you can avoid displaying Last Name, First Name, and SS# boxes if there is no input retrieved from **InputBox**. The same concept can be applied to the next event source code, **chkFName_Click( )**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The source code of **chkFName**'s **On Click** event is as follows:

```
Private Sub chkFName_Click()
    If [chkFName] = True Then
        [txtFName] = InputBox("Please enter your First Name here.", _
        "First Name")
        If txtFName.Value = "" Then
            txtFName.Visible = False
            txtSS.Visible = False
        Else
            txtFName.Visible = True
        End If
    Else
        txtFName.Visible = False
        txtSS.Visible = False
    End If
        If txtLName.Visible Then
            If txtFName.Visible Then
            txtSS.Visible = True
            [txtSS] = ""
            txtSS.SetFocus
            End If
        End If
End Sub
```

2. The second way to enter the First Name, Last Name, and SS# is to use the **Option Group** control. From the toolbox, click the **Option Group**, as shown in Fig. 4-9, and click the **Detail** section to drop the **Option Group** control and work through the Wizard (see Fig. 4-9a-f). You need to make sure the **Control Wizard** is activated before you click the **Option Group**.

Fig. 4-9

Enter the label names of each option you want to create. In this case, they are **Last Name** and **First Name**, as shown in Fig. 4-9a. Click NEXT.

Fig. 4-9a

Fig. 4-9b shows the default choice if you click **Yes.** You can select either Last Name or First Name from the drop-down list. Click NEXT.

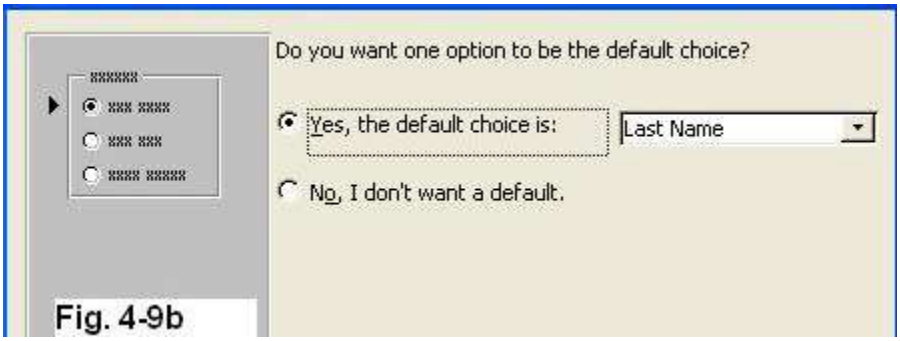


Fig. 4-9b

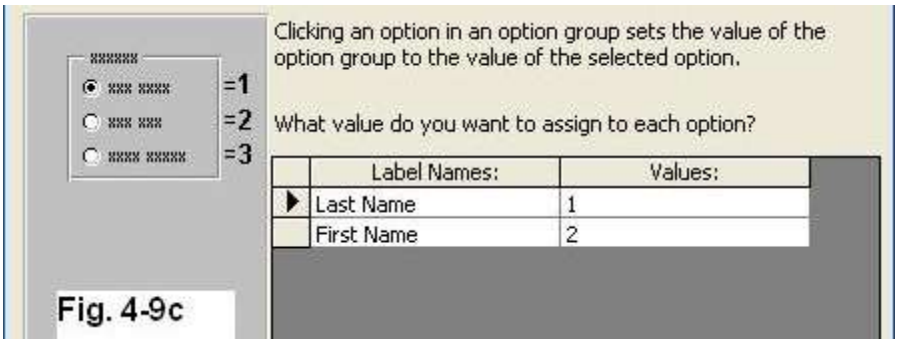


Fig. 4-9c

Accept the default values shown in Fig. 4-9c. You will need to use these values in the event procedure source code. Click NEXT.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

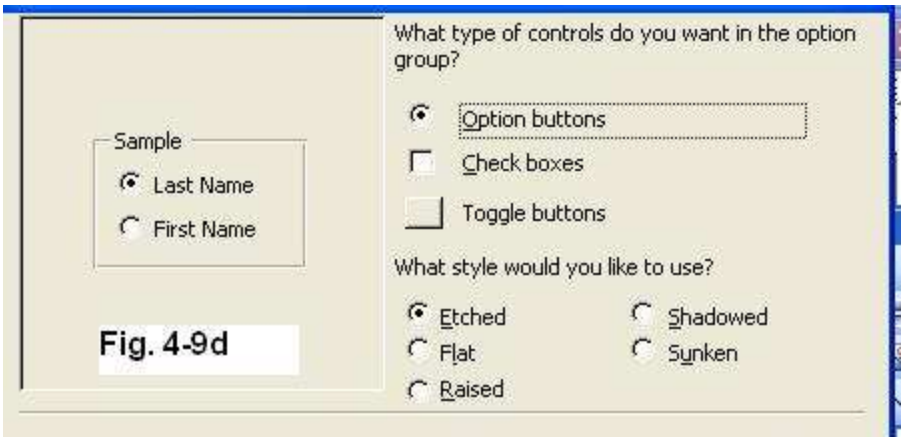
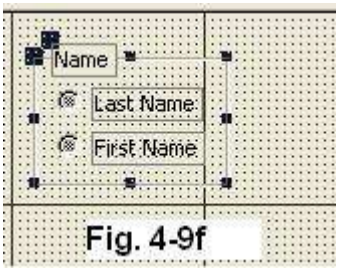


Fig. 4-9d shows the type of controls you can select. **Option buttons** allow exactly one selection while **checkboxes** allow none or multiple selections. **Toggle buttons** are either **ON** or **OFF**. In this example, choose the **Option buttons**. You will see a preview in the **Sample** section as you change the style selection. Click NEXT.

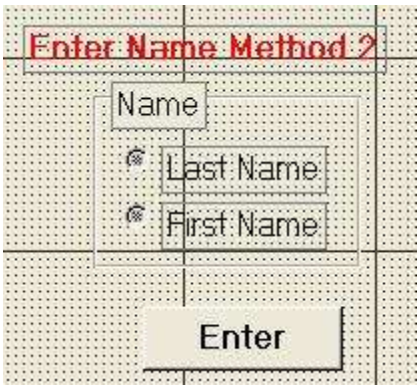
In Fig. 4-9e, the caption will appear in the top left-hand corner of the frame. Click FINISH. The final work is shown in Fig. 4-9f. The font size and color of both labels can be formatted in their **Properties**.



2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

After changing the format properties, the complete **Option Group** control is:



The properties of each object are:

Control Types	**Properties**	**Values**
Label	Caption	Enter Name Method 2
	Fore Color	255
Command Button	Name	cmdEnter
	Caption	Enter
	Event	On Click
Option Group	Name	fraName
	Caption	Name

The next task is to code the event **On Click** for the **Enter** command button. The code uses the **Select Case** structure. If the user chooses the **Last Name** option inside the **Option Control**, the value of **fraName** is 1. Retrieving the **Last Name** input in the code of **Case 1** is the same as Enter Name Method One's. The same thing applies to **Case 2** if the **First Name** option is selected. The following is the code for **cmdEnter On Click** event.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

```
Private Sub cmdEnter_Click()
Select Case fraName
    Case 1
    txtLName.Value = InputBox("Please enter your Last Name here.", _
    "Last Name")
    If txtLName.Value = "" Then
        txtLName.Visible = False
        txtSS.Visible = False
    Else
        txtLName.Visible = True
    End If
    If txtLName.Visible Then
        If txtFName.Visible Then
        txtSS.Visible = True
        [txtSS] = ""
        txtSS.SetFocus
        End If
    End If
    Case 2
    txtFName.Value = InputBox("Please enter your First Name here.", _
    "First Name")
        If txtFName.Value = "" Then
            txtFName.Visible = False
            txtSS.Visible = False
        Else
            txtFName.Visible = True
        End If
    If txtLName.Visible Then
        If txtFName.Visible Then
        txtSS.Visible = True
        [txtSS] = ""
        txtSS.SetFocus
        End If
    End If
End Select
End Sub
```

The above are two different design options to retrieve input from users. The next example will include the combo box control (Fig. 4-10).

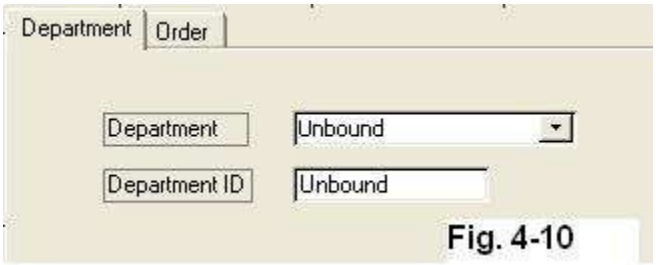
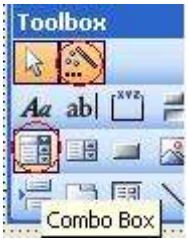


Fig. 4-10

Add a tab control to the form from the **Toolbox**. Name this tab control **TabCtlData**. On the first tab, add two kinds of control objects: the textbox and combo box. Select the combo box from the **Toolbox** to go through the Wizard as displayed in the following figure. The **Control Wizard** needs to be activated before you can use it.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6



The following is a step-by-step walk-through of the Wizard. In Fig. 4-11, you will enter the value list. Click NEXT.

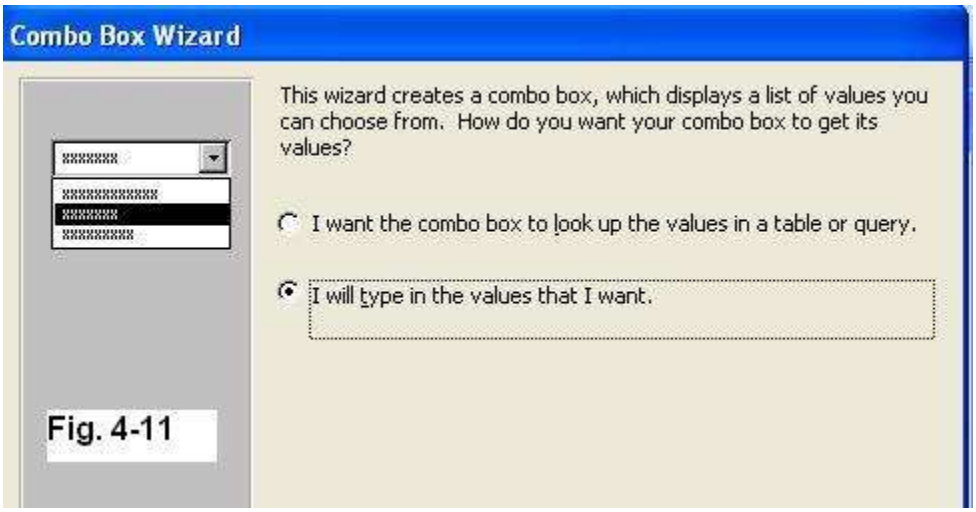


Fig. 4-11

In Fig. 4-11a, you will create one column entry by taking the default value 1. Complete the entry, as shown in the figure, for all departments. Click NEXT.

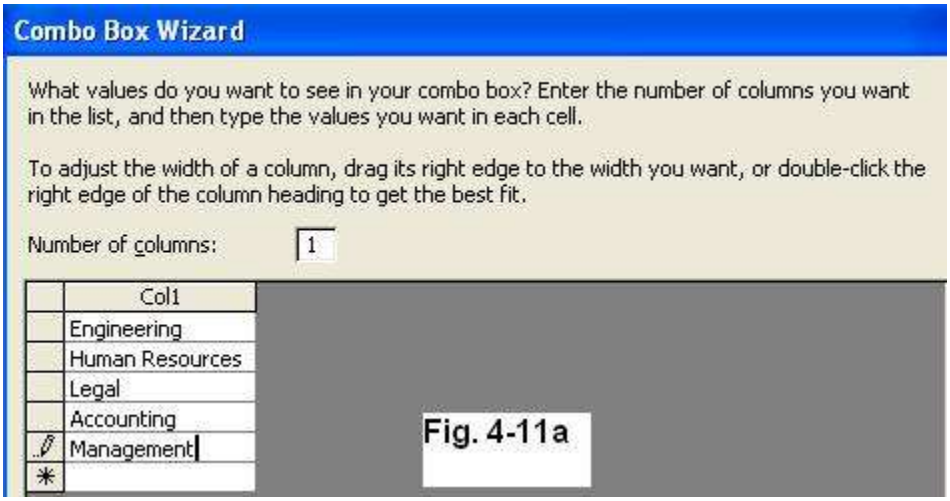


Fig. 4-11a

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Enter the caption of the label shown in Fig. 4-11b.

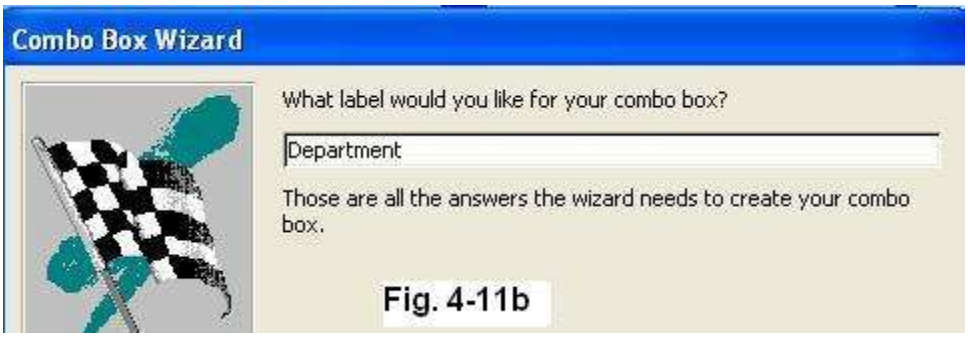


Fig. 4-11b

Control Types	Properties	Values
Tab page	Caption	Department
	Name	pgeDept
Combo box	Name	cboDept
	Event	AfterUpdate
Text box	Name	txtDeptID
Label	Caption	Department ID

```
Private Sub cboDept_AfterUpdate()
Select Case cboDept
    Case "Engineering"
        [txtDeptID] = "E"
    Case "Human Resources"
        txtDeptID.Value = "H"
    Case "Legal"
        txtDeptID.Value = "L"
    Case "Accounting"
        txtDeptID.Value = "A"
    Case "Management"
        txtDeptID.Value = "M"
End Select
End Sub
```

The above code passes the department ID to a textbox, **txtDeptID**, based on the selection of the variable **cboDept** in the **Select Case** structure.

The next dish will discuss the Function procedure and Sub procedure. It extends the application of VBA code.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

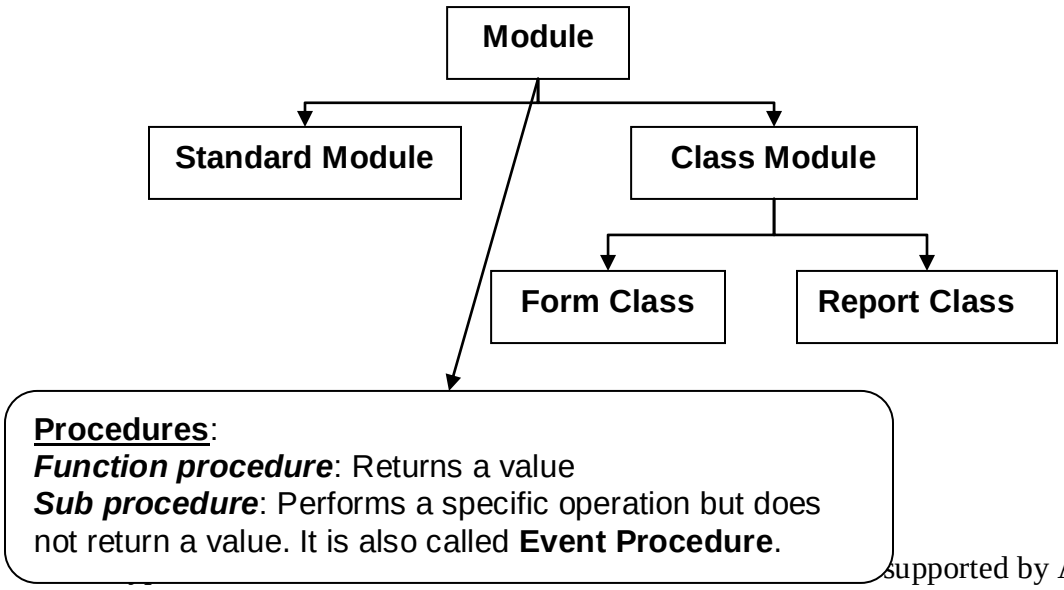
ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Chapter 5 Function and Sub Procedures

table of contents

Flavor 5.1 Basic structure of module

The basic element of the VBA language is the **module**. The figure shown below briefly explains the correlation between components in a module. The focus here will be on **Sub Procedures** and **Function Procedures**.



supported by Access VBA. Class modules are directly related to an object, such as a form or report. Form class modules contain event procedures for the associated controls, like a combo box, or objects, such as a textbox. Standard modules, on the other hand, have no association with an object. They store a collection of variables and user-defined procedures, which can be shared among Access programs. You can add a standard module either from the module objects in the Microsoft Access window, or through the **Visual Basic Editor** (VBE).

Understanding modules and procedures

In Chapter 4, you worked primarily with **Code Builder** to create event procedures on existing forms. Because forms are essential objects in Microsoft Access applications, this type of procedure is probably where you'll write most of the code. However, another important type of procedure you'll create here is called a ***general*** procedure. General procedures don't run automatically in response to events. Instead, you have to execute them yourself, either by referring to them in an expression on a form or report, or by using them in VBA codes. Typically, general procedures come in two types: Sub procedures and Function procedures. Event procedures are all Sub procedures.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The following, in bold, are a few reasons on why you should use **General Procedures**:

An expression cannot perform complex operations. For simple and quick operations, an expression can efficiently meet needs. Yet, for complex operations, it becomes a cumbersome task to complete an expression and sometimes, nearly impossible to create one. This feature will create procedures to perform needed operations and will refer to them in your expressions. Additionally, **Visual Basic Code** can perform complex operations more efficiently than using expressions.

You can reuse a program code to repeat a task. The event procedures you create often perform the same or similar actions at different times. For example, you might often format the textbox content with different font, size, or color. You may copy the same code into every event procedure that uses it; however, this is inefficient. Instead, you can reuse the program code to perform the common tasks by creating the general procedure.

It is easy to manage the VBA code, because it breaks up long tasks into procedural units. By dividing a long task procedure into individual manageable procedural units, the work is simplified and the codes are easy to read and understand.

You can choose private over public, avoiding conflicts. In a form or report class module, declare procedures using the **Private** keyword unless you have a reason to call them from outside the form. In a standard module, most procedures will need to be declared using **Public**, because you'll want to use them throughout your database. If they're only used to perform subtasks for other procedures in the module, be sure to declare them using **Private**. Making procedures P**rivate** whenever possible uses less memory, and can help avoid conflicts.

Flavor 5.2 Introductions to Sub Procedures

table of contents

Sub procedures must begin with a **Sub** statement and stop with an **End Sub** statement. They can contain executable Visual Basic statements, such as declaration and assignment statements. Sub procedures can take arguments such as variables, constants, and/or expressions. If no arguments are provided, the beginning **Sub** statement must contain an empty set of parentheses. Its syntax is:

[**Private** | **Public** | **Friend**] [**Static**] **Sub** *name* [**(***arglist***)**]
[*statements*]
[**Exit Sub**]
[*statements*]
End Sub

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

The **Sub** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Sub** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private** statement, the procedure is not available outside the project.
Private	Optional. Indicates that the **Sub** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Sub** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Sub** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Sub**, even if they are used in the procedure.
name	Required. Name of the **Sub**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.
statements	Optional. Any group of statements to be executed within the **Sub** procedure

Example:

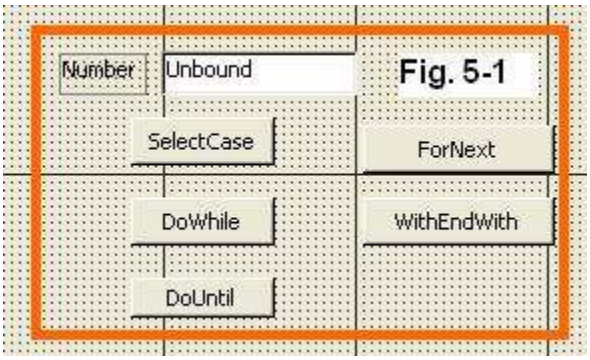
```
' Sub procedure definition.
' Sub procedure with two arguments.
Sub SubComputeArea(Length, TheWidth)
    Dim Area As Double    ' Declare local variable.
    If Length = 0 Or TheWidth = 0 Then
    ' If either argument = 0.
        Exit Sub    ' Exit Sub immediately.
    End If
    Area = Length * TheWidth    ' Calculate area of rectangle.
    Debug.Print Area    ' Print Area to Debug window.
End Sub
```

In the above code, the apostrophe (’) lets you add comments. VB will ignore all statements to the right of apostrophe, where you can add notes for future reference. The **Debug.Print** statement is printed in the Immediate window. This window is a tool that helps you troubleshoot your application. You can press Ctrl+G to activate the Immediate window. The next few examples show you how the Sub Procedure works under different structures, which were introduced in Dish 4: **Select Case**, **Do Loop**, and **For Next**.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

As shown in Fig. 5-1, add a textbox named **txtNum** with the label **Number**. Four command buttons with the following properties need to be added to the **VBACodes** form.



Controls	Properties	Values
Command button	Name	cmdCaseSelect
	Caption	SelectCase
	Event	On Click
Command button	Name	cmdDoWhile
	Caption	DoWhile
	Event	On Click
Command button	Name	cmdDoUntil
	Caption	DoUntil
	Event	On Click
Command button	Name	cmdForNext
	Caption	ForNext
	Event	On Click
Command button	Name	cmdWithEndWith
	Caption	WithEndWith
	Event	On Click

The corresponding Sub Procedure of the above command buttons' **On Click** event is:

```
Private Sub cmdWithEndWith_Click()
With Me.txtNum
    .BackColor = 16777082
    .ForeColor = 16777680
    .Value = "25"
    .FontSize = "12"
    .FontName = "Arial"
End With
End Sub
```

In the above code, a keyword, **Me**, is used. **Me** is an implicitly declared variable. You can use **Me** to write generic codes in a form or a report without identifying the activated object name. In other words, **Me** refers to the activated object, which could be a form, report, textbox, or combo box. For the above code, the activated object is the form **VBACodes**. One of the

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

controls on this form is called **txtNum**. The code **Me.txtNum** is the same as **[Forms]![VBACodes].txtNum**. By using this structure, the **txtNum** will be formatted by those properties.

Next is the **Select Case** structure:

```
cmdCaseSelect                                   Click

Private Sub cmdCaseSelect_Click()
Dim intage As Integer
    intage = Nz(Me.txtNum.Value, 0)
Select Case intage
    Case 0
        MsgBox "You must enter a number."
    Case 1 To 18
        MsgBox "You are just a kid."
    Case 19, 20, 21
        MsgBox "You are almost an adult."
    Case 22 To 40
        MsgBox "Good deal."
    Case Is > 40
        MsgBox "Getting Up There!"
    Case Else
        MsgBox "You entered an invalid number."
End Select
End Sub
```

The **Nz** function takes care of **Null** values, which may, sometimes, cause problems in computations. You can use the **Nz** function to return zero, a zero-length string (" "), or another specified value when a **Variant** is **Null**. For example, this function can convert a **Null** value to another, preventing it from occurring in other expressions. The syntax is as follows:
Nz(***variant***, [***valueifnull***])
The **Nz** function has the following arguments:

Argument	**Description**
variant	A variable of data type **Variant**.
valueifnull	Optional (unless used in a query). A **Variant** that supplies a value to be returned if the *variant* argument is **Null**. This argument enables you to return a value other than zero or a zero-length string. If you use the **Nz** function in an expression in a query without using the *valueifnull* argument, the results will be a zero-length string in the fields that contain null values.

In the above code, **Nz(Me.txtNum.Value, 0)** it converts the value of **txtNum** to zero if **txtNum** contains **Null**. Otherwise, it keeps any number entered by the user in the **txtNum**. It will display the message box **MsgBox** depending on the value of **intage**, which is entered by user in the **txtNum**.

The next three structures are examples of iteration. The first two loops, **Do Until** and **Do While**, are pretest loop structures, as shown in the code below. However, if you would like to have a posttest loop structure, in order to execute the statement at least once, you need to remove the apostrophe from the code in green and put the apostrophe on the other loop structure.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

The third structure, **For Next** will run the statement ten times. However, if you remove the apostrophe from

For intCounter = 1 To 10 Step 2

and add it to

For intCounter = 1 To 10

the statement will be executed only five times instead of ten.

```
Private Sub cmdDoUntil_Click()
Do Until Nz(Me.txtNum.Value) = 40
    Me.txtNum.Value = Nz(Me.txtNum.Value) + 1
Loop
'Do
'    Me.txtNum.Value = Nz(Me.txtNum.Value) + 1
'Loop Until Nz(me.txtnum.value)=40
End Sub
```

```
Private Sub cmdDoWhile_Click()
Do While Nz(Me.txtNum.Value) < 40
    Me.txtNum.Value = Nz(Me.txtNum.Value) + 1
Loop
'Do
'    me.txtnum=nz(me.txtnum.value)+1
'Loop while Nz(me.txtnum.value)<40
End Sub
```

```
Private Sub cmdForNext_Click()
Dim intCounter As Integer
    For intCounter = 1 To 10
'For intCounter = 1 to 10 step 2
        Me.txtNum.Value = Nz(Me.txtNum.Value) + 1
    Next intCounter
End Sub
```

The next example uses the **If..Then.. Else** structure.

