

**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**LPG DISTRIBUTOR STOCK & SALE SYSTEM**

**Graduation Project**

**COM 400**

**Student : Ahmet ÖZBEK**

**Supervisor : Assoc. Prof. Rahib ABİYEV**

**Nicosia – 2008**

## ACKNOWLEDGEMENTS

*"Firstly, I would like to thank to my supervisor Mr Rahib ABIYEV, Elbrus IMANOV, Dr.Kaan UYAR, Okan DONANGİL, Ümit İLHAN for their great advise and recommendation for finishing my project properly also, teaching and guiding me in others lectures.*

*I am greatly indepted to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my brother Mahmut ÖZBEK and my uncle Ahmet Şerif ÇULCU did for me during my educational life, also I want to say thanks to my mother Saliha ÖZBEK. I dedicate my project to them.*

*I thank all the staff of the faculty of engineering for giving facilities to practise, teaching and solving problem in my complete undergraduation program*

*I thank my friends Burak MELEK and Kerim ALTANHAN for their help, they get tired with me, and they helped me and give morale everytime.*

*I thank them with my all*

*Finally, I promise to do my best in my life as an bachelor of engineer after finishing my undergraduate program"*

# **TABLE OF CONTENTS**

<b>ACKNOWLEDGEMENT</b>	i
<b>TABLE OF CONTENTS</b>	ii
<b>ABSTRACT</b>	iii
<b>INTRODUCTION</b>	1
 <b>CHAPTER ONE – MICROSOFT ACCESS DATABASE</b>	 2
1.1 Microsoft Access Database Fundamentals	2
1.2 Microsoft Access Reports Tutorial	5
1.3 Creating a Simple Query in Microsoft Access	11
1.4 Creating Forms in Microsoft Access	16
1.5 How do I encrypt an Access 2007 database?	18
<b>CHAPTER ONE – DELPHI PROGRAMMING LANGUAGE</b>	19
2.1 A brief history of Borland's Delphi	19
2.2 Delphi For Beginners:	21
2.3 A Glossary of Delphi Programming Technical Terms	23
2.4 Understanding Delphi Project Files (.DPR)	37
2.4.1 New: Delphi Project	37
2.4.2 Project File	37
2.4.3 Project Unit	38
2.4.4 An Example: Hide Main Form / Hide Taskbar Button	39
2.5 Understanding the Birth, Life and Death of a Delphi Form	40
2.6 Understanding and Using Functions and Procedures	43
2.7 Understanding and Using Loops	50
2.8 Understanding Typed Constants in Delphi	54
2.9 Running Delphi Applications With Parameters	57
<b>CHAPTER THREE – DEVELOPMENT OF LPG DISTRIBUTOR STOCK &amp; SALE SYSTEM</b>	60
<b>CONCLUSION</b>	69
<b>REFERENCES</b>	70
<b>APPENDIX</b>	71

## **ABSTRACT**

The aim of this project is to register petrol station program that contain registration, all applications and also customers, sales, stocks and lpg application. The program was prepared by using Delphi programming and using MS Access database.

This project consist of so many forms and menus. The main form of the arrive the others forms . Which are include information about the sales,stocks and customers.

I think to give this system to GAPGAZ A.Ş.

To show results, show the efficiency of the program of LPG sale and stock in program of the using in other chapters.



## INTRODUCTION

Information technology (IT), as defined by the Information Technology Association of America (ITAA), is "the study, design, development, implementation, support or management of computer-based information systems, particularly software applications and computer hardware." IT deals with the use of electronic computers and computer software to convert, store, protect, process, transmit and retrieve information, securely.

Recently it has become popular to broaden the term to explicitly include the field of electronic communication so that people tend to use the abbreviation ICT (Information and Communications Technology), it is common for this to be referred to as IT & T in the Australasia region, standing for Information Technology and *Telecommunications*.