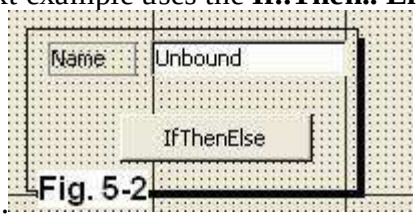


Fig. 5-2

A command button, **cmdIfThenElse**, will be added to the **VBACodes** form. A textbox, **txtName**, will be put into the form as well. The code below will display messages according to the content of **txtName**.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

```
cmdIfThenElse                                   Click
Private Sub cmdIfThenElse_Click()
If IsNull(Me.txtName.Value) Then
    MsgBox "Name is Blank."
Else
    MsgBox "Your Name is " & Me.txtName.Value
End If
End Sub
```

The following **Sub Procedure** example uses the **Static** keyword.

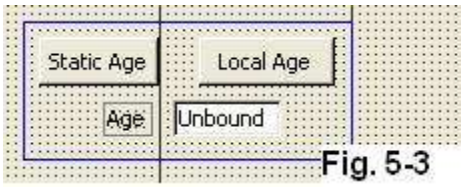


Fig. 5-3

Two command buttons, shown in Fig. 5-3, are added to the form. Their names are **cmdlocalage** and **cmdStaticAge**, respectively. A textbox, **txtnewage**, is also added to the form.

```
cmdlocalage                                     Click
Private Sub cmdlocalage_Click()
Dim intage As Integer
intage = intage + 1
Me.txtnewage.Value = intage
End Sub
```

Variables declared within a procedure normally lose their values when the procedure ends. Each time you click the button, **cmdlocalage**, the above code will reset the variable **intage** back to 0. When you click **cmdStaticAge**, the code below will not reset the value of **sintage** to 0. It uses the **Static** keyword to replace the **Dim** keyword to keep the value from the last execution. This is useful for a procedure that accumulates a value each time it is called.

```
Private Sub cmdStaticAge_Click()
Static sintAge As Integer
sintAge = sintAge + 1
Me.txtnewage.Value = sintAge
End Sub
```

How to call Sub procedures

After you have created a Sub procedure in a standard module, you can access it from anywhere in the application. You can execute the Sub procedure by calling it from another procedure, attached to a form or any of the form's controls. Since one procedure cannot contain another procedure in it, you must call procedures to use them. You can call a Sub procedure from another procedure by typing the name of the Sub procedure.

You can also call a Sub procedure by using the **Call** keyword. The syntax is:

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Call *procedurename*

The procedure that calls another procedure is known as the *calling* procedure and the procedure that is being called is known as the *called* procedure.

You need to declare the public variable, **strAnimal**, in the area shown in the code below. This way, you don't need to declare the variable in the Sub procedure because it is a public variable and can be used by any Sub procedure.

```
Option Compare Database
Dim strAnimal As String 'if not to declare here, code will not work
Option Explicit
```

Add a command button, **cmdAnimal**, and a textbox, **txtAnimal**. You need two more command buttons, **cmdGoodBye** and **cmdHello**, which will be used in **Flavor 5.3 Function**.

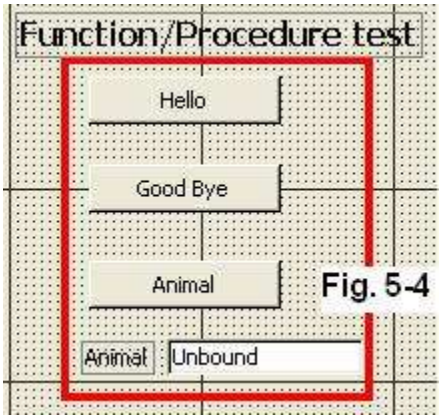


Fig. 5-4

With the command button **cmdAnimal**, you will enter following code.

```
Private Sub cmdAnimal_Click()
strAnimal = "Birds"
txtAnimal = strAnimal
MsgBox "Animal change"
Call ChangeParty
txtAnimal = strAnimal
End Sub

Private Sub ChangeParty()
    strAnimal = "Bulls"
End Sub
```

When you click the button, the first animal, Birds, will appear in the textbox. Next, a message box appears, showing you the message "Animal Change". Click the OK button in the message box. Then the content of the textbox will change to "Bulls". This is an example of how one Sub procedure calls another Sub procedure to perform using the **Call** statement.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Flavor 5.3 Introductions to Function procedures

table of contents

Functions are a group of statements that perform specific tasks and return a value. What separates a Function procedure from a Sub procedure? A Function procedure returns a value, while a Sub procedure does not. Functions can be used to perform operations on variables, display information, or obtain information from a user. A Function procedure can be created in a standard module.

You can also create Function or Sub procedures that use variables as arguments. An ***argument*** is a variable, constant, or expression that adds information to a procedure. These arguments, also called parameters, are passed to the Function procedure when it is called.

You will write the following codes into either a Sub or a Function procedure.

```
Private Sub cmdGoodBye_Click()
Call SayGoodBye
End Sub
```

```
Private Sub cmdHello_Click()
MsgBox "Hello! By clicking Command Buton."
End Sub
```

```
Private Function SayGoodBye()
MsgBox "Good bye! Using the function."
End Function
```

When you click the button **cmdHello**, which is running the Sub procedure, the message box appears, alerting you that it is run by a Sub procedure, not a Function.

On the other hand, when you click the button **cmdGoodBye** to call Function **SayGoodBye**, a message box displays the message “Good bye! Using the function”.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipes. 2nd edition.

Flavor 5.4 Many ways to declare variables

table of contents

Variables can be public or private and can be in various places in your database, just like procedures. How variables work depends on where and how you declare them. So far, most of the variables you used in examples have been private, declared using the **Dim** keyword. When the procedure finishes running, variables declared using **Dim** disappear.
Here are different levels where you can declare variables:

Module-level

Use the **Dim** statement in the **Declarations** section of the module, outside of procedures, for declaring variables that you want to use throughout a module. The variable will keep its value between procedure calls. Use this type of variable when several procedures in the module all work with the same data.

Public

Public variables are useful for application-wide information, such as the names of the application users. To declare a variable that you'll use throughout an application, use the **Public** instead of the **Dim** statement. Place the **Public** statement in the **Declarations** section of any module. The variable will keep its value when your application is running. If you declare a public variable in a form or report module, the object's name needs to be included in all references to it that occur outside the object.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Flavor 5.5 Handling errors

table of contents

The program execution will return to the statement where the error occurred when you use the **Resume** keyword. There are three possible ways to return program control to the procedure:
Resume: returns program control to where the error occurred.
Resume Next: returns program control to the statement after the statement where the error occurred.
Resume Label: returns program control to a predetermined line number. If you know where an error is likely to occur and don't want to interrupt the running code, you can use the **On error GoTo Next** statement to suspend error handling. This statement will make Microsoft Access skip over the line that causes an error.

The technique you choose will depend on the types of errors you anticipate and how you want to handle them. If you don't know how you want to deal with an error, the best choice is to display the description of the error in a message box, and then exit the procedure.

What About the Code the Wizard Converted from a Macro?

The code converted by the Wizard from a macro will be stored in the module, as shown in Fig. 5-5.

Fig. 5-5

The error handling will automatically be created as shown in Fig. 5-6. Here's the complete sample procedure:

```
' McrDemo1
'
'-----------------------------------------------------------------
Function McrDemo1()
On Error GoTo McrDemo1_Err

    ' Open form Customers in Edit mode
    DoCmd.OpenForm "Customers", acNormal, "", "", acEdit, acNormal
    ' Go to record number 2 on form Customers
    DoCmd.GoToRecord acForm, "Customers", acGoTo, 2
    ' maximize Customers form
    DoCmd.Maximize
    ' Tell user form customers will close
    Beep
    MsgBox "Will close the customer form.", vbOKOnly, "Close customer form"
    ' close customers form
    DoCmd.Close acForm, "Customers"

McrDemo1_Exit:
    Exit Function

McrDemo1_Err:
    MsgBox Error$
    Resume McrDemo1_Exit

End Function
```

Fig. 5-6

If you would like to use it as a Sub procedure rather than a Function, you can change the procedure from Function to Sub instead. The **Exit Function** will be changed to **Exit Sub**.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Chapter 6 Improve the Taste

table of contents

Flavor 6.1 Improve the Parameter Query

The parameter query is very easy to use when there is only one parameter needed from the user. However, for certain scenarios, including multiple parameters, invalid/incorrect data, data not found in the database, and inability to return to the previous parameter input, writing the VBA code to fix the problems can be a nightmare.

One way to create a parameter input form is by providing selections of all parameters for the user.
In one query, make a list of parameters your project needs in the order you want users to select from, and what type of form control you want users to input. In this example, they are **Category Name**, **Unit Price 1**, and **Unit Price 2**.
Create a **Blank Form** and switch to **Design View** with the following controls, each with input parameters for **qryImprovedParameter**:

Query field	Data Type	Control Type	Parameter Reference
Category Name	Text	List box	Forms!frmProduct!lstCateName
Unit Price	Currency	Text box	Forms!frmProduct!txtPrice1
Unit Price	Currency	Text box	Forms!frmProduct!txtPrice2

The **Category Name** control type can be either a **list box** or a **combo box**, depending on your design preference. If you would like to create this **list box** by hand, select Ribbon Design as shown in Fig. 6-1 and use the **Control Wizard**. **Table/Query** for the **RowSourceType** property and **Categories** for the **RowSource**. Keep the rest as the default options.

Fig. 6-1

In the first step of the Wizard, keep the default option, as shown in Fig. 6-2. Click **Next** to move to Fig. 6-3.

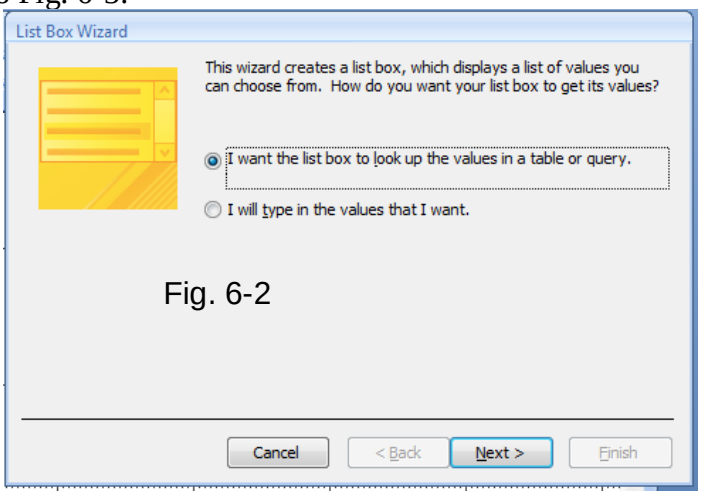


Fig. 6-2

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

As seen in Fig. 6-3, choose the **Table: Categories** as the data source. Click **NEXT** to move to Fig. 6-4.

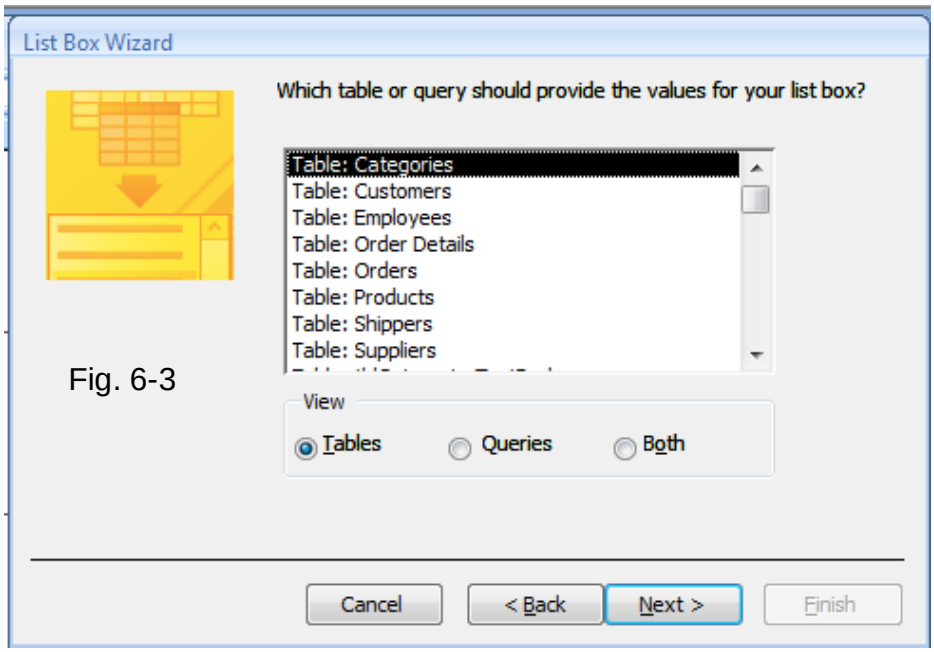


Fig. 6-3

In Fig. 6-4, select both fields to be displayed on the list box. Click **NEXT** to move to Fig. 6-5. In Fig. 6-5, select **CatergoryName** and leave **Ascending** order as is. Click **NEXT** to move to Fig. 6-7.

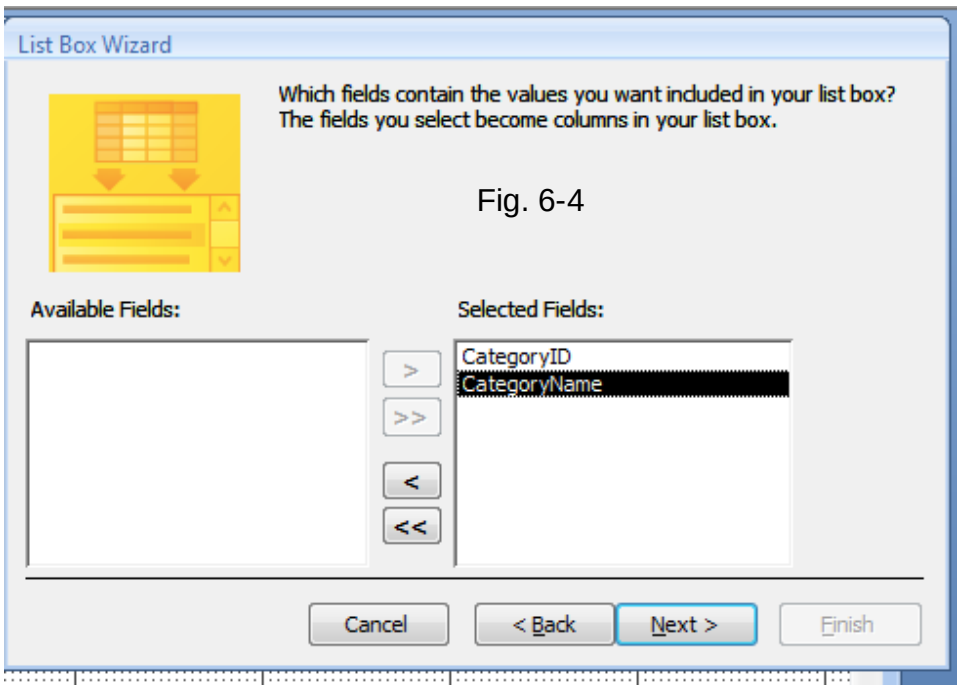


Fig. 6-4

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

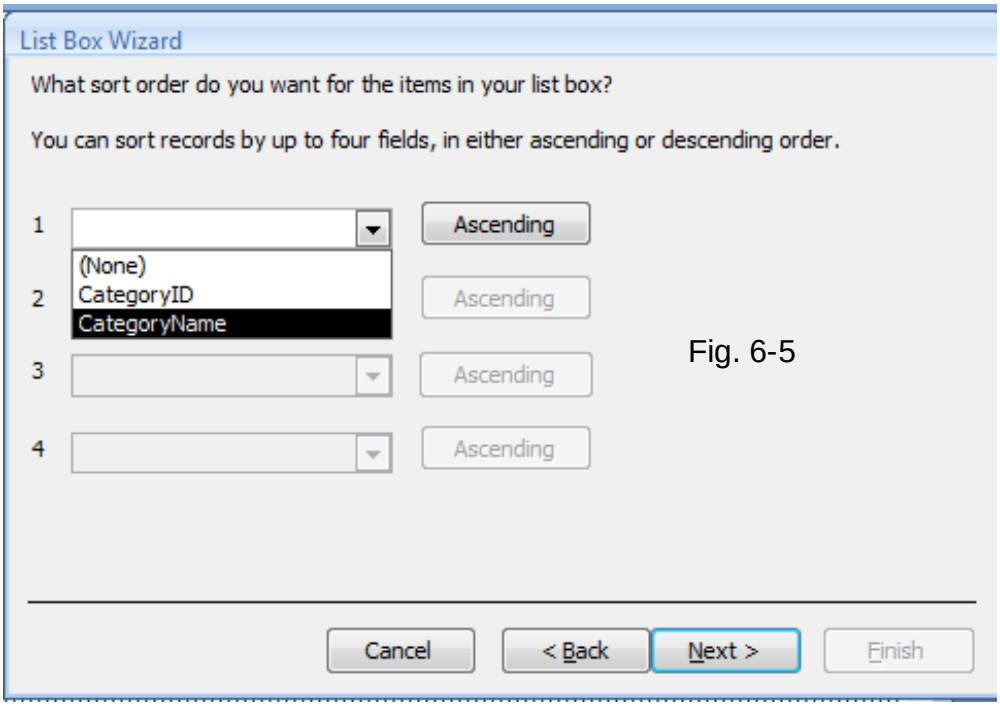


Fig. 6-5

Following Fig. 6-6, you will keep the box "Hide key column (recommended)" as is. By doing so, users will only see the **Category Name** column, not both columns in the list box. Another option of this control is to use the combo box instead. Click **NEXT** to continue to Fig. 6-7.

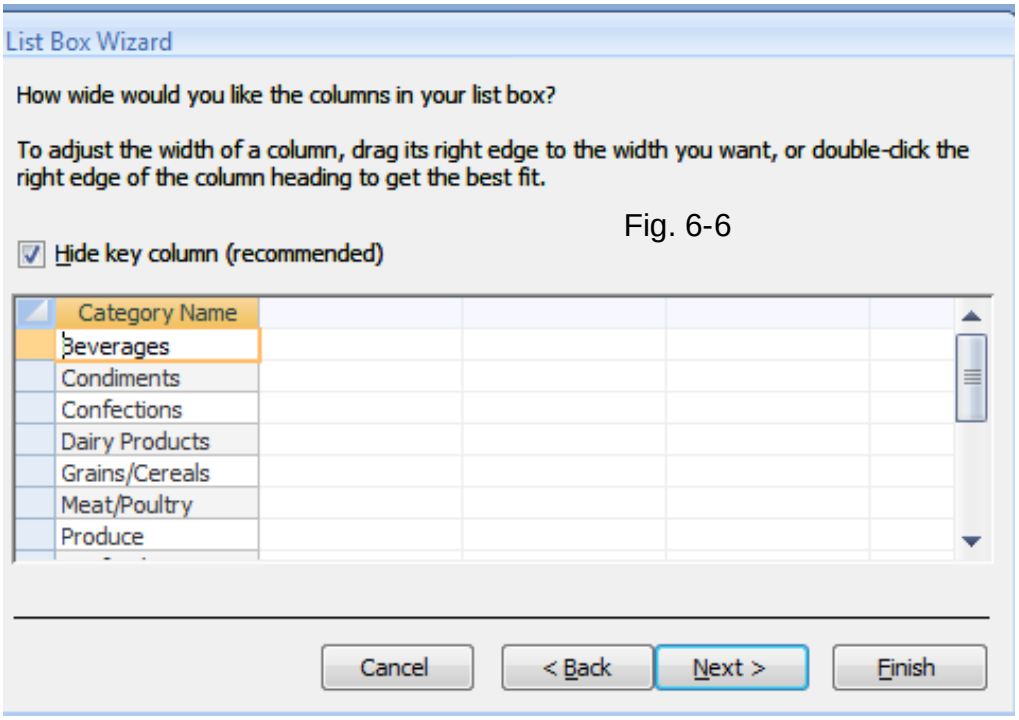


Fig. 6-6

In Fig. 6-7, you can rename the label if you would like, or you can keep it as is and rename it on the form after you finish this Wizard. Click FINISH. Fig. 6-8 is in **Design View**. You may want to switch to **Form View** to see what the form will look like and make necessary adjustments. For example, you may need to adjust either the width or the height of the list box in order to display certain names in the list box. The final size of the list box will depend on the entire form design, which is related to other objects on the form. You don't

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

want the list box to be either too small or too big compared to other controls on the form. Keep in mind you are not only a programmer, but also an artist. Your design is artwork.

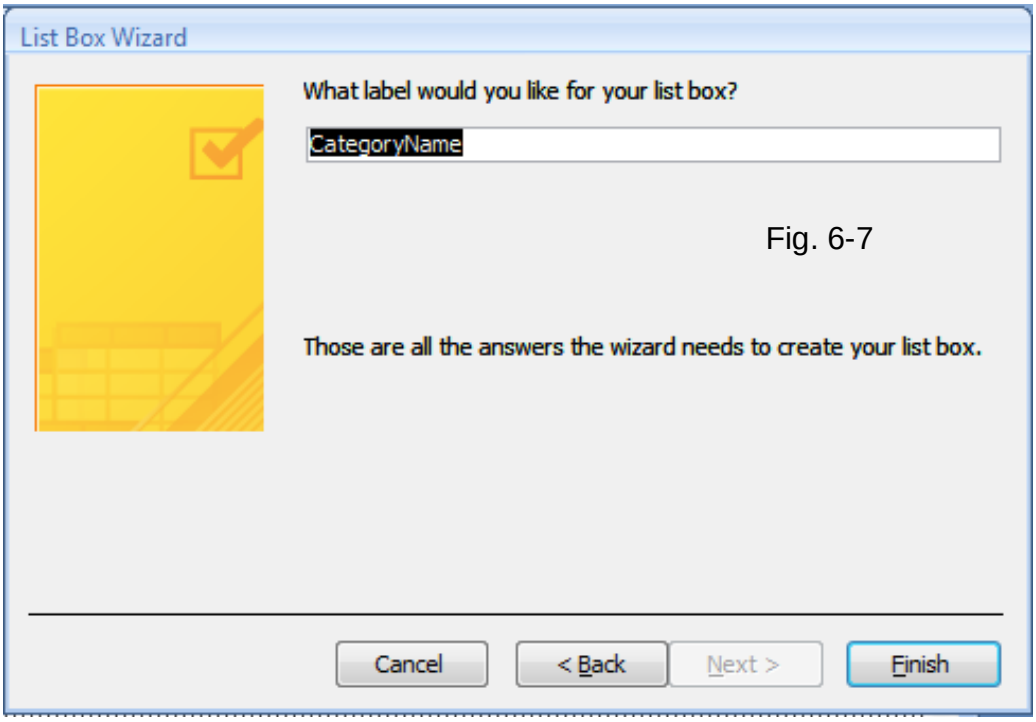


Fig. 6-7

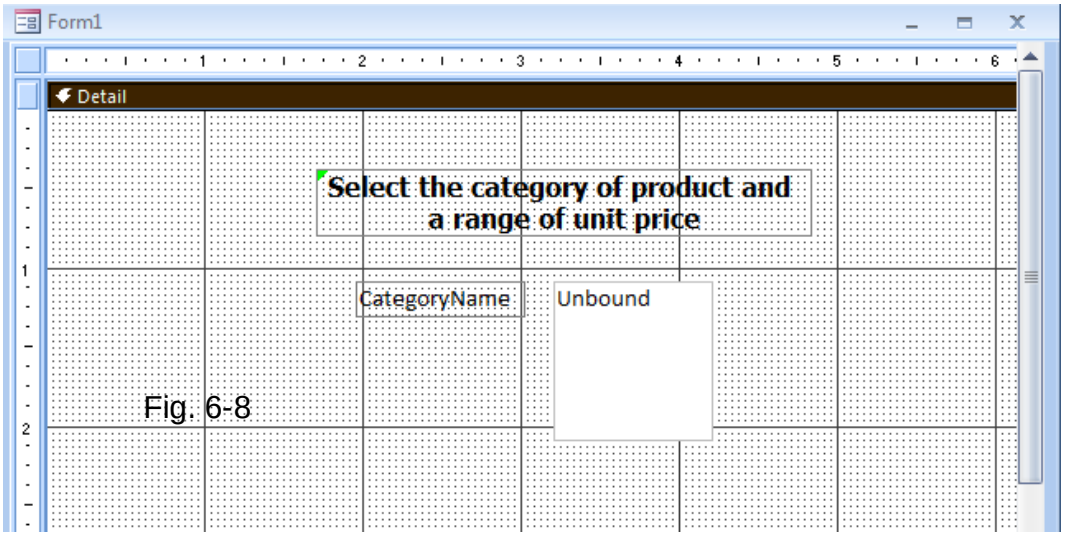


Fig. 6-8

Two textboxes are needed for taking the price range. You will also need to name these controls according to the list of controls at the beginning of this section. Figure 6-9 is showing the control property **Name** for the first price **txtPrice1** in the range. The second price text box should be named **txtPrice2** and the list box will be named **lstCateName**. The "lst" is the prefix for list box. Finally, the **Bound Column** value should be changed from 1 to 2. In doing so, the selection in the list box will use the **Category Name**, instead of the **Category ID**, to match the **Category** field in the **Query Criteria**. If you don't make this value change, you will not get any data, even if all three selections are correct.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

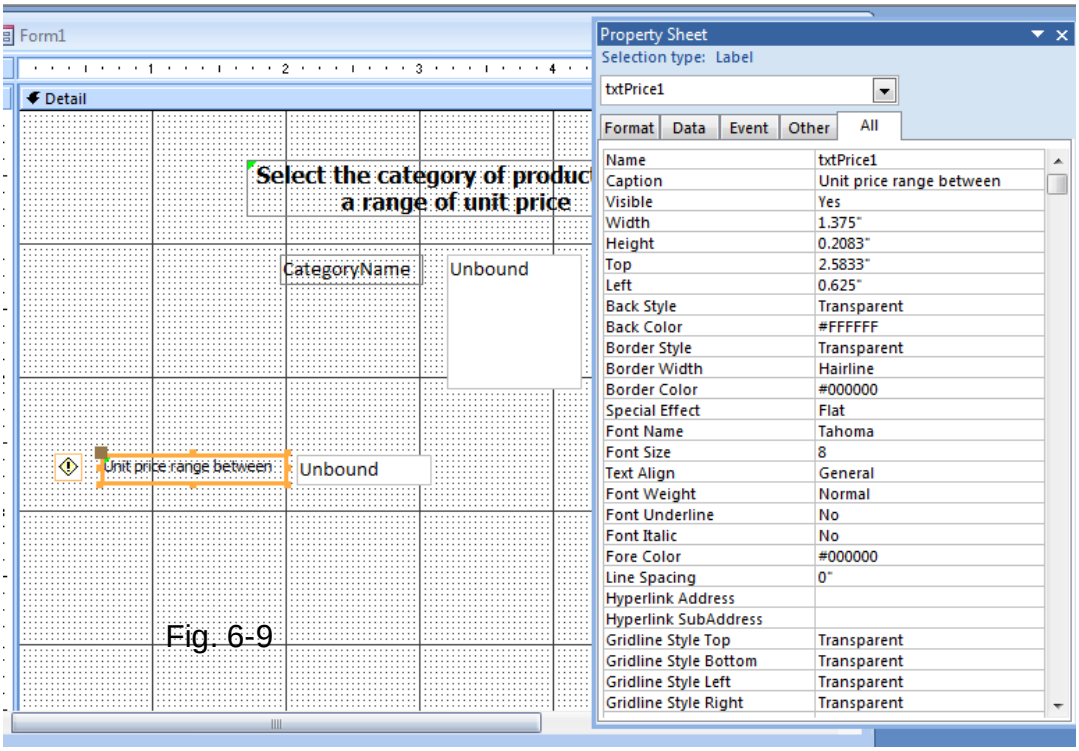


Fig. 6-9

On this form, make two command buttons, OK and Cancel; the codes, from using the command button Wizard covered in the previous chapter, behind these two commands are as follows:

```
Option Compare Database

Private Sub cmdCancel_Click()
On Error GoTo Err_cmdCancel_Click
    DoCmd.Close
Exit_cmdCancel_Click:
    Exit Sub
Err_cmdCancel_Click:
    MsgBox Err.Description
    Resume Exit_cmdCancel_Click

End Sub
Private Sub cmdOK_Click()
On Error GoTo Err_cmdOK_Click

    Dim stDocName As String

    stDocName = "qryImprovedParameter"
    DoCmd.OpenQuery stDocName, acNormal, acEdit

Exit_cmdOK_Click:
    Exit Sub

Err_cmdOK_Click:
    MsgBox Err.Description
    Resume Exit_cmdOK_Click
End Sub
```

Fig. 6-10

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

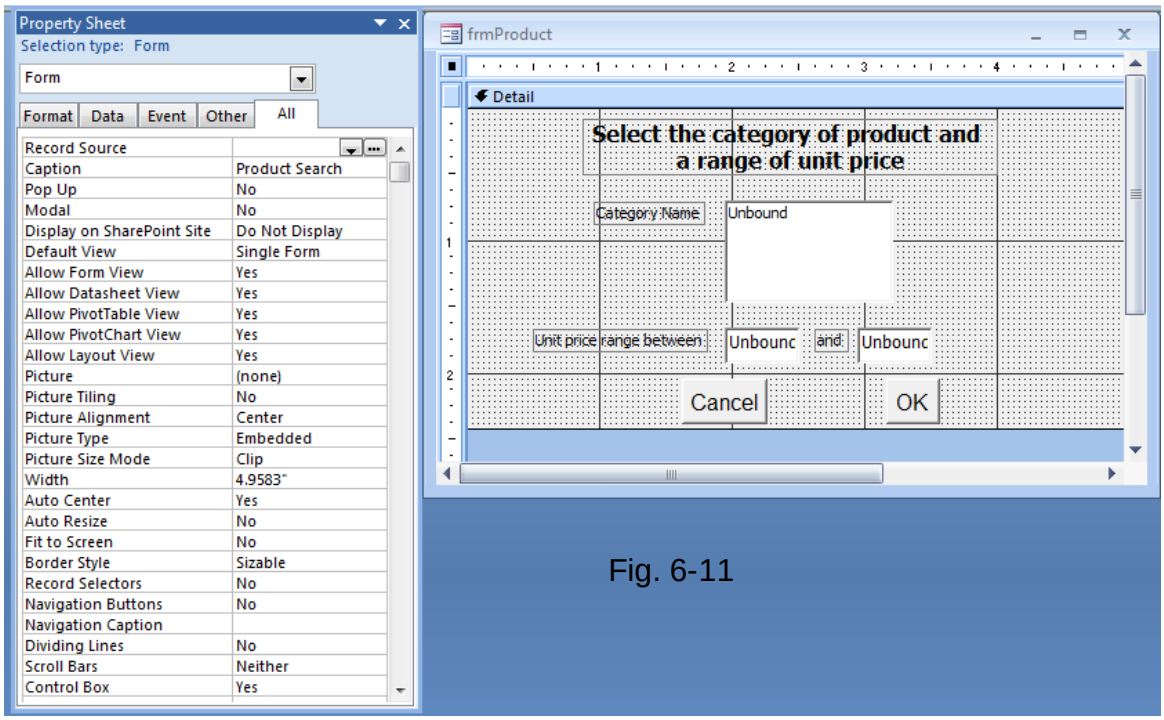


Fig. 6-11

Open the Form property to make following changes if you would like:

Caption: Product Search (type it)
Scroll Bars: Neither (select it from the list)
Record Selector: No (select it from the list)
Navigation Buttons: No (select it from the list)
Min Max Buttons: None (select it from the list)
Close Button: No (select it from the list)
The **Form View** is shown in Fig. 6-12. This is only a suggested design. The important part of this design will be the next item: the query.

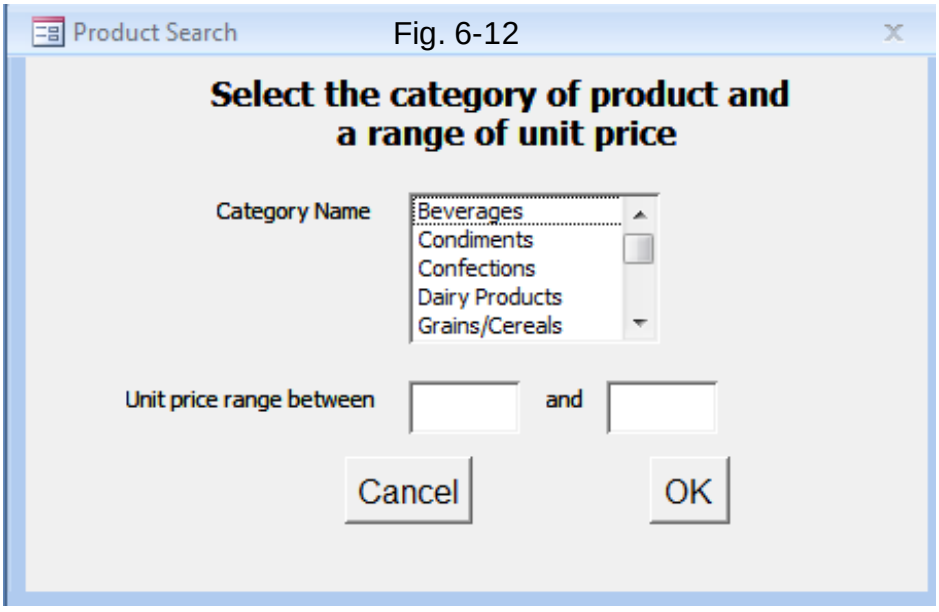


Fig. 6-12

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Create the query, named **qryImprovedParameter**. This is the bridge that will connect the front, input form, and the back, search query. It is different compared to the default parameter query. Here, instead of using the square brackets, you will enter the references to the form controls for each parameter. You don't need to worry about the cases because object names are not case-sensitive in Access. Type keywords, such as "between", in lowercase. As the cursor leaves the criteria cell, if you enter the data correctly, Access should automatically capitalize the first letter of each keyword. This can help catch typos because, for incorrect data, Access will not capitalize the keyword, signaling you to modify the data you entered.

In the **CategoryName** field on the query, enter:
forms!frmProduct!lstCateName
In the **UnitPrice** field, enter
between forms!frmProduct!txtPrice1 and forms!frmProduct!txtPrice2

Fig. 6-12a and 6-12b show the complete work on QBE.

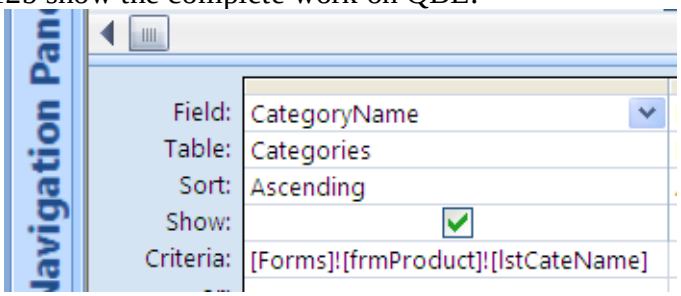
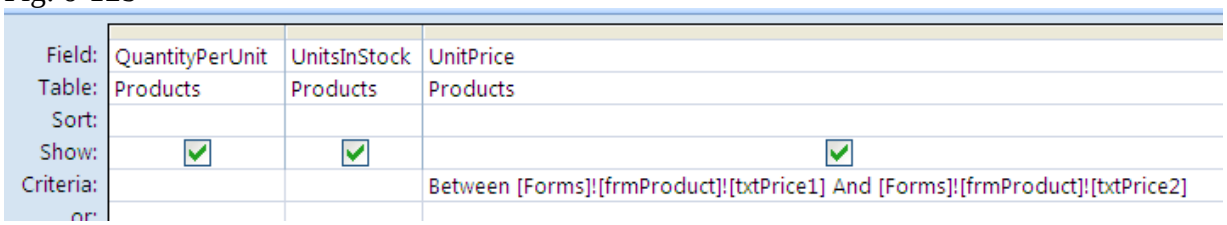


Fig. 6-12a

Fig. 6-12b



You will then declare the data type of the parameters by selecting, on the Ribbon **Design**, **Parameters** on **Show/Hide** group as shown on the Fig. 6-12b1 (note: if you see the **Parameters** in dim switching the query to **Design View** will make it available). Use the same name for each parameter and choose the proper data type, as described in Step 1. The parameter dialog window is shown in Fig. 6-12c. Make sure no extra brackets are added when these control parameters are entered (note: you may not need to type it at all if you complete Fig. 6-12b properly in the QBE).

Fig. 6-12b1

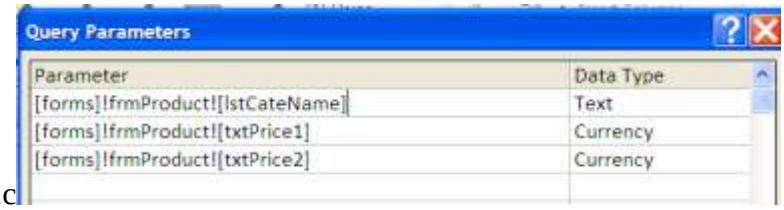
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

[Forms]!frmProduct![lstCateName]
[Forms]!frmProduct![txtPrice1]
[Forms]!frmProduct![txtPrice2]

They are not case-sensitive, so you don't need to follow the exact case as given as long as these characters are correct. Click OK to close the parameter dialog window.

Fig. 6-12c



One more place is needed to check before this design is ready and completed. In the QBE, switching to SQL view either from the View group (see Fig. 6-12c1) or right-click on mouse (see Fig. 6-12c2) so you can see SQL script (see Fig. 6-12d) to remove the outside bracket for each parameter (refer to Fig. 6-12e) if there are extra brackets. Keep the bracket for each object (i.e. frmProduct) or control (i.e. txtPrice1).

Fig. 6-12c1

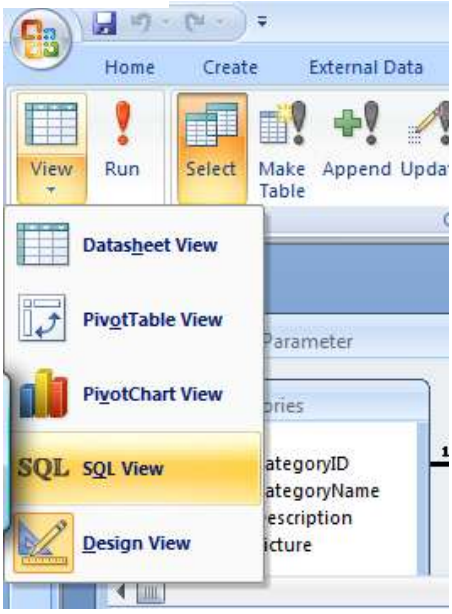


Fig. 6-12c2

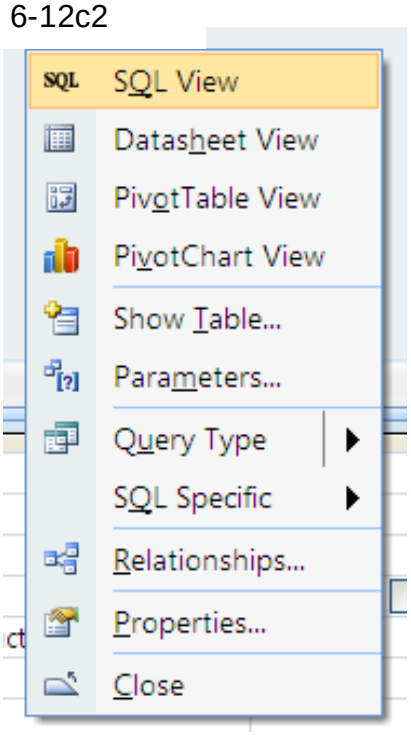


Fig. 6-12d

```
PARAMETERS [forms]!frmProduct![lstCateName] Text ( 255 ), [forms]!frmProduct![txtPrice1] Currency, [forms]!frmProduct![txtPrice2] Currency;
SELECT Categories.CategoryName, Products.ProductName, Products.QuantityPerUnit, Products.UnitsInStock, Products.UnitPrice
FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
WHERE (((Categories.CategoryName)=[Forms]![frmProduct]![lstCateName]) AND ((Products.UnitPrice) Between [Forms]![frmProduct]![txtPrice1] And
[Forms]![frmProduct]![txtPrice2]))
ORDER BY Categories.CategoryName, Products.ProductName;
```

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

```
[[Forms]!frmProduct![lstCateName]]
[[Forms]!frmProduct![txtPrice1]]
[[Forms]!frmProduct![txtPrice2]]
```

Fig. 6-12e

Save the query and close it.
Test it by opening the form, **frmProduct**, in **Form View** and select or enter the value of each parameter (i.e. category name is Beverages, the first price is $10, the second is $50) as shown in Fig. 6-12f. Click OK to execute the query once you have input all the data. Click Cancel to close the form.

This entry form provides extra time for users to double-check the information they enter. If any parameter is incorrect, they may fix it, as no query will be executed before the OK button is clicked. In particular, when there are a lot of required parameters that need to be entered in a design, this increases the accuracy of the parameter inputs.

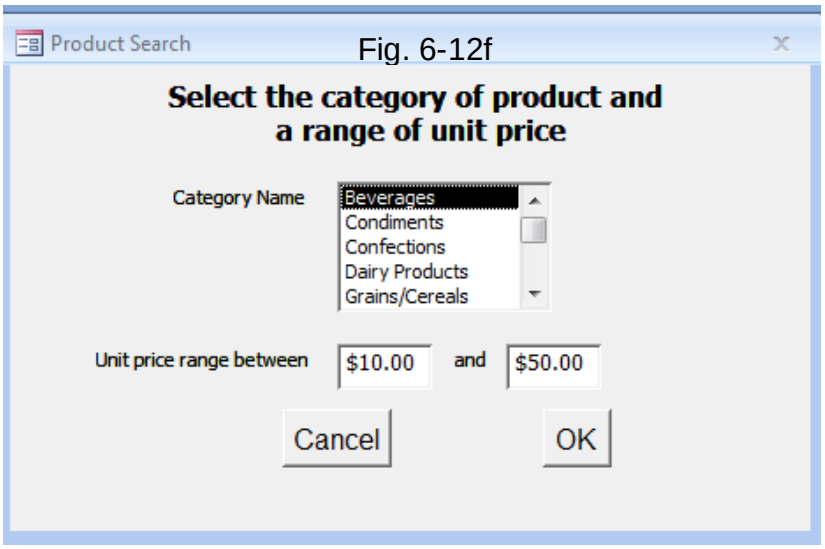


Fig. 6-12f

A query result is shown in Fig. 6-12g.

qryImprovedParameter

Category Name	Product Name	Quantity Per Unit	Units In Stock	Unit Price
Beverages	Chai	10 boxes x 20 bags	39	$18.00
Beverages	Chang	24 - 12 oz bottles	17	$19.00
Beverages	Chartreuse verte	750 cc per bottle	69	$18.00
Beverages	Ipoh Coffee	16 - 500 g tins	17	$46.00
Beverages	Lakkalikööri	500 ml	57	$18.00
Beverages	Laughing Lumberjack Lager	24 - 12 oz bottles	52	$14.00
Beverages	Outback Lager	24 - 355 ml bottles	15	$15.00
Beverages	Sasquatch Ale	24 - 12 oz bottles	111	$14.00
Beverages	Steeleye Stout	24 - 12 oz bottles	20	$18.00

Record: 1 of 9 No Filter Search

Fig. 6-12g

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.2 Finding matches as the value in a range of the second table table of contents

A typical example of finding a match for a value is finding a letter grade based on the score of each student. The first table, **tblstudents** as shown in Fig. 6-13, stores all the students' information including their names and scores. The second table, **tblgradescale**, as shown in Fig. 6-14, has the grade scale for the scores. You will need to display the names of each student and their letter grades in a query.

tblstudents : Table Fig.6-13

StudentID	Names	Scores
1	Mary	87
2	John	76
3	Terry	67
4	Minjan	56
5	Shen	74
6	Kent	91
7	Sue	54
8	Sara	62
9	Steve	79

tblgradescale : Table

Grade	LowEnd	HighEnd
A	90	100
B	80	89
C	70	79
D	60	69
F	0	59
	0	0

Fig. 6-14

Create a new query, **qryLetterGrade**, to include both tables. Since there is no relationship between these two tables, there is no Wizard that can create it. You will need to manually create this query.

In the QBE, drag the fields from both tables that you would like to display in the result. For example, names and scores from **tblstudents** and the grade field from **tblgradescale** are pulled into the QBE.

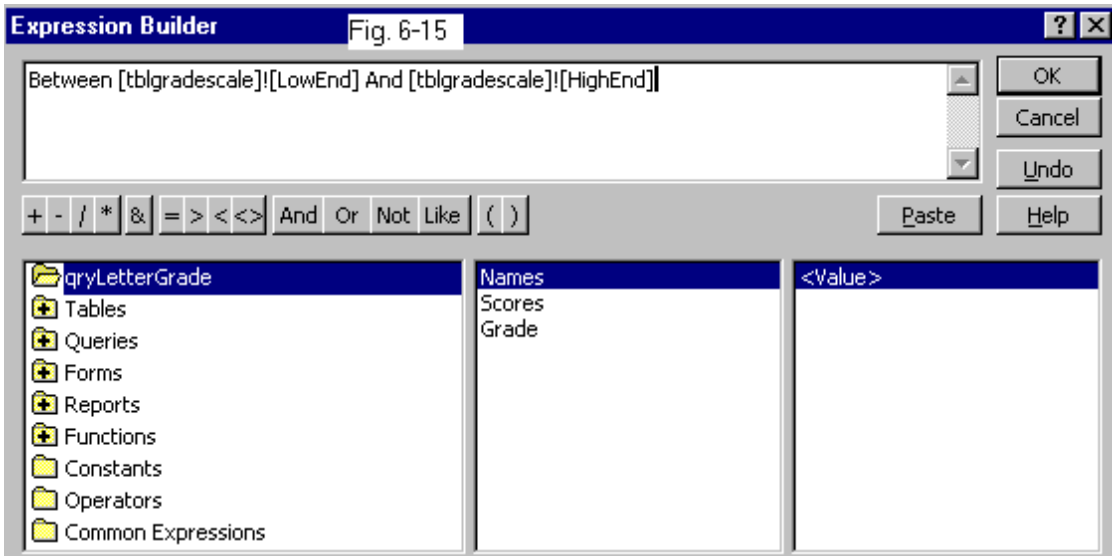
The key area is in the criteria of the scores field. If you don't want to type the expression by hand, enter the following expression by right-clicking the mouse and choosing **Build**, then **Expression Builder**, as shown in Fig. 6-15:

Between [tblgradescale].[LowEnd] And [tblgradescale].[HighEnd]

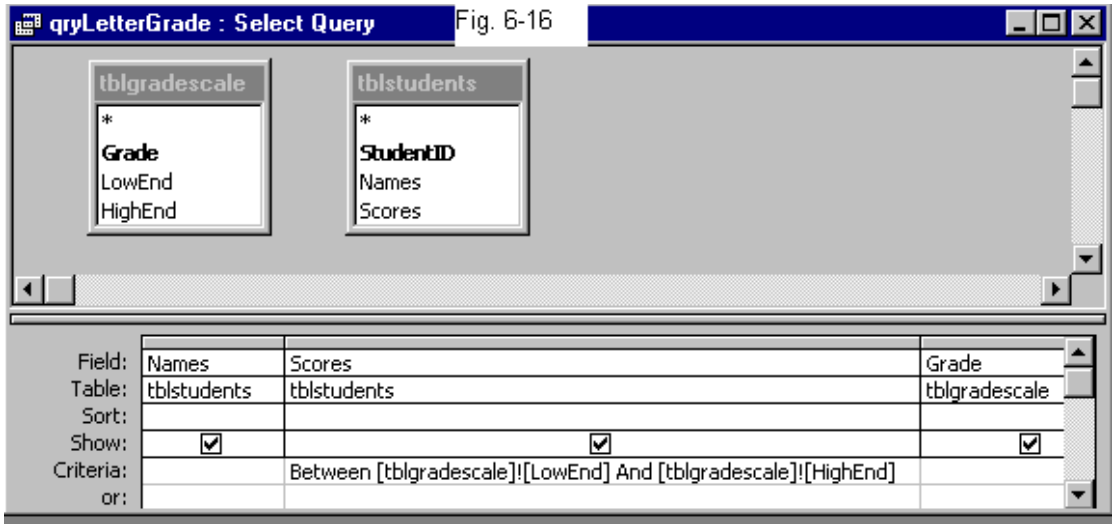
It is best to use the mouse to select the field you need rather than type it, because it decreases the amount of potential typos. Often, a lot of unnecessary bugs are caused by unfound or misspelled names. The field **LowEnd** and **HighEnd** can be found by clicking **Tables** on the left panel and then **tblgradescale**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



Click OK in the **Expression Builder** and go back to QBE, as shown in Fig. 6-16.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Run the query; the result is shown in Fig. 6-17.

qryLetterGrade : Select Query Fig. 6-17

Names	Scores	Grade
Mary	87	B
John	76	C
Terry	67	D
Minjan	56	F
Shen	74	C
Kent	91	A
Sue	54	F
Sara	62	D
Steve	79	C

Record: 1 of 9

Flavor 6.3 Speed up the item selection in one combo box based on the selection in another combo box

table of contents

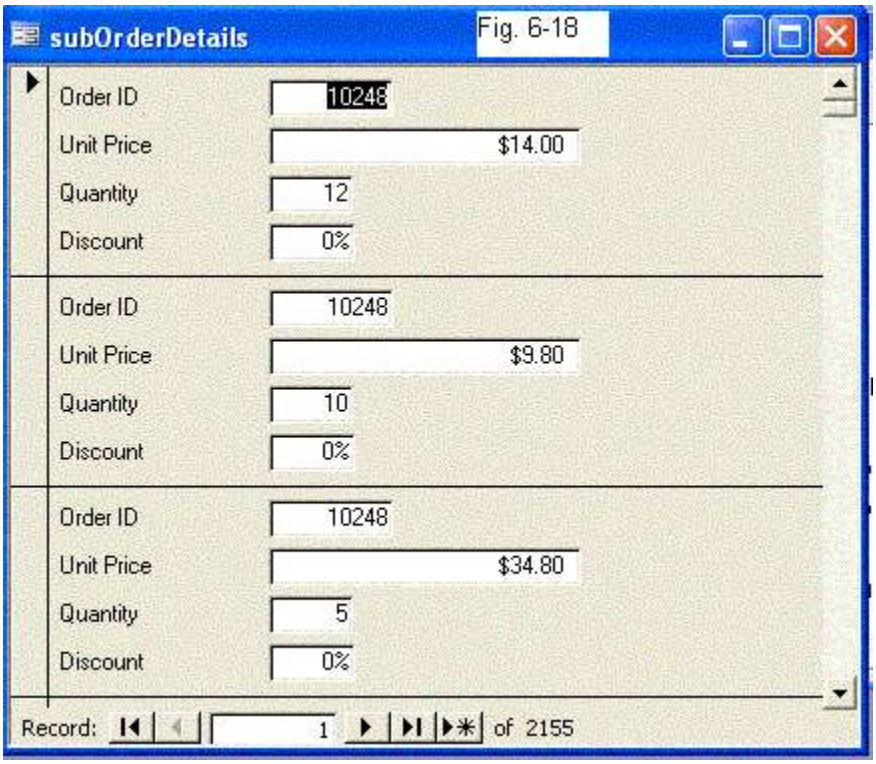
This is one of the options for designing a form with multiple selections. For example, there are two combo boxes on a form: the first is to select the state the customer is in, and the second box is the name of the customer. When you choose a state, you only want the combo box to list the customers of the selected state, not the entire customer list. By doing so, the processing of the parameters will increase in efficiency. If you select the category of products in the first combo box, the second box will list the names of the products that belong to the category chosen, rather than the entire list of products. Then, on the subform, it lists all orders that have this particular product. This flavor will help you design such a form.