Today, the term information technology has ballooned to encompass many aspects of computing and technology, and the term is more recognizable than ever before. The information technology umbrella can be quite large, covering many fields. IT professionals perform a variety of duties that range from installing applications to designing complex computer networks and information databases. A few of the duties that IT professionals perform may include data management, networking, engineering computer hardware, database and software design, as well as the management and administration of entire systems.

The aim of this project is to develop a simple Stock Management System for small companies. The project consists of introduction, three chapters and conclusion.

Chapter One; describes general terms of Microsoft Access Database and the processes of creating a database.

Chapter Two; describes the main lines of Borland Delphi Programming Language such as reserved words, simple codes, methods and basic events.

Chapter Three; is the User's Manual of the program that gives information about the system developed as Stock Management System.

## **Microsoft Access Database Fundamentals**

Are you overwhelmed by the large quantities of data that need to be tracked in your organization? Perhaps you're currently using a paper filing system, text documents or a spreadsheet to keep track of your critical information. If you're searching for a more flexible data management system, a database might be just the salvation you're looking for.

What is a database? Quite simply, it's an organized collection of data. A database management system (DBMS) such as Access, FileMaker Pro, Oracle or SQL Server provides you with the software tools you need to organize that data in a flexible manner. It includes facilities to add, modify or delete data from the database, ask questions (or queries) about the data stored in the database and produce reports summarizing selected contents.

Microsoft Access provides users with one of the simplest and most flexible DBMS solutions on the market today. Regular users of Microsoft products will enjoy the familiar Windows "look and feel" as well as the tight integration with other Microsoft Office family products. An abundance of wizards lessen the complexity of administrative tasks and the ever-present Microsoft Office Helper (you know... the paper clip!) is available for those who care to use it. Before purchasing Access, be sure that your system meets Microsoft's minimum system requirements. To further our discussion, let's first examine three of the major components of Access that most database users will encounter - tables, queries, forms. Once we've completed that we'll look at the added benefits of reports, web integration and SQL Server integration.

Tables comprise the fundamental building blocks of any database. If you're familiar with spreadsheets, you'll find database tables extremely similar.



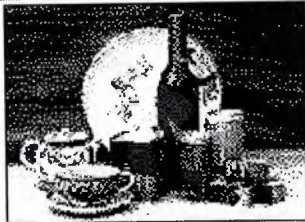
Employees : Table						
	Employee ID	Last Name	First Name	Title	Title Of	Birth Date
*	1	Davolio	Nancy	Sales Representative	Ms.	08-Dec-1968
+	2	Fuller	Andrew	Vice President, Sales	Dr.	19-Feb-1962
+	3	Leverling	Janet	Sales Representative	Ms.	30-Aug-1963
+	4	Peacock	Margaret	Sales Representative	Mrs.	19-Sep-1958
+	5	Buchanan	Steven	Sales Manager	Mr.	04-Mar-1955
+	6	Suyama	Michael	Sales Representative	Mr.	02-Jul-1963
+	7	King	Robert	Sales Representative	Mr.	29-May-1960
+	8	Callahan	Laura	Inside Sales Coordinator	Ms.	09-Jan-1958
+	9	Dodsworth	Anne	Sales Representative	Ms.	02-Jul-1969

The table above contains the employee information for our organization -- characteristics like name, date of birth and title. Examine the construction of the table and you'll find that each column of the table corresponds to a specific employee characteristic (or **attribute** in database terms). Each row corresponds to one particular employee and contains his or her information. That's all there is to it! If it helps, think of each one of these tables as a spreadsheet-style listing of information.