By default, Access will not automatically connect two combo boxes on the same form when you create them. However, in order to make the above task work, you must make a parameter query for the second combo box with criteria that corresponds with the selection from the first combo box. It is very similar to the example in Flavor 6.1.

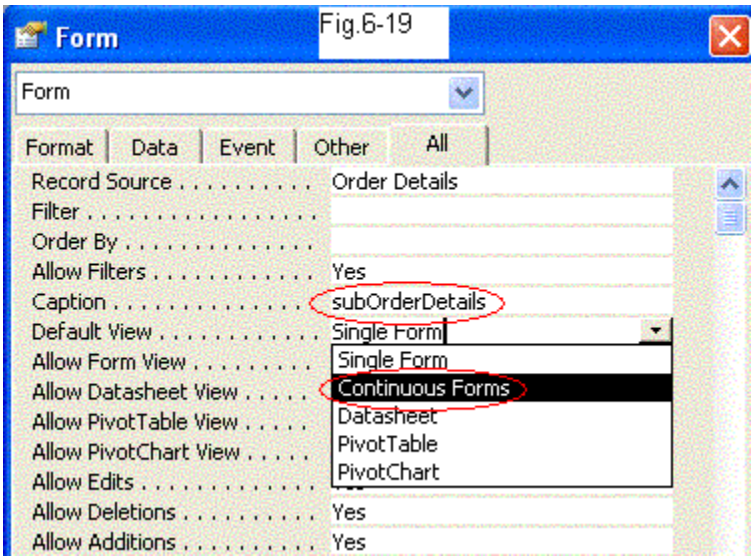
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

1. Creates a form based on the table **Order Details**. Select the fields as displayed in Fig. 6-18.



Switch to **Design View** to change the properties, like in Fig. 6-19: **Default View** to **Continuous Forms** and **Caption** to **subOrderDetails**.



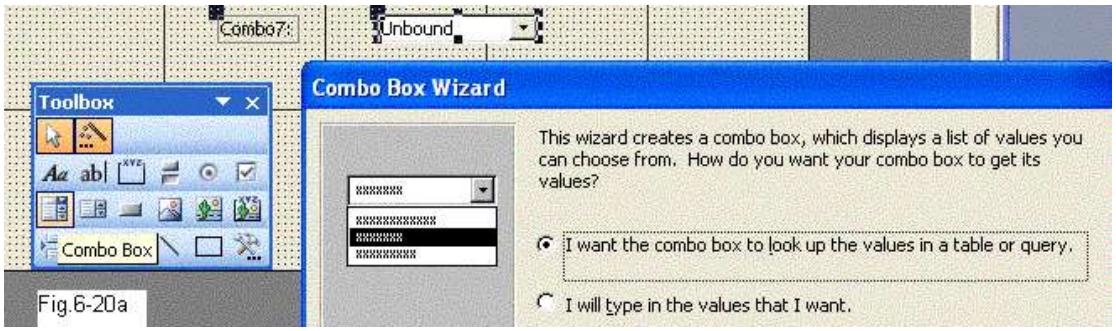
Save this from as **subOrderDetails** for use as a subform later on.

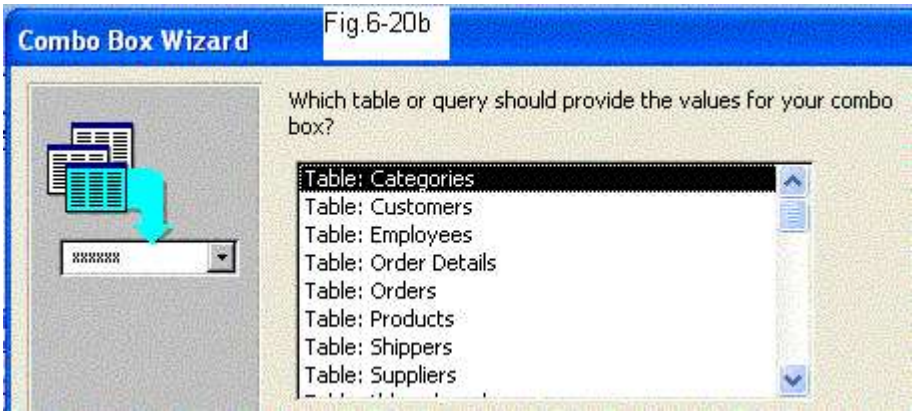
2. Create an unbound form that will be used as the main form. On this main form there will be two combo boxes: the first one, named **cboCateName**, is to list the names of categories and the second one, named **cboProdName**, is to list the names of products. Some quick images of how to create these two combo boxes are shown in Fig. 6-20a-h.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

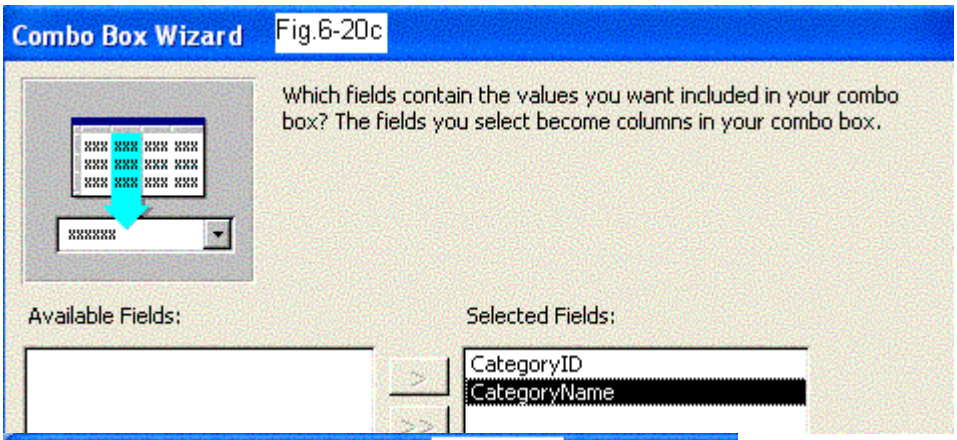
ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

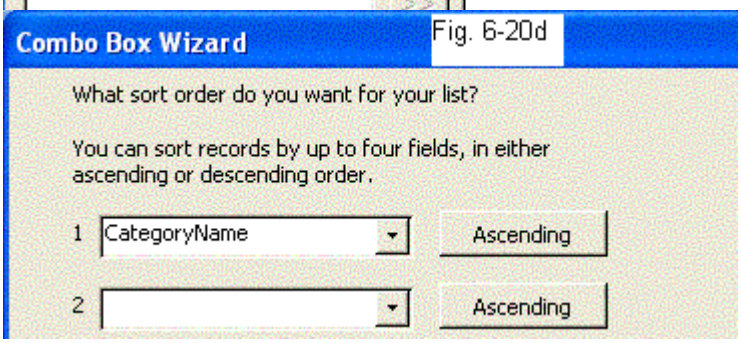
The first one to create is the **Category Name** combo box. You can take advantage of Control Wizard to create this object. The **Category Name** combo box will be built from table **Categories** as depicted in the next few figures.



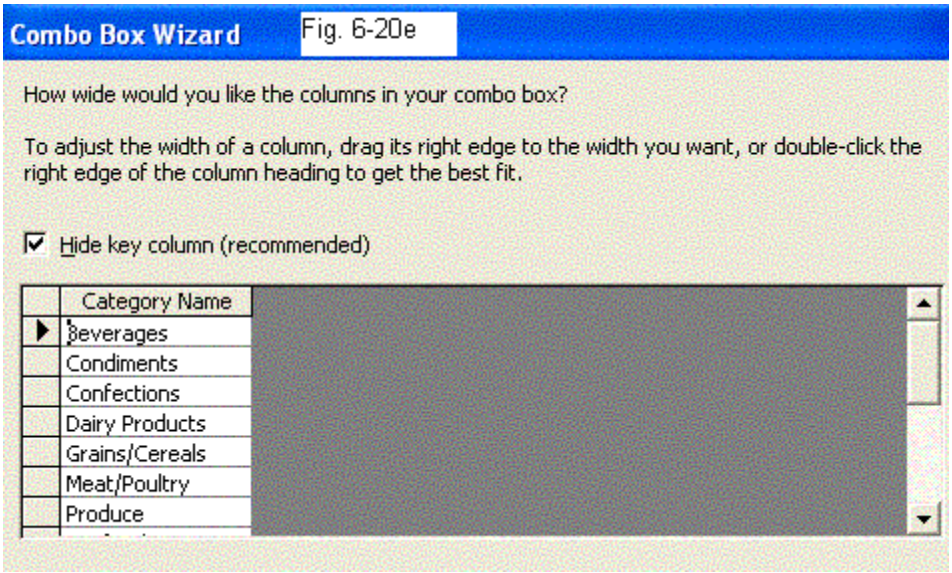


In Fig. 6-20c, it doesn't matter if you pick both or just one field, though one must be **CategoryName**.



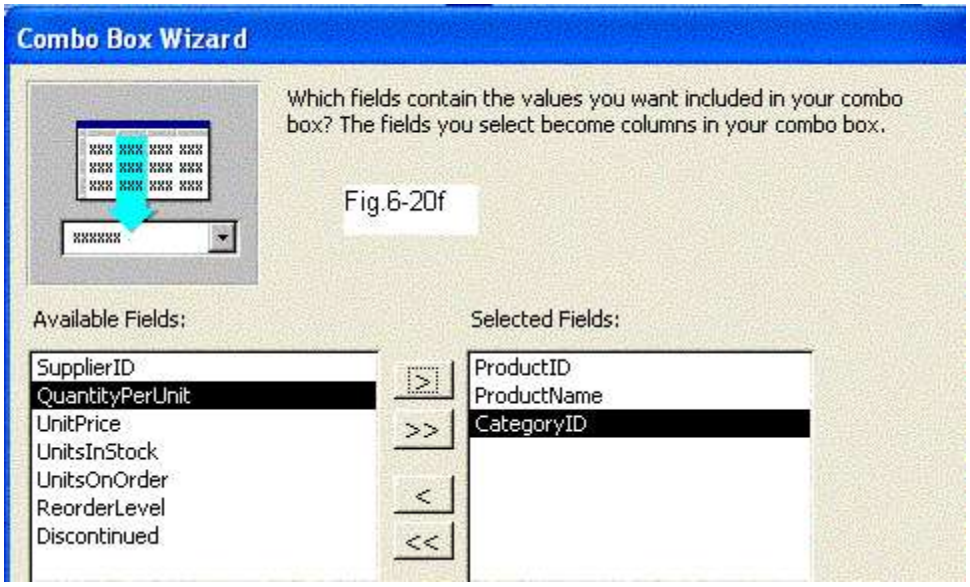


It is best to display sorted category names in the combo box.



Leave it as is. Click NEXT and keep the label name or type in a name you like. The last step is to close this Wizard by clicking the FINISH button.

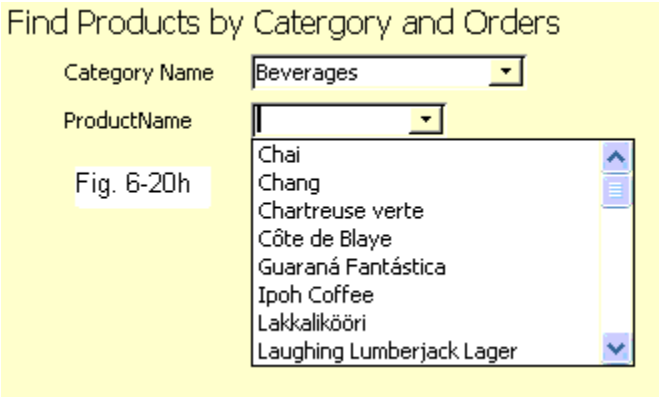
Use the same method to build the second combo box. Choose three columns: **ProductID**, **ProductName**, and **Category ID** from table **Products** and sort by **ProductName** (see Fig. 6-20f and 6-20g).



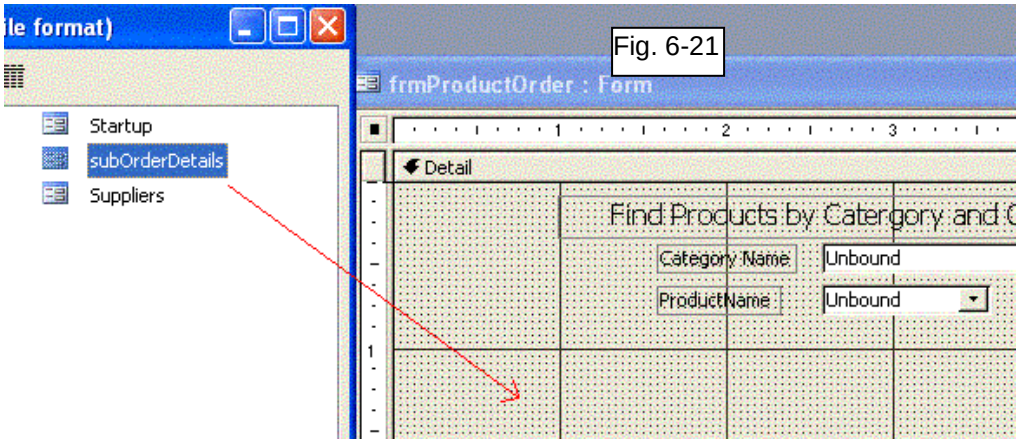
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

The complete design of the two combo boxes is shown in Fig. 6-20h.

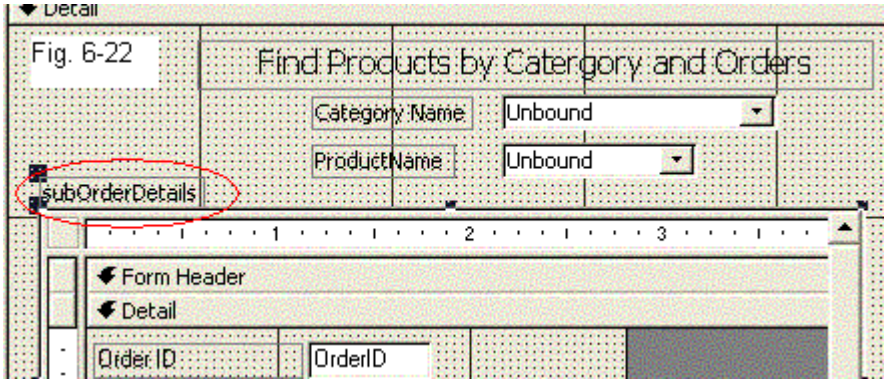


3. Save the main form, **frmProductOrder**, and open it in **form design view**. Drag the form, **subOrderDetails**, to the top of the **frmProductOrder**, as shown in Fig. 6-21. Then, remove the label of the **subOrderDetails** as illustrated in Fig. 6-22.

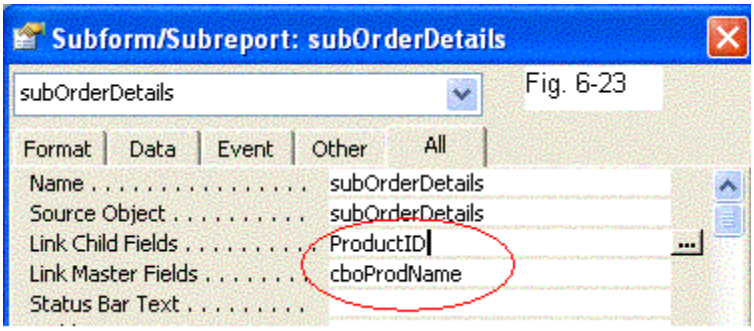


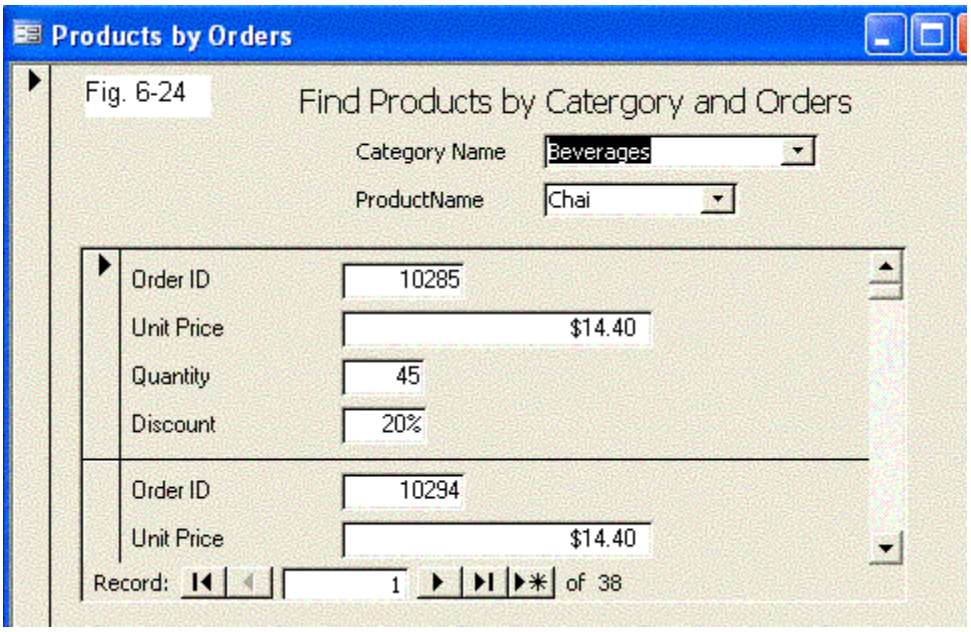
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



4. Now you need to make a connection between the main form and the subform. Open the property of **subOrderDetails**. In order to keep the subform in sync with the main form's two properties, **LinkChildFields** and **LinkMasterFields**, you need to enter **ProductID** and **cboProdName** into **subOrderDetails**, respectively, as shown in Fig. 6-23. Enter **Products by Orders** into **Caption** on the **frmProductOrder** like in Fig. 6-24.

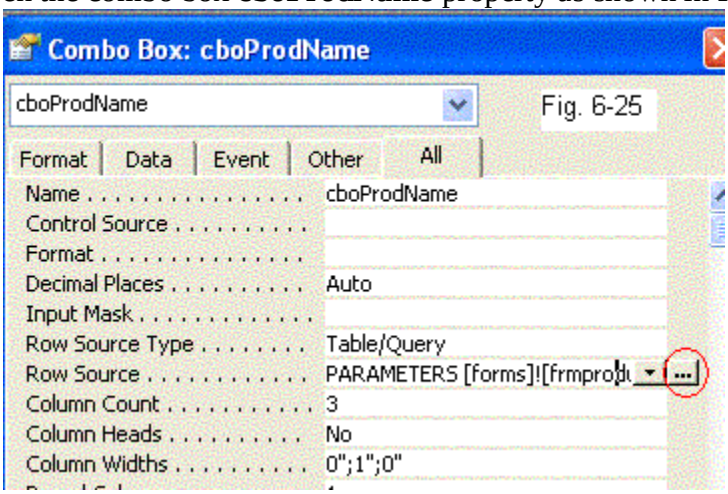




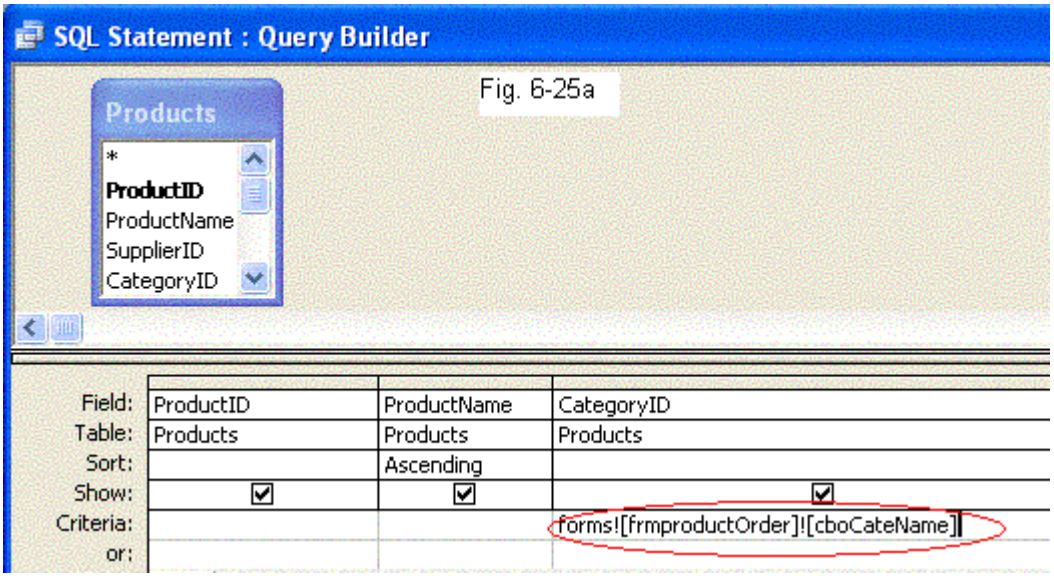
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

5. Form a sync mechanism between **Category Name** and **Product Name** combo boxes. When the category is picked, the products that are in that category will be displayed in the **Product Name** box for selection. For a similar case, you can refer to is the example in Flavor 6.1. Open the combo box **cboProdName** property as shown in Fig. 6-25.



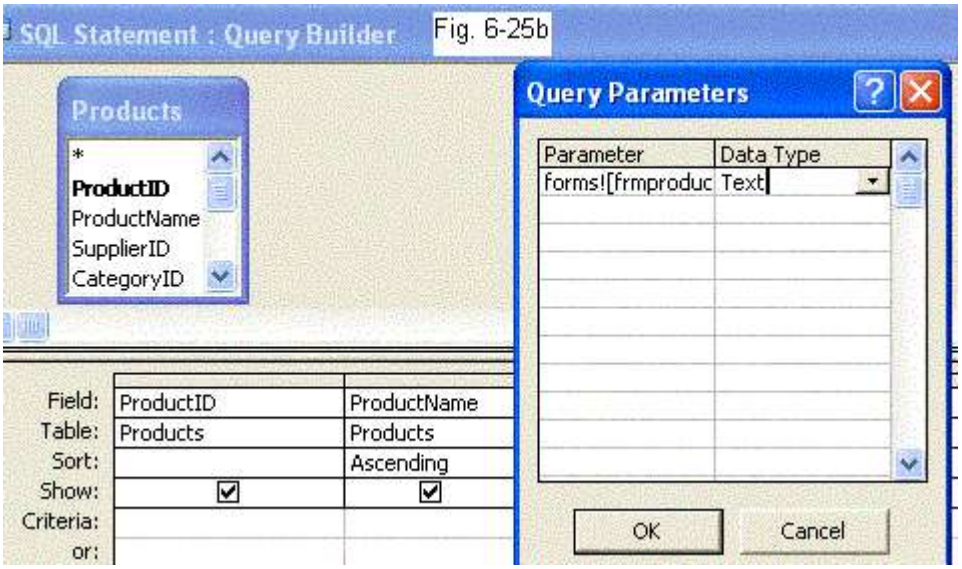
First, you will click inside the **Row Source** and then click the button to open the query builder**.** Enter **forms![frmProductOrder]![cboCateName]** (not case-sensitive) as the criteria of **CategoryID** in the query, as shown in Fig. 6-25a.



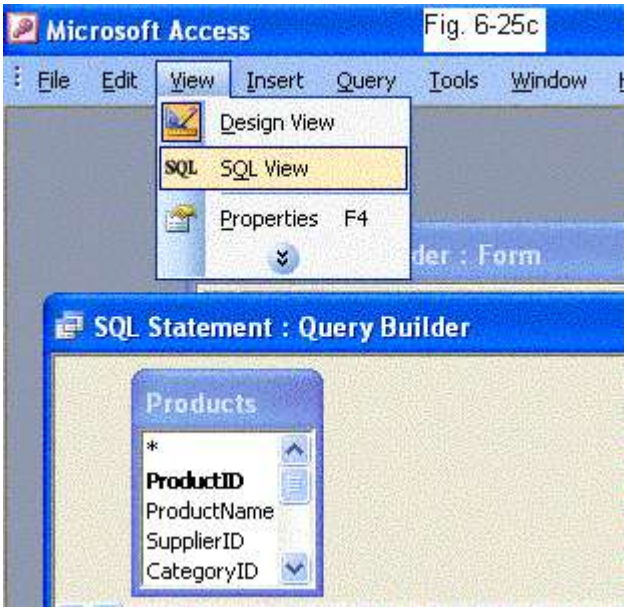
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

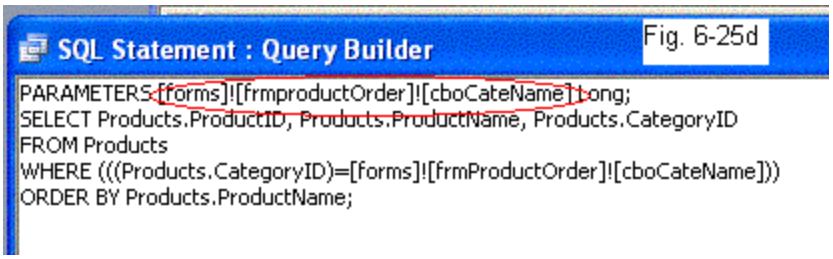
ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Next, you need to declare the data type of this parameter by going to menu, Query, Parameters to open the Query Parameters window (see Fig. 6-25b). However, the Data Type should be the same as the Data Type of **CategoryID** in the table **Category**, **Long Integer**. Click OK to close the windows of Query Parameters.



Before this parameter query can work, switch the **Query View** to **SQL view** as done in Fig. 6-25c. Remove the outside bracket on the criteria **forms![frmProductOrder]![cboCateName]** (refer to Fig. 6-25d).

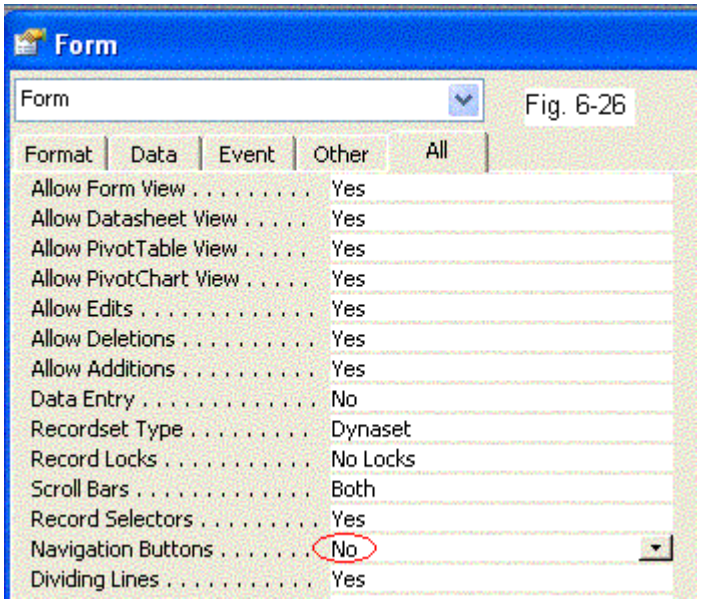




2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Before you move on to the next step, remove the navigator button on the main form by changing the value from **Yes** to **No** in the form property, as shown in Fig. 6-26.



6. When the form **Products by Orders** opens, no pre-selected item should be shown in either combo box. To make the combo boxes blank, click the **Event** tab on the **Products by Orders** form property. Choose the **Code Builder** when you click the ... on the right side of **On Open** event to open it. Enter the following code:

```
Private Sub Form_Open(Cancel As Integer)
    cboCateName = Null
    cboProdName = Null
End Sub
```

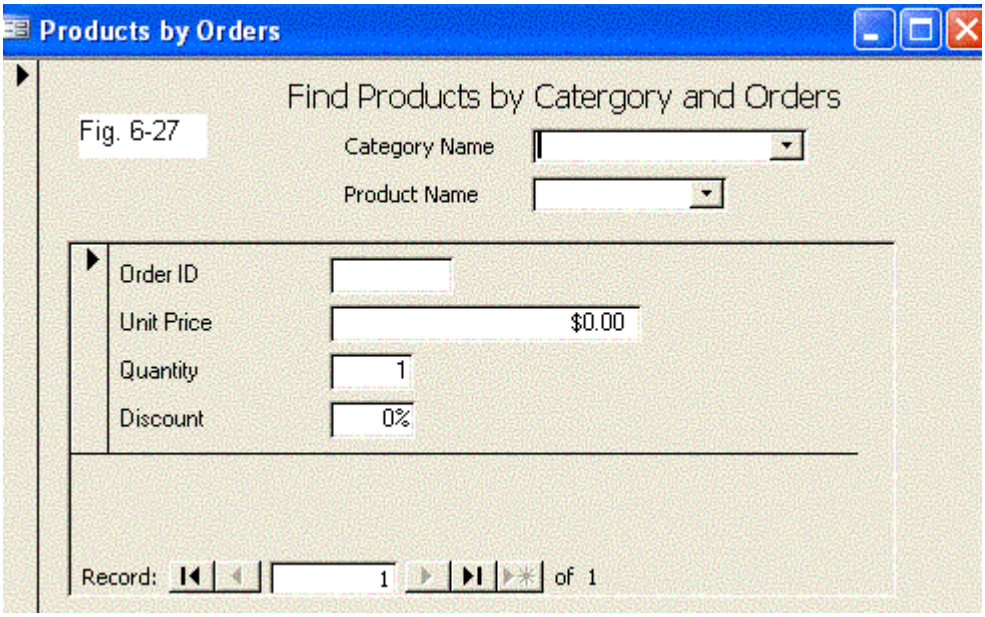
7. When the **Category Name** changes, you expect the second combo box, **Product Name**, to either be updated with a new value, or to requery if the selection in the second combo box changes, but is still under the same **Category Name**. Go to the properties of the combo box **cboCateName** and select the event **AfterUpdate**. Click ... on the right side of **AfterUpdate** to open the **Code Builder**. Enter the following code:

```
Private Sub cboCateName_AfterUpdate()
    cboProdName = Null
    cboProdName.Requery
End Sub
```

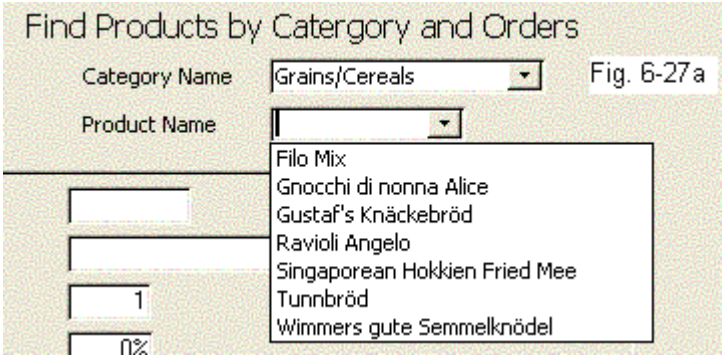
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

8. Now, it is time to test the form to see if it works. The following should occur if the form is correct:
After opening the form **Products by Orders**, the **Category** and **Product** names should be blank (as shown in Fig. 6-27);



The subform should be blank as well;
Choose one category (for example, Grains/Cereals); the **Product Name** box should stay blank until a selection is clicked from the drop-down list. The list of the **Product Name** will display the products that belong to the selected **Category Name** (refer to Fig. 6-27a).



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Click the "+" sign in front of the **Category ID** "5" to expand it.

Categories : Table Fig. 6-27b

Category ID	Category Name	
1	Beverages	Soft drinks, coffees
2	Condiments	Sweet and savory s
3	Confections	Desserts, candies,
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, p
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean
8	Seafood	Seaweed and fish
(AutoNumber)		

Record: 1 of 8

The products belonging to this category are shown in Fig. 6-27c.

Categories : Table Fig. 6-27c

Category ID	Category Name	
1	Beverages	Soft drinks, coffees, tea
2	Condiments	Sweet and savory sauc
3	Confections	Desserts, candies, and
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, past

Product ID	Product Name
22	Gustaf's Knäckebröd
23	Tunnbröd
42	Singaporean Hokkien Fried Mee
52	Filo Mix
56	Gnocchi di nonna Alice
57	Ravioli Angelo
64	Wimmers gute Semmelknödel
utoNumber)	

6	Meat/Poultry	Prepared meats

Record: 1 of 8

You can then compare the **Product Name** from Fig. 6-27a and Fig. 6-27c. They are identical except their order of display. It is understandable because one uses the order of **Product Name** and the other uses the order of **Product ID**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

The next step is to test the record on the subform. As shown in Fig. 6-27d, the **Product Name** is **Filo Mix**. Using **Filter by Selection**, you see there are 29 orders.

Order Details : Table

Fig. 6-27d

Order ID	Product	Unit Price	Qu
10322	Filo Mix	$5.60	
10334	Filo Mix	$5.60	
10388	Filo Mix	$5.60	
10446	Filo Mix	$5.60	
10449	Filo Mix	$5.60	
10517	Filo Mix	$7.00	
10558	Filo Mix	$7.00	
10563	Filo Mix	$7.00	
10586	Filo Mix	$7.00	
10594	Filo Mix	$7.00	
10620	Filo Mix	$7.00	
10683	Filo Mix	$7.00	

Record: 1 of 29 (Filtered)

Compare the results from Fig. 6-27d and Fig. 6-27e. They match.

Products by Orders

Find Products by Catergory and Orders

Fig. 6-27e

Category Name: Grains/Cereals

Product Name: Filo Mix

Order ID	10322
Unit Price	$5.60
Quantity	20
Discount	0%
Order ID	10334
Unit Price	$5.60

Record: 1 of 29

The test is completed and has been confirmed.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.4 Use a cookie cutter: create a template for all custom forms and reports table of contents

In a database project, a lot of time spent on making standard objects (i.e. forms and reports) could be saved if there was a template for use. Access allows you to create a form or report to be used as a template for new forms or reports. The procedure for making a form template is the same as making a report template.

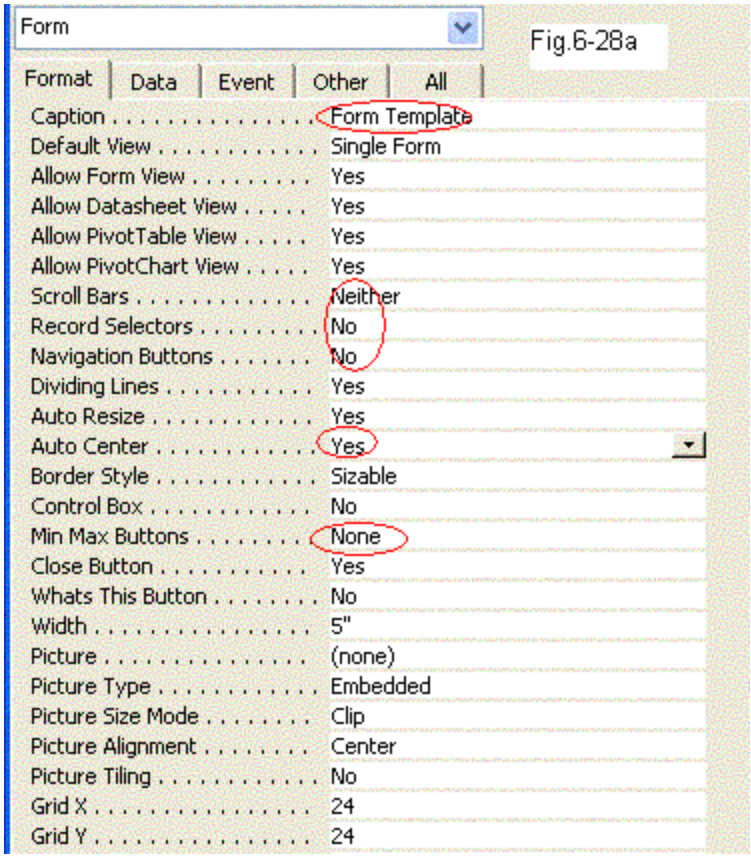
Open a blank form by clicking **New**, **Design View**, with no tables or queries as the data source, as shown in Fig. 6-28**.**

Switch to the **Form Property View** as shown in Fig. 6-28a. The following properties (Format tab) are suggested for your template settings:

Caption: name will appear on the form title bar
Record Selector: no need to have a record selector bar
Navigation button: no need to have a record navigator because there is no record
Minimize/maximize buttons: controls your **Form View**
Scrollbars: neither
Close button: if your form comes with the "close" feature, select NO
AutoCenter: select YES to center the form automatically when it opens

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

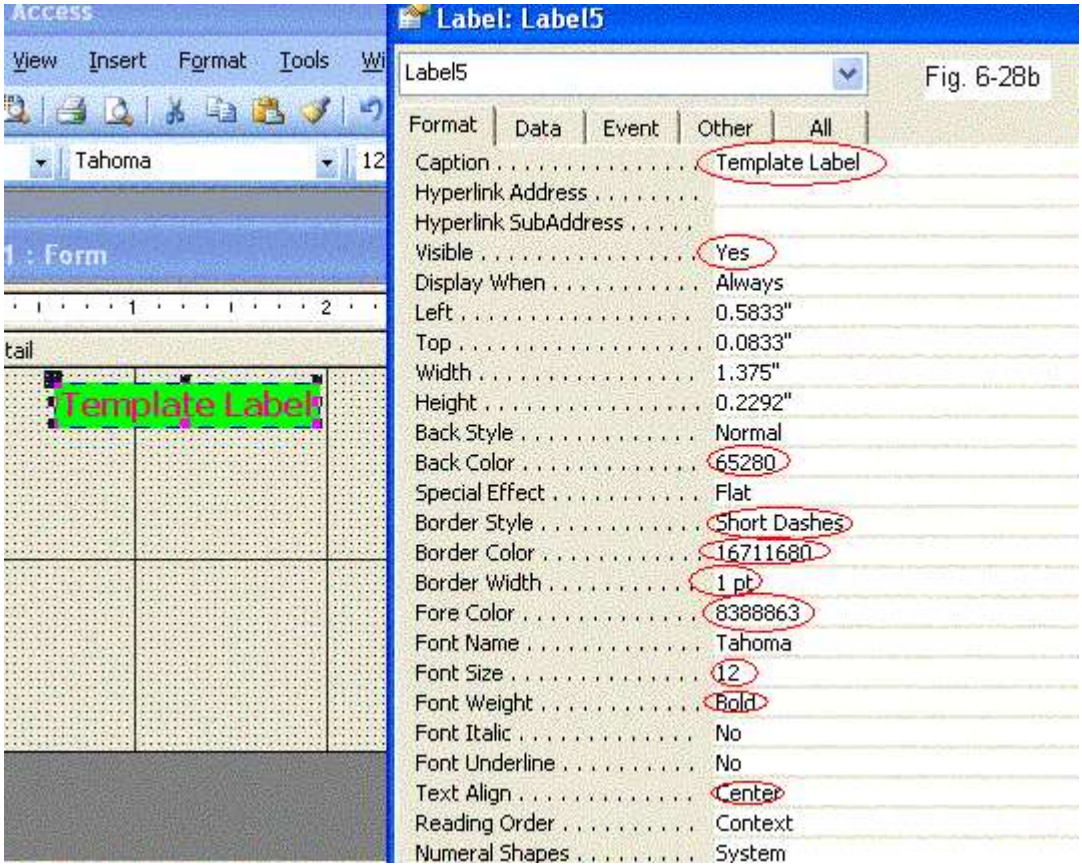
ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



Other properties are up to the individual's design preference, such as the form background color in the **Detail** section, Form header/footer section. In the Toolbox, you will be able to change the controls on the form (i.e. label, text box, combo box). Select all controls that could be needed for your design and make necessary changes in their properties. Remember that you don't need to use all the controls of the template when you make a new form. You can just choose what you need on the new form and leave unused controls invisible, by changing their properties, or simply delete them. As can be seen in Fig. 6-28b, the label box's colors, text size, border style, and alignment can be changed, among other properties. These properties are set as defaults on the template. Later on, if you need different values or designs, you can modify their properties.

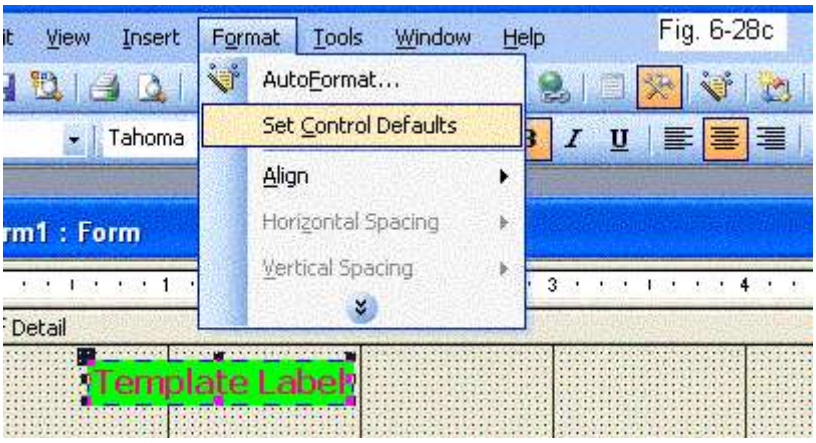
2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



This template is not done yet! The following two steps must be completed before you have a finished template.

1. All controls have the **Visible** property set as **Yes**. For all controls, select the menu, **Format, Set Control Defaults** (see Fig. 6-28c and Fig. 6-28d). Save the template under a name you like. Access usually uses a form named Normal as its form template, which is hidden from users. It is the same design for a report template. If you don't create a template, then Access will use its built-in form, which is what you see whenever you click **New** to create a new form or report. Of course, if you make a customized form a template and name it **Normal**, then Access will use this form as the default template. If you name this template form with a different name, you can still ask Access to use the template you create as the default by completing step 2.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

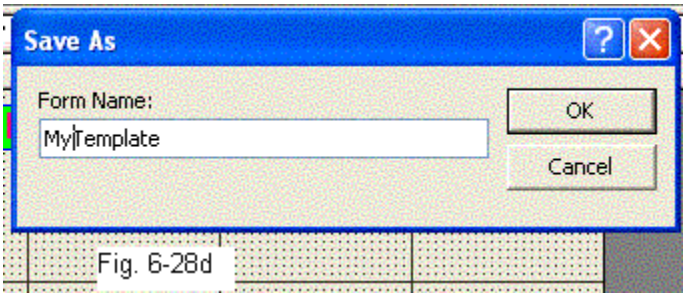


Fig. 6-28d

2. Open menu, **Tools**, **Options**. Click the tab **Forms/Reports** as shown in Fig. 6-28e.

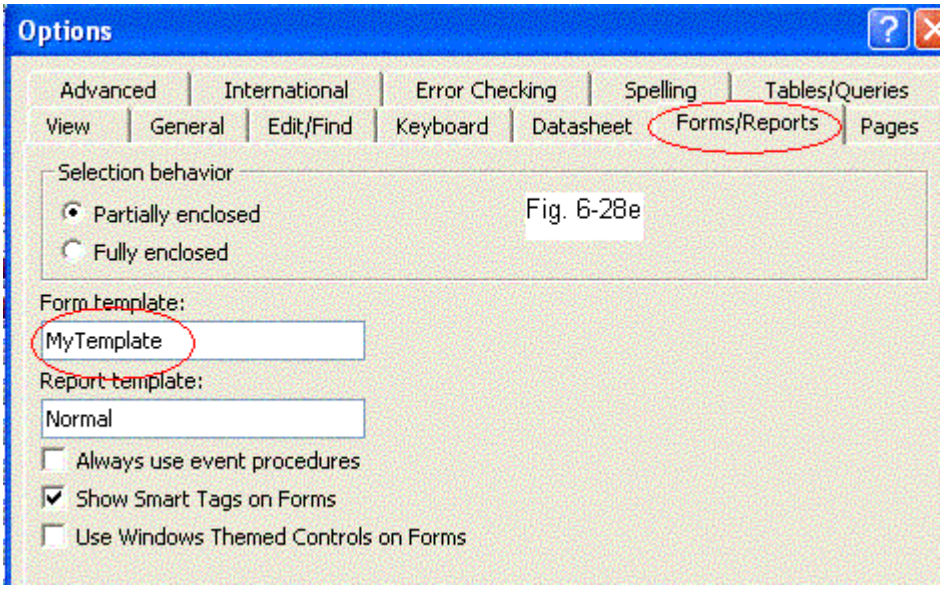


Fig. 6-28e

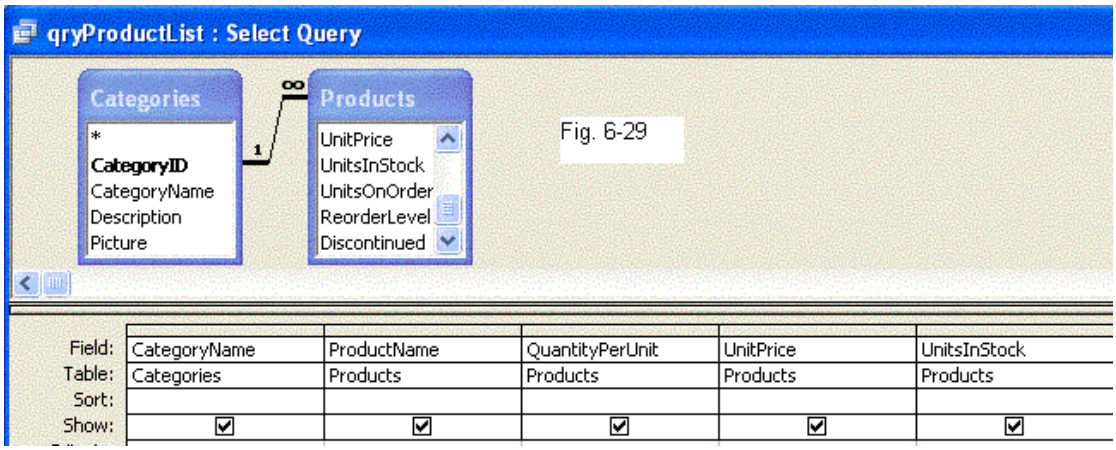
Change the form template from **Normal** to the name of the template you created. In Fig. 6-28e, the example is **MyTemplate**.

Flavor 6.5 Display the parameter value on a report

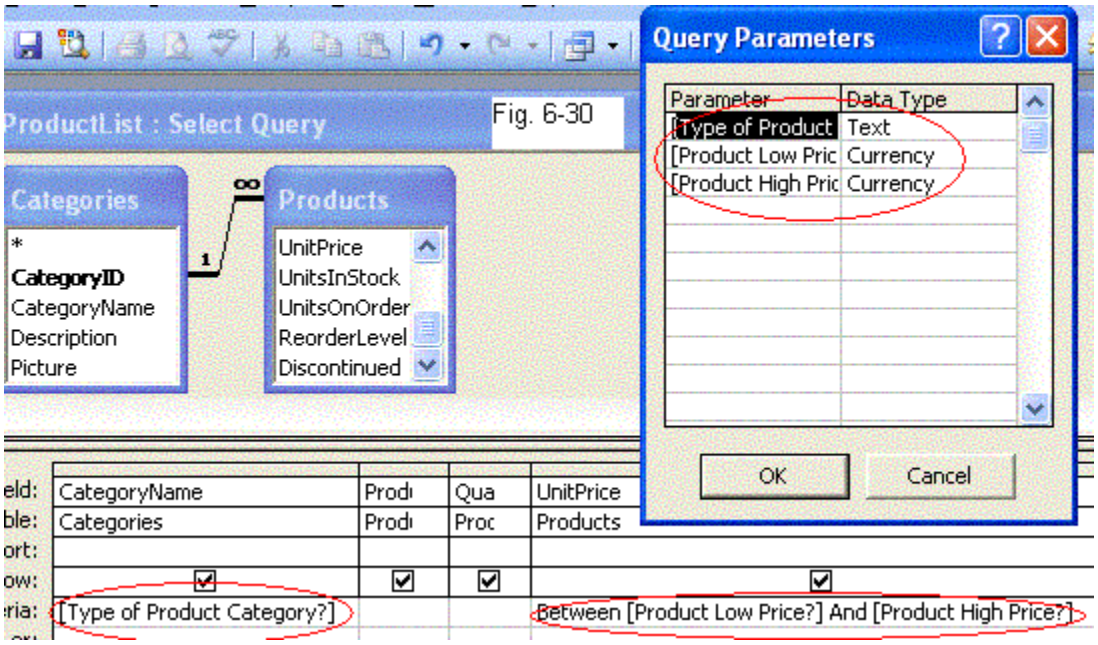
table of contents

The parameter query filters the record by taking parameter input from users and placing it into the criteria of the query. It is best if these parameters are shown on the report, because then users know what parameters they should input. There are a couple different ways to make this happen. You can integrate 6.1 and some 6.2 examples with this design. Here is the simplest way: create a parameter query and prompt for the input parameters. The report will display the result of the query.

1. Make a new query, **qryProductList**, as shown in Fig. 6-29, from the table **Products** and **Categories** with three parameters: **Category Name**, **Product Low Price** and **Product High Price**.



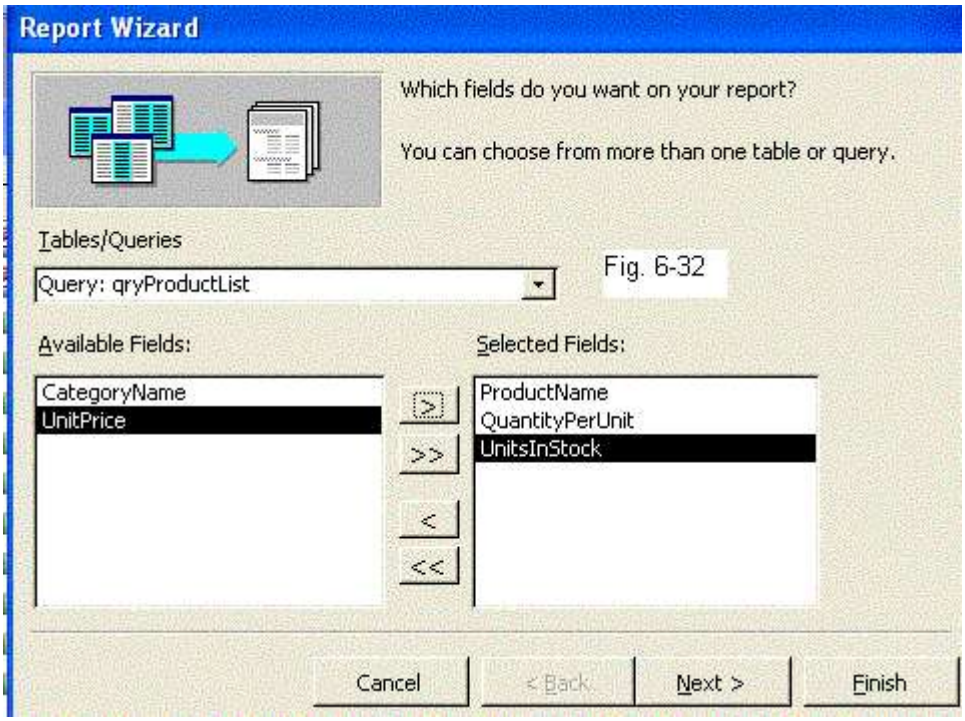
2. Declare the parameter in both the parameter dialog window and in the criteria of the QBE, as shown in Fig. 6-30.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

3. Create a report by using the report Wizard, as shown in Fig. 6-31. This report will be based on the result of the query **qryProductList**. Only three fields are on the report (see Fig. 6-32). The report will be sorted by product name and named **rptProductList**.



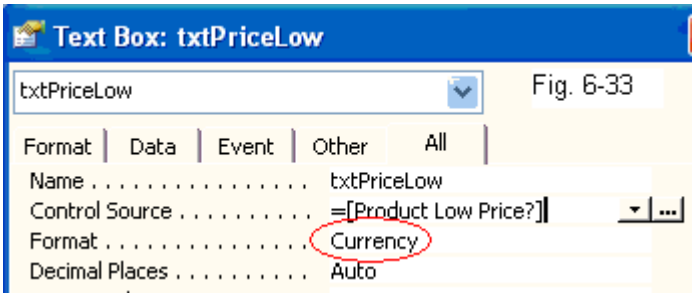
4. Two textboxes will be added onto this report, in the **Page Header**, **Report Header** section or any other place you prefer. These two boxes will display the name of the category and the product price range as was input into the parameter query. Their properties are as follows:

Object name	**Control source**	**Format**
txtCateName	=[Type of Product Category?]	
txtPriceLow	=[Product Low Price?]	Currency
txtPriceHigh	=[Product High Price?	Currency

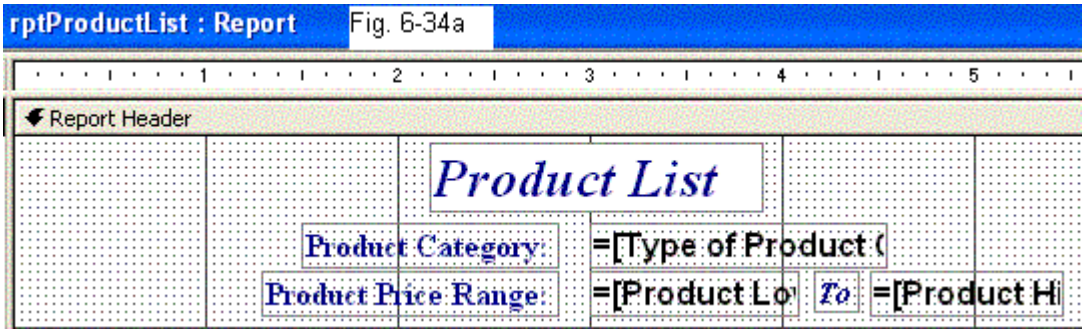
Fig. 6-33 shows one of the textbox's properties.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



The above three objects are important because they display the values of the parameters, which fit the criteria used in the query. You may also want to rename the label boxes to reflect its contents, such as **Category Name** for the **txtCateName** box. The final design is shown in Fig. 6-34a and 6-34b in **Design View** and print preview.



Test following
Enter Category Name: seafood
Enter Low Price: 10
Enter High Price: 100
into three parameter prompts.

Fig. 6-34b

Product List

Product Category seafood

Product Price Range $10.00 *To* $100.00

Product Name	*Quantity Per Unit*	*Units In Stock*
Boston Crab Meat	24 - 4 oz tins	123
Carnarvon Tigers	16 kg pkg.	42
Escargots de Bourgogne	24 pieces	62
Gravad lax	12 - 500 g pkgs.	11
Ikura	12 - 200 ml jars	31
Inlagd Sill	24 - 250 g jars	112
Nord-Ost Matjeshering	10 - 200 g glasses	10
Röd Kaviar	24 - 150 g jars	101
Spegesild	4 - 450 g glasses	95

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.6 Add line numbers on the report

table of contents

On the report, you need to add the sequential number on each item, which will not change when the content of each item changes. A property, **RunningSum**, of a textbox is the key component in this task. Open the report **Sales Totals by Amount** in **Design View**.

1. Add a new, unbound textbox to the **Detail** section either on the far right (i.e. the last column on the report print preview) or far left (i.e. the first column on the report print preview) with the following properties:
ControlSource value =1,
Name the box **Counter**,
Running Sum change from No to **Over All**
as shown in Fig. 6-35.

2. Save the report and preview it to see if the Counter works, as shown in Fig. 6-35a.

Sale Amount:	Order ID:	Company Name:	Counter:
$11,188	10417	Simons bistro	1
$10,496	10479	Rattlesnake Canyon Grocery	2
$10,192	10540	QUICK-Stop	3
$10,165	10691	QUICK-Stop	4
$9,921	10515	QUICK-Stop	5
$9,195	10424	Mère Paillarde	6
$8,623	10514	Ernst Handel	7
$6,635	10776	Ernst Handel	8
$6,475	10607	Save-a-lot Markets	9

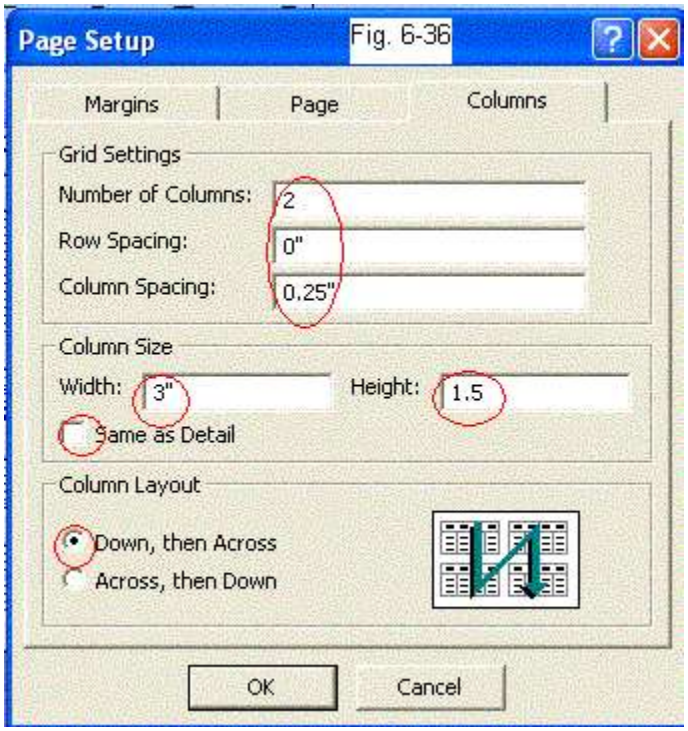
Fig. 6-35a

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.7 Make a report with multiple columns table of contents

On the **Report Wizard** you can create a report with only one column. There is no **Column** setup property to create a multiple-column report. Instead, you will look for the **Page Setup** dialog window from the menu: **File**, **Page Setup** (see Fig. 6-36).



1. Make sure that, before opening the **Page Setup**, you open the report that is to have multiple columns, **rptSuppliers**, in **Design View**. Enter the following values into the dialog window:

Number of Columns: 2
Row Spacing: 0
Column Spacing: 0.25
Column Size: Width: 3
Height: 0.25
Same as Detail: unchecked
Column Layout: Down, then Across

Column Spacing is the space between columns, which can be tailored for your design. For **Same as Detail**, Access will copy the width and height of the **Detail** section on the report into the width and height controls. It will not add more columns if this box is checked. For mailing-label style columns, you may want to pick "**Across, then Down**" instead.
Click OK when you complete the entry.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

2. Let's look at the report before and after the **Page Setup** changes: Fig. 6-36a and 6-36b, respectively.

rptSuppliers

Company Name	Aux joyeux ecclésiastiques
Supplier ID	18
Contact Name	Guylène Nodier
Contact Title	Sales Manager
Address	203, Rue des Francs-Bourgeois
City	Paris
Region	
Postal Code	75004
Country	France
Phone	(1) 03.83.00.68
Fax	(1) 03.83.00.62
Home Page	
Company Name	Bigfoot Breweries
Supplier ID	16
Contact Name	Cheryl Saylor
Contact Title	Regional Account Rep.
Address	3400 - 8th Avenue
City	Bend

Fig. 6-36a

rptSuppliers

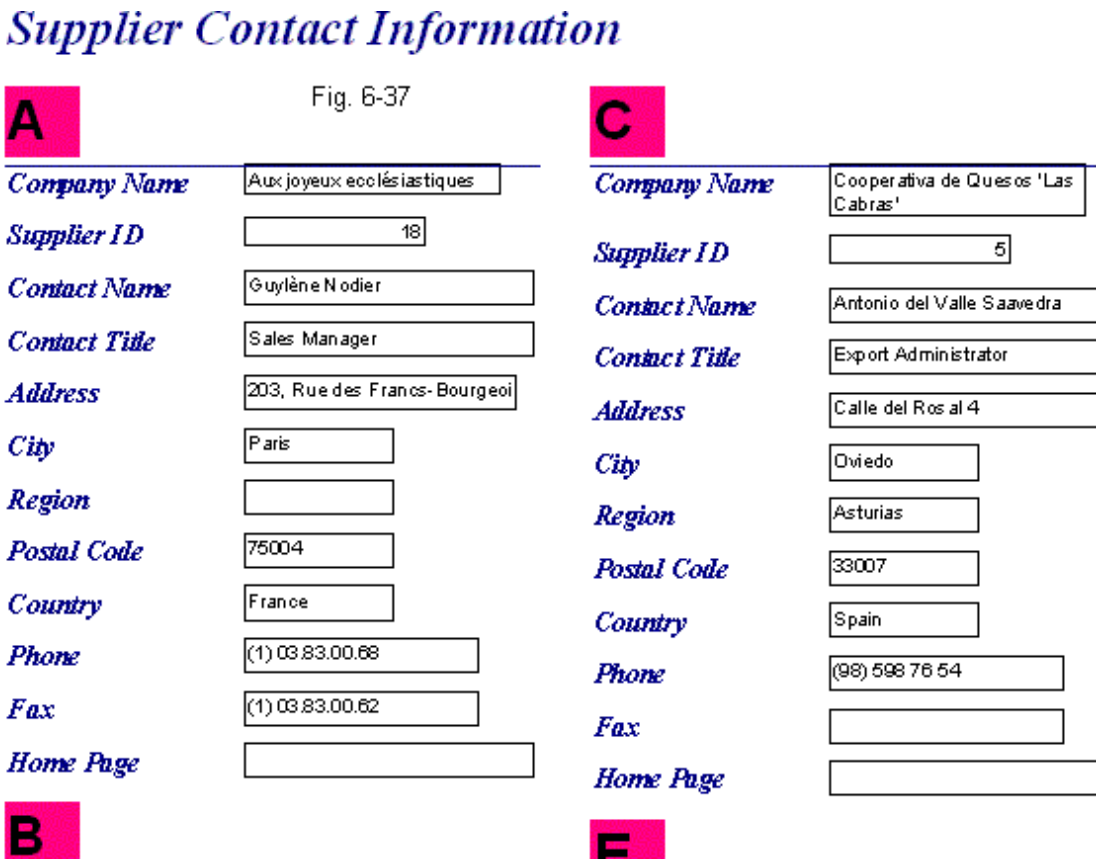
Fig. 6-36b

Company Name	Aux joyeux ecclésiastiques	*Company Name*	Cooperativa de Quesos 'Las Cabra
Supplier ID	18	*Supplier ID*	5
Contact Name	Guylène Nodier	*Contact Name*	Antonio del Valle Saavedra
Contact Title	Sales Manager	*Contact Title*	Export Administrator
Address	203, Rue des Francs-Bourgeois	*Address*	Calle del Rosal 4
City	Paris	*City*	Oviedo
Region		*Region*	Asturias
Postal Code	75004	*Postal Code*	33007
Country	France	*Country*	Spain
Phone	(1) 03.83.00.68	*Phone*	(98) 598 76 54
Fax	(1) 03.83.00.62	*Fax*	
Home Page		*Home Page*	
Company Name	Bigfoot Breweries	*Company Name*	Escargots Nouveaux
Supplier ID	16	*Supplier ID*	27
Contact Name	Cheryl Saylor	*Contact Name*	Marie Delamare
Contact Title	Regional Account Rep.	*Contact Title*	Sales Manager
Address	3400 - 8th Avenue	*Address*	22, rue H. Voltor
City	Bend	*City*	Montbeau

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

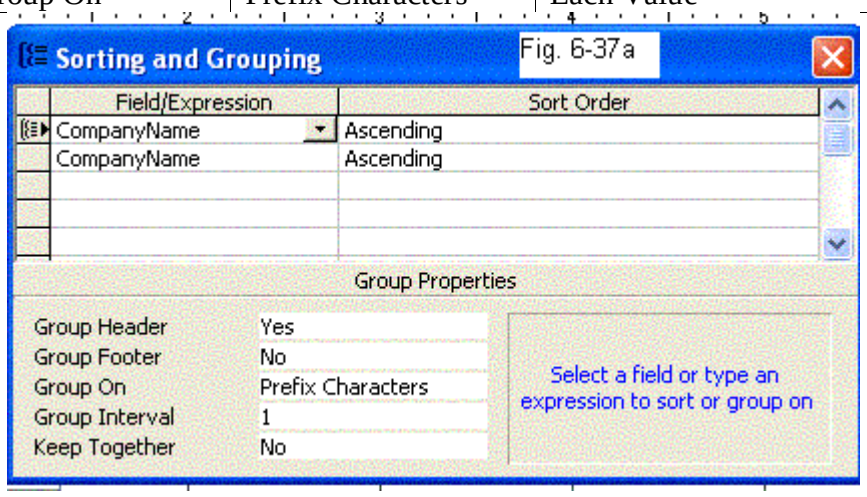
You can further improve the flavor of your dish by adding **Letter Grouping**, as highlighted in Fig. 6-37.



Follow the instructions below to make this first letter grouping on the report. You can do this on the phone book as well.

1. Open the **Sorting and Grouping** dialog box by selecting menu, **View**, **Sorting and Grouping**. Add the field **CompanyName** twice, as done in Fig. 6-37a, with the following settings and keep the rest of the properties as default (no change):

Group Properties	Top CompanyName	Botton CompanyName
Group Header	Yes	No
Group On	Prefix Characters	Each Value



2006-2009 © All Rights Reserved.

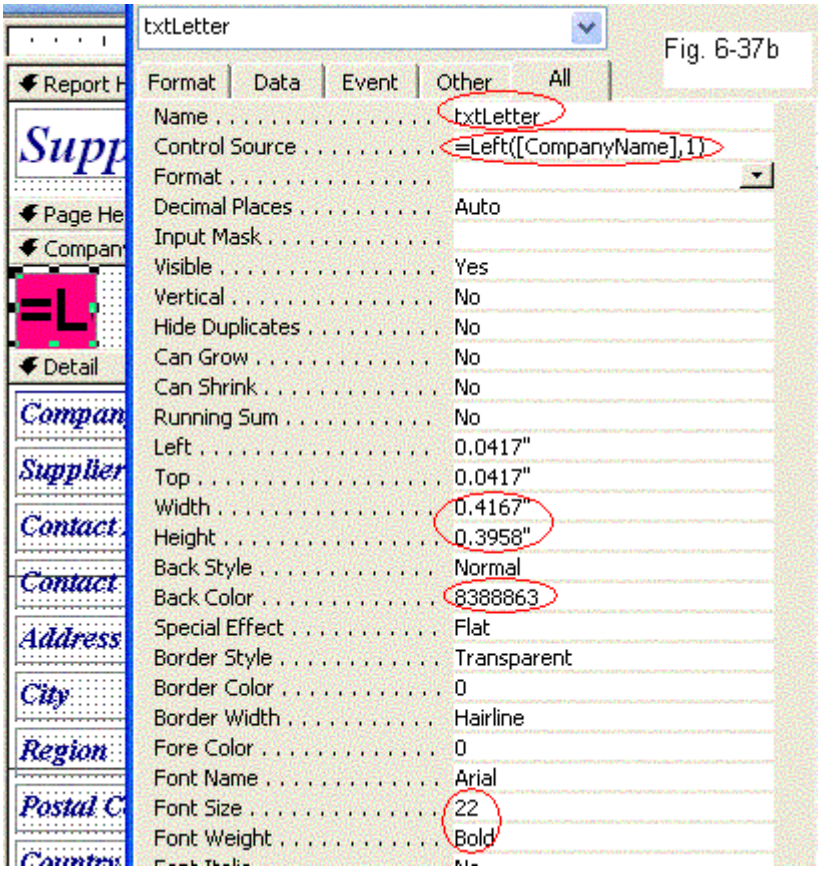
ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

2. Add an unbound textbox to the **CompanyName Header** section, which is the grouping field, as shown in Fig. 6-37b with the following properties settings:

Property	**Setting**
Name	txtLetter
Control Source	=Left([CompanyName],1)
Font size	22
Font weight	Bold

You may want to use the **Expression Builder** to complete the **Control Source** setting. You can use the mouse to make adjustments if you are not sure what values for the width and height should be. Regarding the **Back Color**, you can click ... next to the right-side of the **Back Color** box to visualize the color palette and choose your color, rather than memorizing the color code (see Fig. 6-37b). Other properties are up to the individual's preference.

3. Change the label box from to **Supplier Contact Information** to **rptSuppliers**. Save the report.

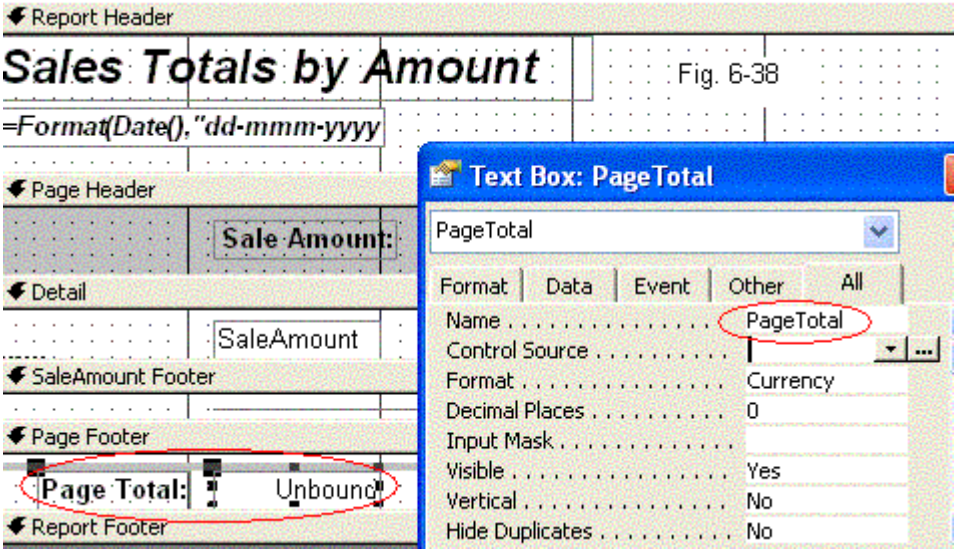


Flavor 6.8 Total on each page

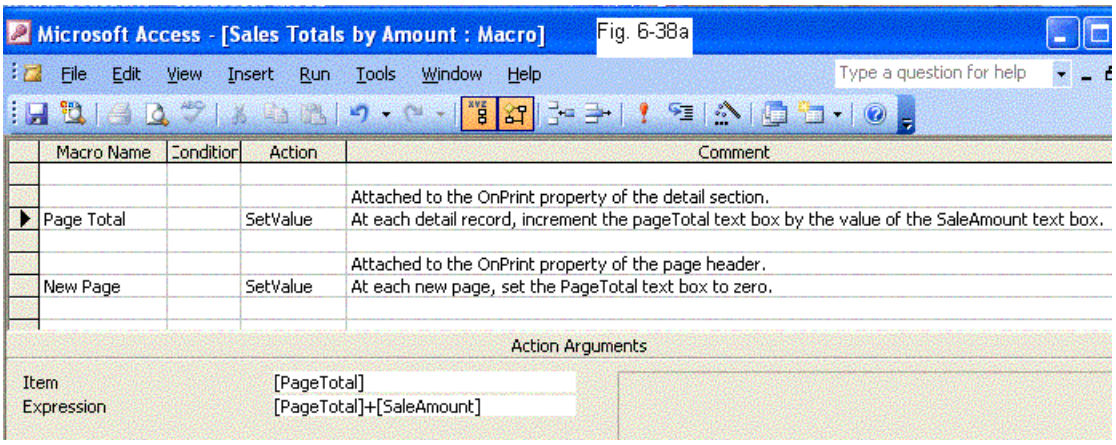
table of contents

If you place a group total in the group footer (i.e. subtotal of each group) of a report, it signifies the subtotal of each group. However, if you put the total in the report footer, it becomes the grand total. There is no direct way to create a page total or subtotal, as Access only allows cumulative calculations to be in a group or report footer. Here is the suggested method to create a page total.

1. Create a report, **Sales Totals by Amount**, and sort or group the data if necessary. Add a textbox, **PageTotal**, into the page footer, as shown in Fig. 6-38.



2. Create two macros, **Page Total** and **New Page**, with a macro group, **Sales Total by Amount**, as illustrated in Fig. 6-38a. Here are the details of each macro:



Macro Name: **Page Total**	Macro Name: **New Page**
Action: **SetValue**	Action: **SetValue**
Item: **[PageTotal]**	Item: **[PageTotal]**
Expression: **[PageTotal]+[SaleAmount]**	Expression: 0

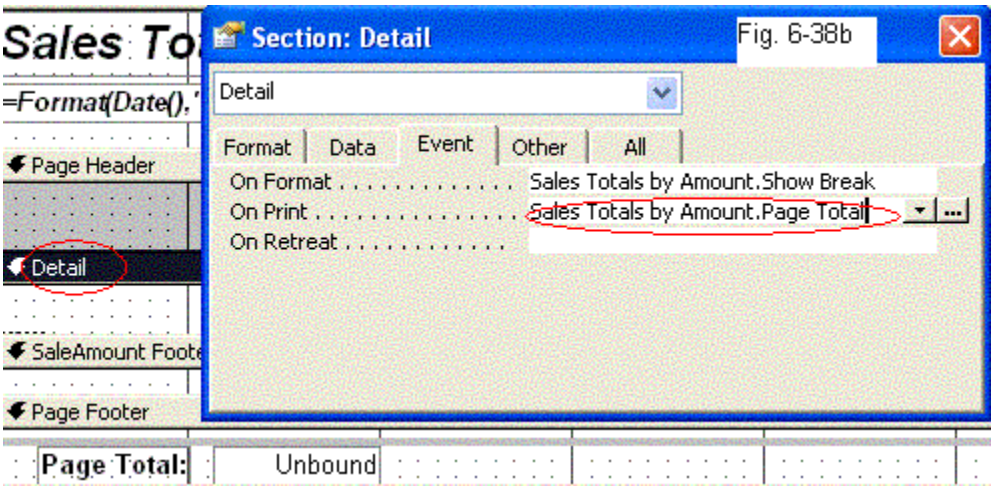
In the **Item** and **Expression** of the Action Arguments, click ... to use the **Expression Builder** to create the arguments. The **New Page** macro will reset the textbox **PageTotal** to zero. The **Page Total** macro should create a running total value of **SaleAmount** in the textbox **PageTotal**. For each new page, the **Page Total** control resets to

2006-2009 © All Rights Reserved.

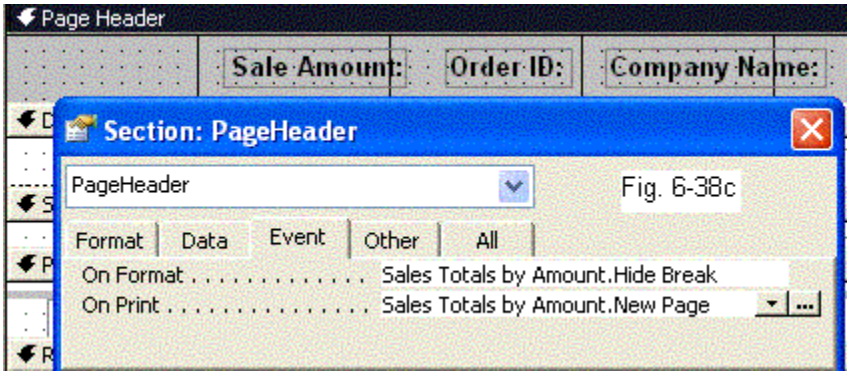
ISBN 0-9787733-0-6

zero. For each record on a page, this textbox will increase by the value of the **SaleAmount**. When Access reaches the last record on the page, the value of the **PageTotal** is the total sales amount for the page.

3. You will assign the macro **Page Total** to the event **On Print** in the **Detail** section, as Fig. 6-38b shows.



4. You will assign the second macro, **New Page**, to the **Page Header** section, also with the **On Print** event, as shown in Fig. 6-38c.



Why shouldn't you assign the macro **PageTotal** to the event **OnFormat**? If you do so, Access will include the value without knowing if the record fits on the current page or not. When the **OnFormat** event occurs, Access has not yet laid out the section for printing. If the record does not fit on the page, Access will move it to the next, while still including the records' value in the first page's total. However, if the **PageTotal** macro is assigned to the event **OnPrint**, the macro runs only the section that is to be printed on the current page. The page total will then have accurate, current records.

5. Save the report.

You may want to print the totals in the page footer rather than those in the group footer, because when Access prints a group footer directly from the detail records, it may end up anywhere on the page. To hide the group footer, create a macro **Hide Footer** and set it so that it will not display the group footer if there are 10 or less records. This macro will be assigned to the **SaleAmount** footer under the name **GroupFooter0** on the event **OnFormat**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Hide the page break control by creating a macro, **Hide Break**, and set its **Visible** value to **No**, as shown in Fig. 6-39. Then assign this macro to the **PageHeader** on the event **OnFormat**. Its purpose is to hide the page break control for every page, except those that satisfy the conditions of the **Show Break** macro, which is when there are 10 records on the page.

Macro Name	Condition	Action	Comment
			Attached to the Sales Totals by Amount report.
			Attached to the OnFormat property of the page header.
Hide Break		SetValue	Hide the page break control when the report opens and for every page
			Attached to the OnFormat property of the detail section.
Show Break	[Counter]=10	SetValue	Display the page break control when the value of the Counter text box
			Attached to the OnFormat property of the SaleAmount footer.
Hide Footer	[Counter]=10	CancelEvent	Don't print the footer when the value of the Counter text box is 10.
			Attached to the OnPrint property of the detail section.
Page Total		SetValue	At each detail record, increment the pageTotal text box by the value of
			Attached to the OnPrint property of the page header.
New Page		SetValue	At each new page, set the PageTotal text box to zero.

Fig. 3-39

Action Arguments

Item	[HiddenPageBreak].[Visible]
Expression	No

As shown under the **Condition** column in Fig. 6-39a, there should be 10 records per page, **[Counter]=10**. If this is met, then the page should end here, making the page break visible. Refer to the line number example in 6.6 to see how the control **Counter** textbox works. Assign this macro, **ShowBreak**, to the **Detail** section with the event **OnFormat**.

			Attached to the OnFormat property of the detail section.
Show Break	[Counter]=10	SetValue	Display the page break control when the value of the Counter text box is 10 (representing
			Attached to the OnFormat property of the SaleAmount footer.
Hide Footer	[Counter]=10	CancelEvent	Don't print the footer when the value of the Counter text box is 10.
			Attached to the OnPrint property of the detail section.
Page Total		SetValue	At each detail record, increment the pageTotal text box by the value of the SaleAmount te:
			Attached to the OnPrint property of the page header.
New Page		SetValue	At each new page, set the PageTotal text box to zero.

Fig. 6-39a

Action Arguments

Item	[HiddenPageBreak].[Visible]
Expression	Yes

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

6. Open the report in print preview to see the changes (Fig. 6-40).

$4,452	10711	Save-a-lot Markets	24
$4,372	10657	Save-a-lot Markets	25
$4,337	10762	Folk och fä HB	26
$4,180	10530	Piccolo und mehr	27
$4,150	10518	Tortuga Restaurante	28
$4,110	10605	Mère Paillarde	29
$3,891	10458	Suprêmes délices	30
$3,869	10401	Rattlesnake Canyon Grocery	31
$3,850	10451	QUICK-Stop	32
$3,815	10672	Berglunds snabbköp	33
$3,687	10789	Folies gourmandes	34
$3,603	10772	Lehmanns Marktstand	35
$3,554	10549	QUICK-Stop	36
$3,463	10718	Königlich Essen	37
$3,436	10698	Ernst Handel	38
$3,424	10709	Gourmet Lanchonetes	39
$3,193	10524	Berglunds snabbköp	40
$3,163	10490	HILARIÓN-Abastos	41
Page Total: $131,811			Page 2 of 3

Fig. 6-40

Flavor 6.9 Add a content range indicator on each page table of contents

A report contains a large amount of data. It takes time to look for a specific piece on a page. If each page contains a data range of its information, you will save a lot of time on searching for data. Such a design can be called a data-range indicator. There are many ways to create one. Microsoft also provides a sample database, Solutions9.mdb, for you to try it on their Microsoft MSDN website.

Here is the simplest method to make a data-range indicator. Create a report or open an existing one, **rptProductListwithRange**, in **Design View**.

1. In the **Page Header** section, put a textbox named **txtFirstOne** with the property **Visible** changed to **No** (refer to Fig. 6-41).

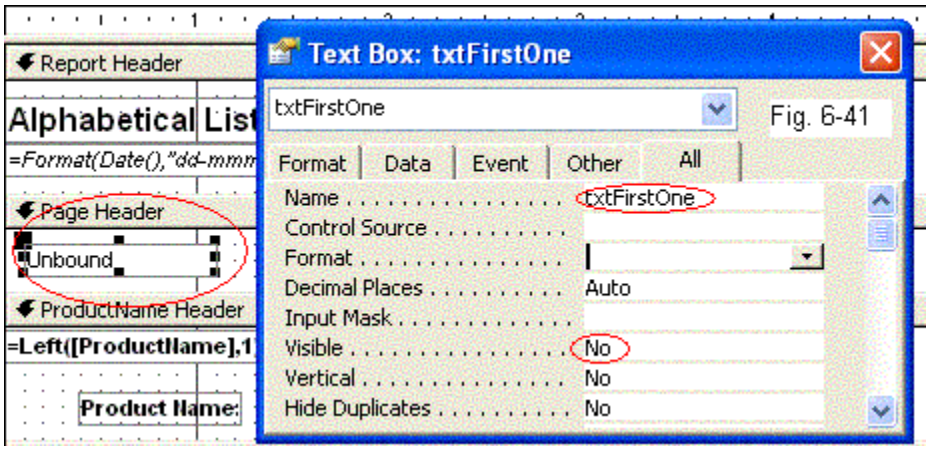


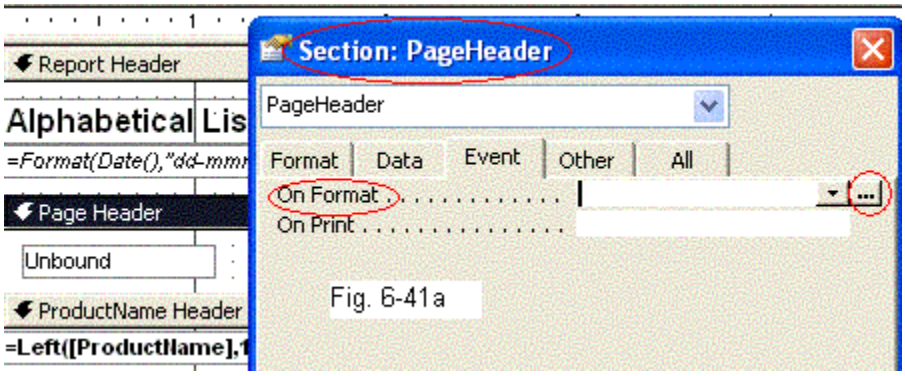
Fig. 6-41

2. Add a second textbox, **txtRange**, in the page footer. Change one of its properties, **Control Source**, to this expression

=[txtFirstOne]& "< ----- >"& [ProductName]

You may want to replace the field **[ProductName]** with one you want to show in the data-range indicator. In this case, the page will display the first and last product name of the page.

3. In order to transfer the first product name into **txtFirstOne**, you must create a procedure with the event **OnFormat** of the **PageHeader** section, as that is where **txtFirstOne** is located (Fig. 6-41a).



Click to open the **Code Builder** and enter following code:

```
Private Sub PageHeader_Format(Cancel As Integer, FormatCount As Integer)
    Me.txtFirstOne = Me.ProductName
End Sub
```

When Access opens the report in print preview, it will display the record from the first one to the last on each page. The event **OnFormat** will pass the first product name to the textbox, **txtFirstOne**, which stays in the box as the procedure is executed. On **Page Footer**, where the textbox **txtRange** is located, select the **Control Source** from its properties and open the **Expression Builder** to build this expression
=[txtFirstOne]& "←------→" & **[ProductName]**

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

4. Save the report and test it. The report is shown in Fig. 6-42.

A

Product Name:	Category Name:
Aniseed Syrup	Condiments

B

Product Name:	Category Name:
Boston Crab Meat	Seafood

C

F

Product Name:	Category Name:	(
Filo Mix	Grains/Cereals	1
Fløtemysost	Dairy Products	1

Fig. 6-42

Product Range **Aniseed Syrup < ---- > Fløtemysost**

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.10 Clean test data after complete the construction

table of contents

During the project design, you may use some "trial" data to assist you in coding and building the objects. Now, the time has come when you will need to wrap up the project and prepare it for shipping. However, you can not keep the test data in the database application when you ship it. It is critical to find an efficient way to remove the trial data without jeopardizing the referential-integrity issue.

Instead of cleaning up the data by hand (selecting and deleting the records), you can write a routine procedure to remove only the test data in the correct order (i.e. first tables of the "many-side" of **one-to-many** relationships before the "one-side" tables). Fig. 6-43 shows the relationship diagram of the example tested in this section.

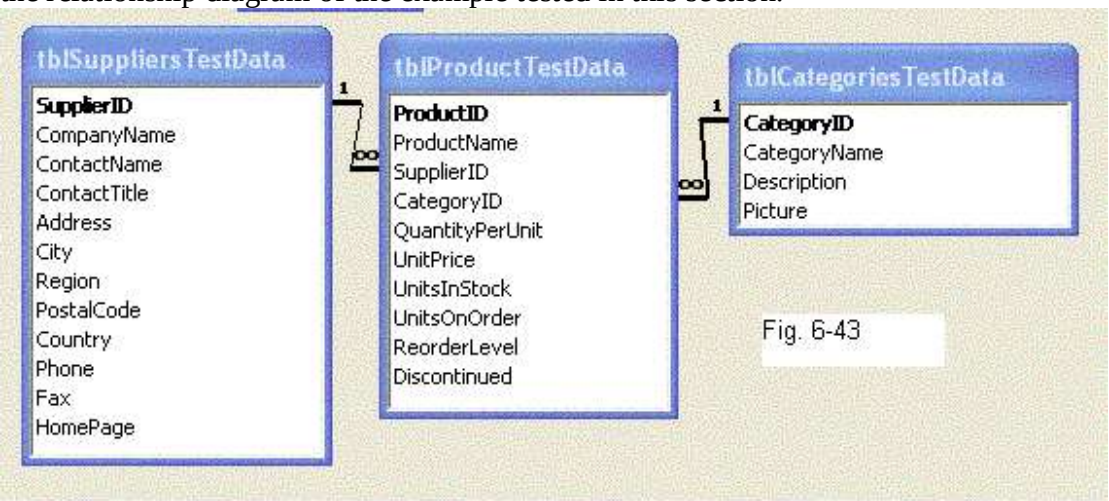


Fig. 6-43

1. Make a new table to specify the order of deletion of each involved table. The structure of this table, **tblTestDataDeleteOrder**, is shown in Fig. 6-44a.

Field Name	Data Type	Description
DelOrder	Number	The order to be deleted for test data.
TableName	Text	The name list of tables which test data need to be deleted.

Fig. 6-44a

The records are in Fig. 6-44b.

tblTestDataDeleteOrder : Table

DelOrder	TableName
1	tblProductTestData
2	tblSuppliersTestData
3	tblCategoriesTestData
0	

Record: 2 of 3

Fig. 6-44b

The smaller number will be deleted first. Tables you don't want to clear should not be entered here.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

2. Create a new module, a public function **DeleteTestData( )**, to conduct this major task. The code is as follows:

```
Public Function DeleteTestData() As Boolean
    'It removes all test data on those tables specified in the tblTestDataDeleteOrder
    'to avoid the referential-integrity problems when deleted by hand.
On Error GoTo ErrHandler
Dim db As DAO.Database
Dim rst As DAO.Recordset
    'Specify where to look for the tables to delete
Set db = CurrentDb()
Set rst = db.OpenRecordset("tblTestDataDeleteOrder", dbOpenSnapshot)
    'Usq SQL statement to delete all records from the table specified in
    'the tblTestDataDeleteOrder until the last table list in the tblTestDataDeleteOrder
Do Until rst.EOF
    db.Execute "delete * from " & rst("TableName")
    rst.MoveNext
Loop
    rst.Close
    Set rst = Nothing
    DeleteTestData = True

ExitHere:
    Exit Function

ErrHandler:
    DeleteTestData = False
    MsgBox "Error " & Err & ": " & Err.Description, , "DeleteTestData()"
    Resume ExitHere
End Function
```

3. Another function, **RestoreBackup( )**, is provided so you can test the "delete trial data" function more than once if necessary. The **RestoreBackup( )** will restore the test data for all three tables: **tblProductTestData**, **tblSuppliersTestData**, and **tblCategoriesTestData**. There are three append queries you will need to create before this function will work: **qryAppSuppliers, qryAppCategories**, and **qryAppProduct**. When you restore the test data, you will append the table on the one-side before the one on the many-side. For example, the many-side table, **qryAppProduct**, will be appended only after the one-side tables, **qryAppSuppliers** and **qryAppCategories**, have been appended. The sample source codes are as follows:

```
Public Function RestoreBackup() As Boolean

Dim db As DAO.Database
Dim qrydf As DAO.QueryDef
On Error GoTo ErrHandle

Set db = CurrentDb()
    'Run the append query qryAppSuppliers to restore records to tblSuppliersTestData
    Set qrydf = db.QueryDefs("qryAppSuppliers")
    qrydf.Execute
    'Run the append query qryAppCategories to restore records to tblCategoriesTestData
    Set qrydf = db.QueryDefs("qryAppCategories")
    qrydf.Execute
    'Run the append query qryAppProduct to restore records to tblProductTestData
    Set qrydf = db.QueryDefs("qryAppProduct")
    qrydf.Execute
    Set qrydf = Nothing
    RestoreBackup = True

ExitHere:
    Exit Function

ErrHandle:
    RestoreBackup = False
    MsgBox "Error " & Err & ": " & Err.Description, , "DeleteTestData()"
    Resume ExitHere
End Function
```

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

4. In order to work on these two functions professionally and efficiently, you will want to make a menu to access them. A form, **frmDeleteTestDataMenu**, needs to be created, with two command buttons to connect these functions using the **Call** statement. The codes behind these two command buttons are:

```
Private Sub cmdDeleteTestData_Click()
    Call DeleteTestData
End Sub

Private Sub cmdRestoreTestData_Click()
    Call RestoreBackup
End Sub
```

The menu is shown as Fig. 6-45.

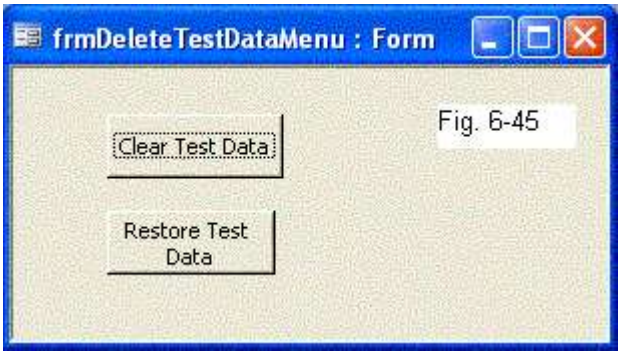


Fig. 6-45

5. Save all modules and forms. Test it by opening the **Delete Test Data** menu.
You may want to compact the database application before you ship it, as Access resets the counter values for all incrementing **AutoNumber** fields (i.e. it will restore the **AutoNumber** to 1 after you delete all test data and compact it).

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.11 How do you control the enabling/disabling of the Shift key? table of contents

When opening the database, it is very common to use the **Shift** key to bypass the **Startup** property or an **AutoExec** macro. In order to secure their database, for most Access developers, it has become essential to disable this bypass key. To disable or enable the **Shift** key, you can write VBA codes or rely on a series of keystrokes of an **AutoKeys** macro.

You will work on two Help documents: **Access Help** and **VBA Help**. Although the code samples found in these Helps may seem too complex, you don't need to worry as long as you understand the key areas. Let's copy the code samples from **VBA Help** into your module.

1. Click the **New** button to create a standard module in the **Module** objects area. Click **Help** on the menu bar when a new Code window is displayed, then click **Microsoft Visual Basic Help** (see Fig. 6-46).

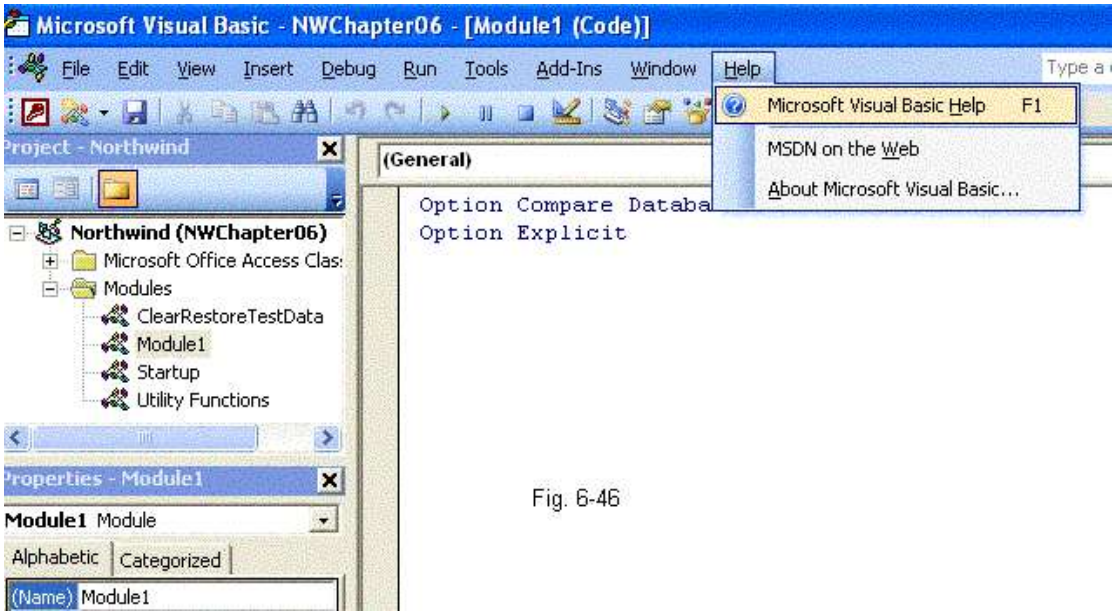


Fig. 6-46

If you did not install the **Help** feature when you installed Microsoft Office, you may be asked to insert an Office 2003 CD (or other version like Office 2000 or XP). You need to have the CD in order to install the **Help**.

2. Type **Disable Shift Key** in the **Search** box as shown in Fig. 6-47. Click the .

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

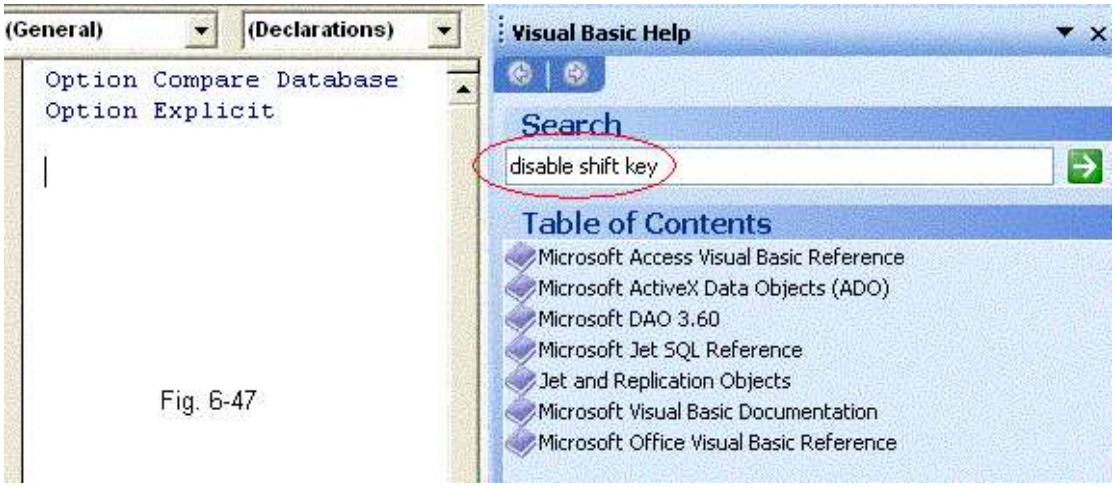


Fig. 6-47

Select the topic **AllowBypassKey Property** in the search results window.

3. The **Help** information is displayed in Fig. 6-48. You may want to resize or reposition the **Help** topic window so you can see as much content as possible.

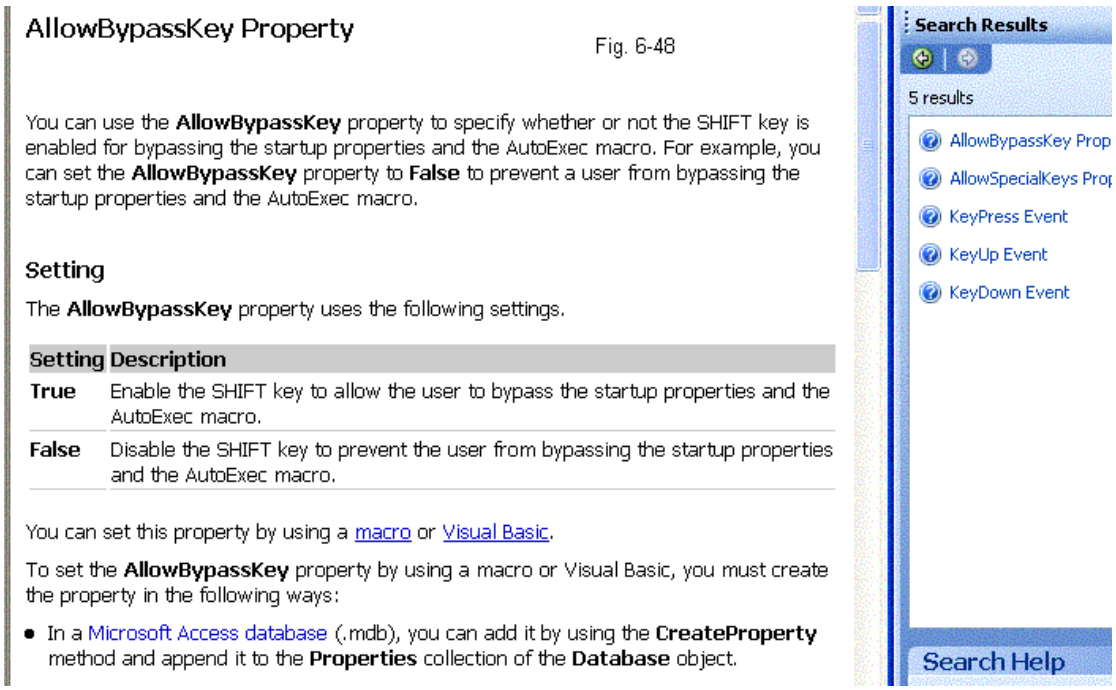


Fig. 6-48

4. Scroll down until you see the Example section, where you can find a code sample for the **SetBypassProperty**. Highlight the entire code section, which is in Fig. 6-49. Copy the code. Make sure you include the **Sub SetBypassProperty( )** line to the **End Function** line. Close the **Help** window and the **Search** results window. Paste the code into the new module. Save the module under the name of **ByPassFunction**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Fig. 6-49

```
Sub SetBypassProperty()
Const DB_Boolean As Long = 1
    ChangeProperty "AllowBypassKey", DB_Boolean, False
End Sub

Function ChangeProperty(strPropName As String, varPropType As Variant, varPropValue As Variar
    Dim dbs As Object, prp As Variant
    Const conPropNotFoundError = 3270

    Set dbs = CurrentDb
    On Error GoTo Change_Err
    dbs.Properties(strPropName) = varPropValue
    ChangeProperty = True

Change_Bye:
    Exit Function

Change_Err:
    If Err = conPropNotFoundError Then     ' Property not found.
        Set prp = dbs.CreateProperty(strPropName, _
            varPropType, varPropValue)
        dbs.Properties.Append prp
        Resume Next
    Else
        ' Unknown error.
        ChangeProperty = False
        Resume Change_Bye
    End If
End Function
```

5. The **ByPassFunction** module has two procedures. The first one is the Sub procedure **SetBypassProperty( )** and the second is the **Function ChangeProperty()**.

6. Change the **Sub** to **Function** on the first procedure. The **End Sub** will automatically change to **End Function**.
7. Create a Boolean argument, **bleValue**, in **Function SetBypassProperty**. This argument value, True or False, will enable or disable the Shift key, depending on how the keystrokes in the macro are chosen. Here is the procedure to add this argument:
Click inside the ( ) to add **bleValue as** and press the spacebar. You will then see the syntax list banner, like in Fig. 6-50.

Fig. 6-50

```
Option Explicit
Function SetBypassProperty(bleValue as b)
Const DB_Boolean As Long = 1
    ChangeProperty "AllowBypassKey",
End Function

Function ChangeProperty(strPropName
    Dim dbs As Object, prp As Variant
    Const conPropNotFoundError = 3270
```

Balloon
BalloonCheckbox
BalloonCheckboxes
BalloonLabel
BalloonLabels
Boolean
BoundObjectFrame

Type **b** and the syntax list will show items that begin with the letter **B**. To insert the **Boolean**, double-click it or press the spacebar after selecting it. Highlight the **False** in the line

ChangeProperty "AllowBypassKey", DB_Boolean, False

and type **bleValue**. The function should look the same as Fig. 6-50a.

Fig. 6-50a

```
Function SetBypassProperty(bleValue As Boolean)
Const DB_Boolean As Long = 1
    ChangeProperty "AllowBypassKey", DB_Boolean, bleValue
End Function
```

8. Save the module. Click **Debug** on the menu, then **Compile Northwind**. Close the Visual Basic window if there is no compile error.

2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6

Next is the Access **AutoKeys** macro group. You can create a macro group that contains custom commands for certain keystroke combinations. Name this macro group as the special name **AutoKeys**. Access will search for an **AutoKeys** macro as you do a keystrokes combination. Then Access runs the macro that is assigned to that key combination. You can create entirely new combinations that replace the default key assignments. However, not all key combinations are available for the **AutoKeys** macro assignment. You can get a list of these keys from **Access Help** (as shown in Fig. 6-50b and Fig. 6-50c) by clicking **Assign an action or set of actions to a key topic**.

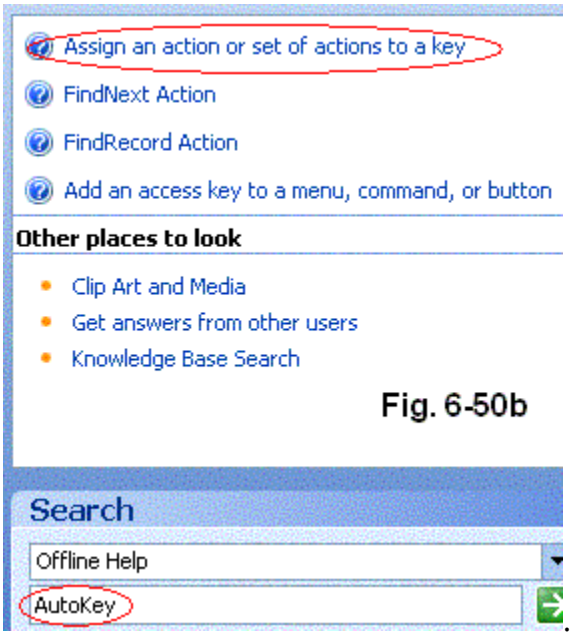


Fig. 6-50b

Click **Syntax for AutoKeys key combinations** to see the list of keys.

SendKeys syntax	Key combination
^A or ^4	CTRL+A or CTRL+4
{F1}	F1
^{F1}	CTRL+F1
+{F1}	SHIFT+F1
{INSERT}	INS
^{INSERT}	CTRL+INS
+{INSERT}	SHIFT+INS
{DELETE} or {DEL}	DEL
^{DELETE} or ^{DEL}	CTRL+DEL
+{DELETE} or +{DEL}	SHIFT+DEL

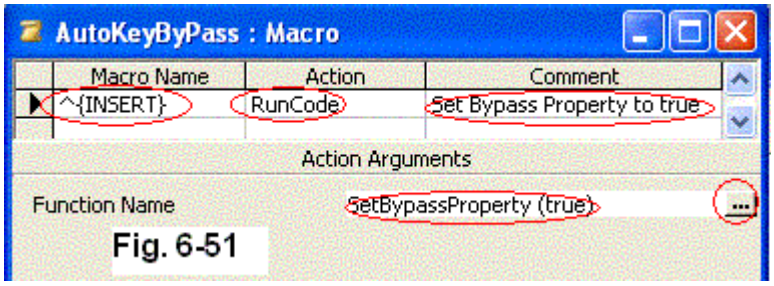
Fig. 6-50c

1. Create a macro to run the **SetBypassProperty** with the Boolean value **True**. This will enable the bypass key. For instance, the key combination for this macro is **Ctrl+Ins**.

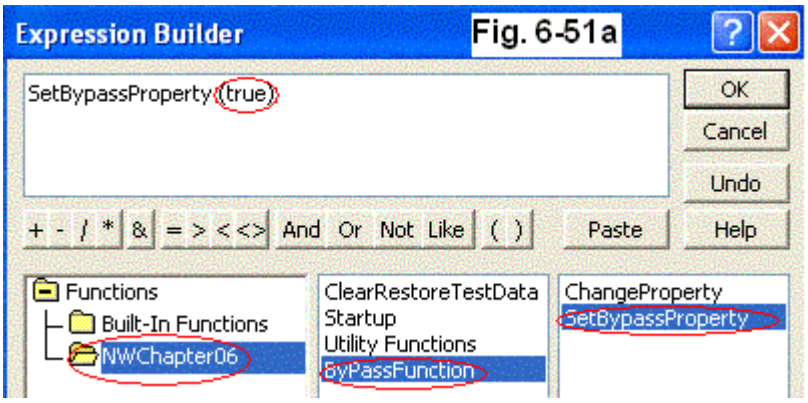
2. In the first row under the **Macro Name**, enter **^{INSERT}.** Move the cursor to the **Action** column. Click the **Action** list box and select **RunCode**. Press the tab key to move to the **Comment** column and enter **Set Bypass Property to true**. Click inside the action argument **Function Name** box; you will see like in Fig. 6-51.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

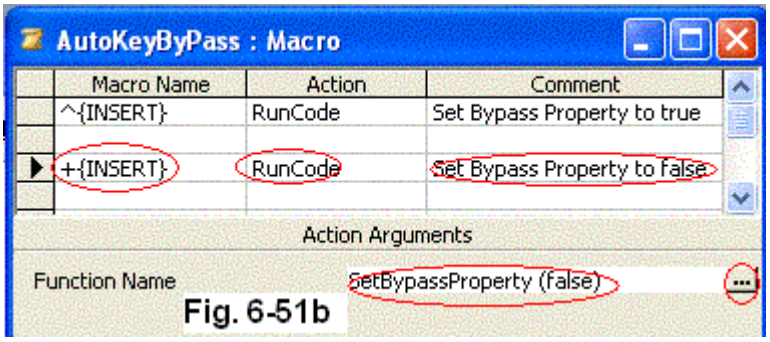
ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.



Then click the **Expression Builder** to insert the function you created, as demonstrated in Fig. 6-51a. Next, double-click the **Functions** folder and **NWDish06** folder. Click **ByPassFunction**, and then double-click **SetBypassProperty** or click **SetBypassProperty** and then the **Paste** button. Change the argument **<bValue>** to **true** inside the **SetBypassProperty( ).**



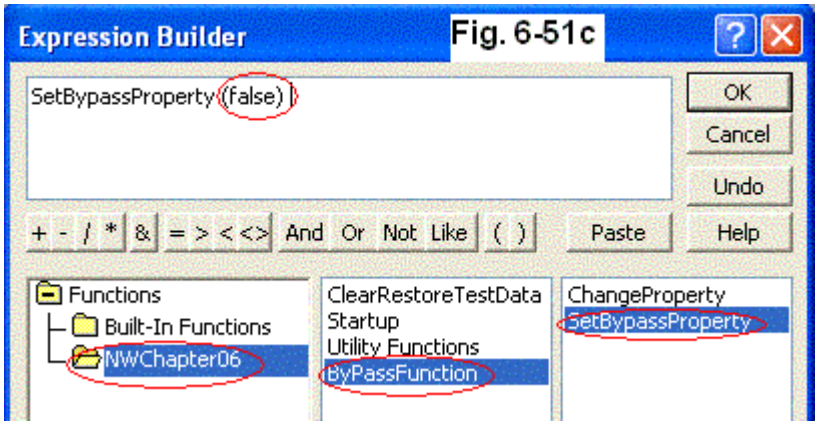
3. Save the macro as **AutoKeyByPass**.
4. To complete the macro, you will enter **+{INSERT}** in the **Macro Name** box to disable the bypass key. Enter **Set Bypass Property to false** into the **Comment** column (see Fig. 6-51b).



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

5. Go to **Expression Builder** to choose the function **SetBypassProperty**, as shown in Fig. 6-51c. Don't forget to replace the argument **<bValue>** with **false**.

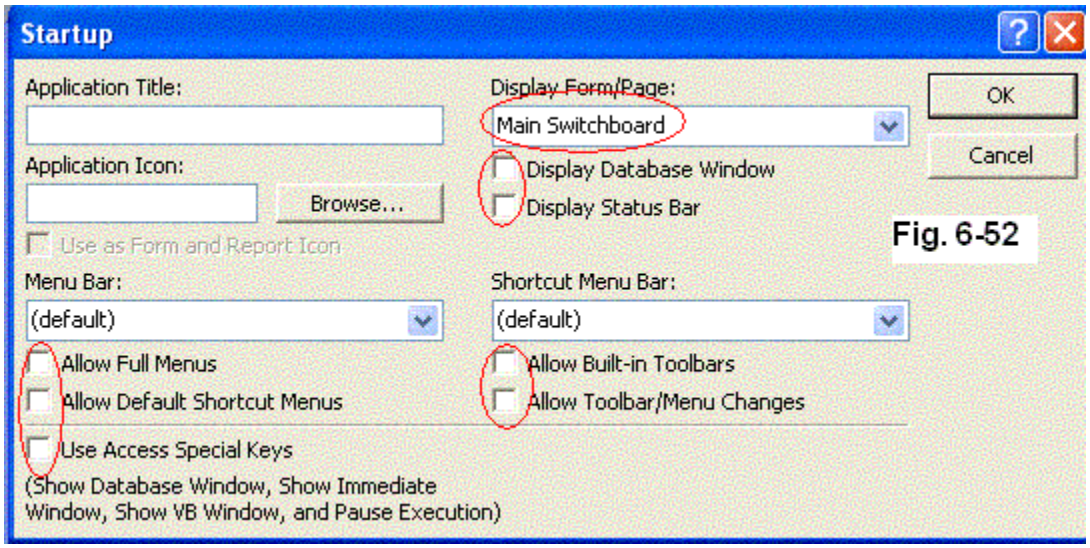


The key combination to disable the bypass key is **Shift+Ins**.
6. Save macro.

Lunch time: Test it

Before you test it, you may want to make a backup copy of your database.

1. Change the **Startup** property settings by opening **Tools** from the menu, then click **Startup** to open the startup dialog window. Select the **Main Switchboard** in the **Display Form/Page** box and uncheck all check boxes (Fig. 6-52). Click OK to close the **Startup** dialog window.



2. Press the **Shift+Ins** keys once to disable the bypass key, and then close the **NWDish06** database and exit Access.

3. Reopen Access, and hold down the **Shift** key when you open the **NWDish06** database. The bypass key should not work, and you should see the **Main Switchboard**.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

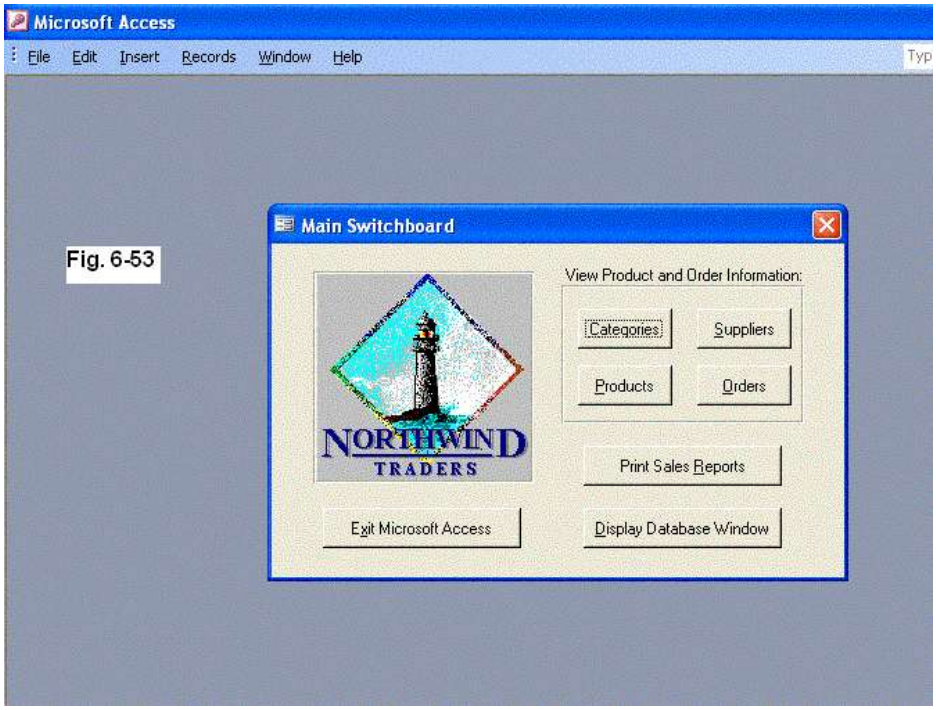


Fig. 6-53

4. Press **Ctrl+Ins** to enable the bypass key, and then click **Exit Microsoft Access** on the **Main Switchboard**, as given in Fig. 6-53.

5. Open Access again, and hold down the **Shift** key as you open the **NWDish06** database. You should then bypass the startup property settings. Now the objects window should appear.

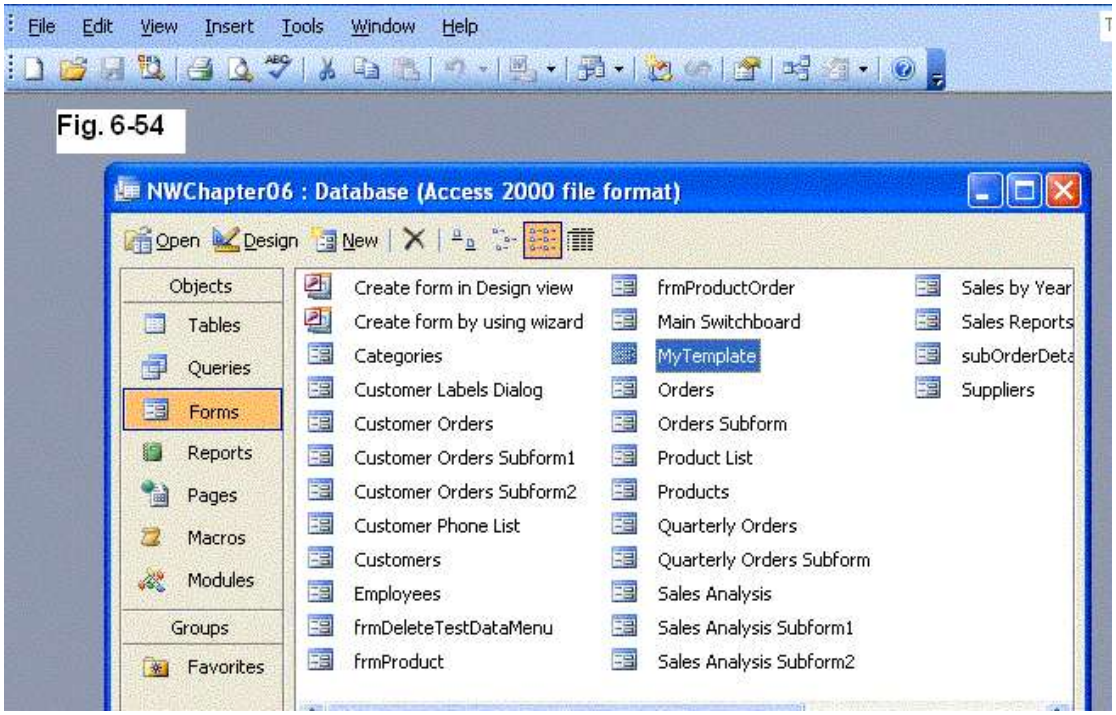


Fig. 6-54

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.12 How to convert the replica back to a regular database table of contents

Here are Microsoft's suggested steps on how to convert a replica into a regular database:

1. Create a new database.
2. Import all the objects, except for tables, from the replicated database into the newly created database.
3. Close the new database, and then open the replicated database.
4. In the Database window, click **Queries** under **Objects**, and then click **New**.
5. In the **New Query** dialog box, click **Design View**, and then click OK.
6. Select the first table in the **Show Table** dialog box. Drag all the fields to the **Field** row, except for those that were generated for replication (Gen_*OLE Fieldnames*, s_ColLineage, s_Generation, s_Guid, s_Lineage). If your application requires one of these fields that was created by replication, then go ahead and add it to the query.
7. On the **Query** menu, click **Make Table**. In the **Table Name** box, select the current table name. Click **Another Database**, and add the name of the database that you created in step 1. Run the query.
8. Repeat steps 4 through 7 for each table in the replicated database.
9. Close the replicated database, and then open the new database. Re-create all indexes and relationships that existed in the replicated database.
10. After you've completed steps 1 through 9, compact and then repair the new database. The new database should be approximately the same size as the replication but should not contain any properties of the replica.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

Flavor 6.13 Reasons why you should consider splitting your database table of contents

The following are typical reasons to split a database:

Your database needs to be used by multiple users on a network rather than a stand alone PC.

You need to continually develop the database even after the database is put to work and you do not have Microsoft Visual Source Safe installed.

You do not allow your users to make design changes to tables.

If you split the database, each user has their own copy of the forms, queries, macros, modules, and reports at client workstations. Therefore, the only data that must be sent across the network are the data in the tables. Any new improvement on any objects, except tables, can be easily updated at each workstation by replacing the existing one with the newest version. All modifications can be done without interrupting the regular business because you will not touch the tables of the server.

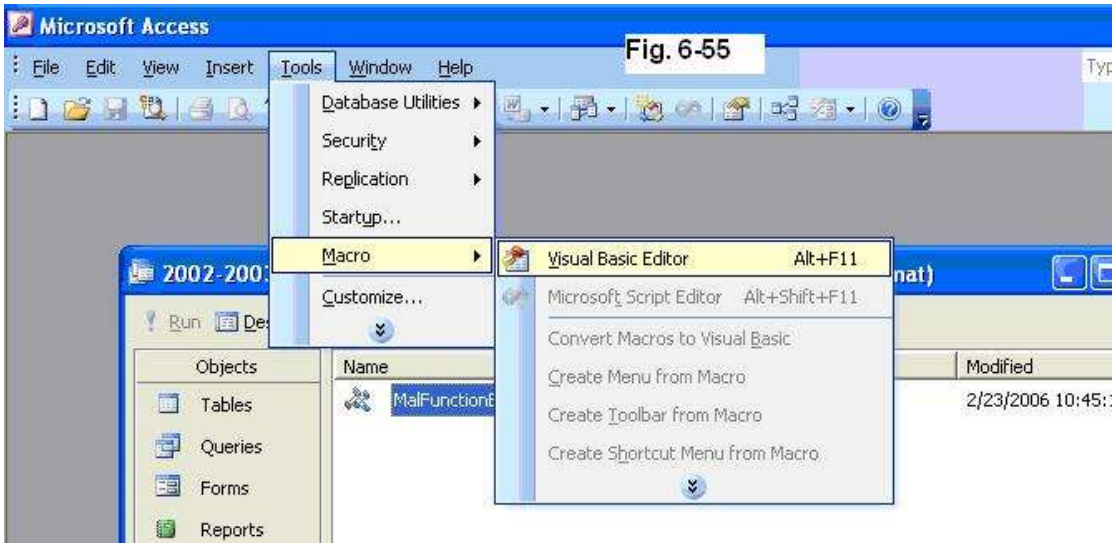
2006-2009 © All Rights Reserved.

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

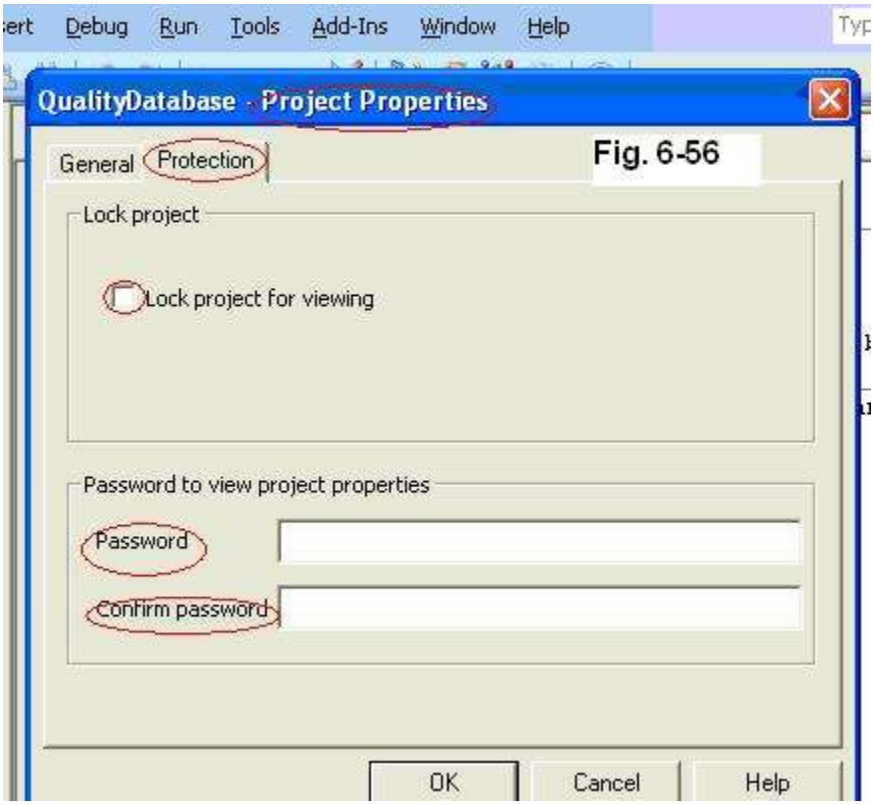
Flavor 6.14 Remove the security on your database table of contents

This procedure assumes your database is split. The database with all tables saved on the server will be called a **backend**. The database with all objects, except tables, installed on client's workstation of a network environment will be called a **frontend**.

1. Open the **frontend** database in **Design View**.
2. Click **Tools** from menu, then **Macro**, **Visual Basic Editor** as shown in Fig. 6-55.



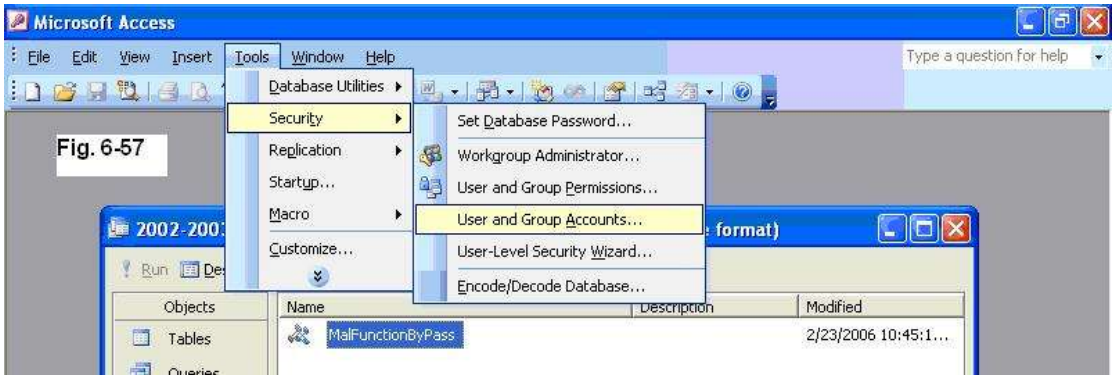
3. In **Visual Basic Editor**, from menu, click **Tools**, **Quality Database properties**, the **Protection** tab, uncheck the box **Lock project for viewing**, and remove the password (refer to Fig. 6-56). Click OK to close the **Visual Basic Editor**.



2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

4. Click **Tools** from the menu, then **Security**, **User and Group Accounts** as illustrated in Fig. 6-57.



In the **Users** tab, click the **Name** box and pick the person you want to delete by selecting the name and clicking **Delete** (Fig. 6-58). Delete all of them except yourself and Admin, just in case you need to login.

Another method to get rid of security settings:
1. Create a blank database and use the Access **System.mdw** group file.
2. Close the above database. Open the existing database with the security settings. Make sure you change the group to the security mdw file.
3. Export all objects one by one from the existing secured database to the new blank database.
4. Change the security to the default **System.mdw**, and open the new blank database (it's no longer blank after you import all objects from the existing database).
5. Double-check all designs in the new database.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.

References

table of contents

www.microsoft.com
Access Help
Access VBA Help
Balter, "Mastering Access 2002 Desktop Development", Sams
Brown, Petrusha, "Access VBA Programming", Osborne
Callahan, "Microsoft Access/Visual Basic Step by step", Microsoft Press
Hentzen, "Access 2000 Programming from the ground up", Osborne
Taylor, Andersen, "Access 2003 Power Programming with VBA", Wiley
Vine, "Access VBA Programming", Premier Press

About the Author

table of contents

Dejang Liu has been a Microsoft Access application developer since Access 2.0 was released in late 80'. In 2000, he became a faculty member of the Computer Information Systems Department at the College of DuPage in Glen Ellyn, Illinois. Before becoming a full-time professor at College of DuPage, Dejang taught as an adjunct at Morton College of Cicero and Joliet Junior College of Joliet from 1996 to 2000, while serving at the East-West University of Chicago as Department Chair of Computer and Information Science Department from 1998 to 2000.

After graduating from the University of Minnesota in 1989 with a master's degree in mechanical engineering, Mr. Liu worked as a consultant, engineer, director, and programmer at various private companies. In 1991, he became a partner of Diamond Software Services, a company that specializes in developing custom database applications. Dejang was invited to work with Diamond InfoTech in Taiwan to localize the Chinese version of Office 95 for the Microsoft Corporation in Taiwan.

Dejang holds a BS in Aeronautical Engineering from Tamkang University, Taiwan and an MBA in MIS from the University of Illinois in Chicago. Currently working on and expecting to receive his doctoral degree in Adult Higher Education from Northern Illinois University in DeKalb in 2009, he was certified as a Microsoft Office Specialist and a Microsoft Authorized Office Specialist Instructor since the release of Office 97.

He is am amateur photographer and golfer. He believes that the summer is a season of "Three R's": rest, recharge, and research. So his summer is filled with activities.

2006-2009 © All Rights Reserved.
Fu Lin Trade Association

ISBN 0-9787733-0-6
Ingredients for Database Recipe. 2nd edition.