<b>Orders</b>																																
<b>Bill To:</b> Alfreds Futterkiste Obere Str. 57 Berlin 12209 Germany	<b>Ship To:</b> Alfreds Futterkiste Obere Str. 57 Berlin 12209 Germany																															
Salesperson: Suyama, Michael	Ship Via: <input checked="" type="checkbox"/> Speedy <input type="checkbox"/> United <input type="checkbox"/> Federal																															
Order ID: 10643	Order Date: 25-Aug-1997 Required Date: 22-Sep-1997 Shipped Date: 02-Sep-1997																															
<table border="1"> <thead> <tr> <th>Product</th> <th>Unit Price</th> <th>Quantity</th> <th>Discount</th> <th>Extended Price</th> </tr> </thead> <tbody> <tr> <td>Spegesild</td> <td>\$12.00</td> <td>2</td> <td>25%</td> <td>\$18.00</td> </tr> <tr> <td>Chartreuse verte</td> <td>\$18.00</td> <td>21</td> <td>25%</td> <td>\$283.50</td> </tr> <tr> <td>Rössle Sauerkraut</td> <td>\$45.60</td> <td>15</td> <td>25%</td> <td>\$513.00</td> </tr> <tr> <td>*</td> <td></td> <td></td> <td>0%</td> <td></td> </tr> </tbody> </table>	Product	Unit Price	Quantity	Discount	Extended Price	Spegesild	\$12.00	2	25%	\$18.00	Chartreuse verte	\$18.00	21	25%	\$283.50	Rössle Sauerkraut	\$45.60	15	25%	\$513.00	*			0%		<table border="1"> <tr> <td>Subtotal:</td> <td>\$814.50</td> </tr> <tr> <td>Freight</td> <td>\$29.46</td> </tr> <tr> <td>Total:</td> <td>\$843.96</td> </tr> </table>	Subtotal:	\$814.50	Freight	\$29.46	Total:	\$843.96
Product	Unit Price	Quantity	Discount	Extended Price																												
Spegesild	\$12.00	2	25%	\$18.00																												
Chartreuse verte	\$18.00	21	25%	\$283.50																												
Rössle Sauerkraut	\$45.60	15	25%	\$513.00																												
*			0%																													
Subtotal:	\$814.50																															
Freight	\$29.46																															
Total:	\$843.96																															
<div> <div>Display products of the month</div> <div>Print Invoice</div> </div>																																
Record: 1 of 830																																

Reports provide the capability to quickly produce attractively formatted summaries of the data contained in one or more tables and/or queries. Through the use of wizards, database users can create reports in literally a matter of minutes. As an example, let's return to our Northwind database. In this case, suppose that our company wishes to produce a catalog to share our product information with current and prospective clients. In previous sections, we learned that this sort of information

could be retrieved from our database through the judicious use of queries. However, recall that this information was presented in a tabular form -- not exactly the most attractive marketing material! Reports allow the inclusion of graphics, attractive formatting and pagination. Take a look at the sample report in the illustration below:

<div> <b>Beverages</b>  <i>Soft drinks, coffees, teas, beers, and ales</i> </div> <div>  </div>			
<i>Product Name:</i>	<i>Product ID:</i>	<i>Quantity Per Unit:</i>	<i>Unit Price:</i>
Chai	1	10 boxes x 20 bags	\$18.00
Chang	2	24 - 12 oz bottles	\$19.00
Chartreuse verte	39	750 cc per bottle	\$18.00
Côte de Blaye	38	12 - 75 cl bottles	\$263.50
Guaraná Fantástica	24	12 - 355 ml cans	\$4.50
Ipoh Coffee	43	16 - 500 g tins	\$46.00
Lakkaikööri	76	500 ml	\$18.00
Laughing Lumberjack Lager	67	24 - 12 oz bottles	\$14.00

Microsoft Access also provides native support for the World Wide Web. Posting data to the web is a breeze. If you have a formatted report that you would like to share with Internet or Intranet users, you can simply export it to an HTML file and publish it to your organization's web server. For those with more complex tastes, the advanced features of Access 2000 provide interactive data manipulation capabilities to web users.

Finally, no discussion of Microsoft Access is complete without mentioning its capability to tightly integrate with SQL Server, Microsoft's professional database server product. If you're in an organization that utilizes SQL Server, you'll be pleased to learn that you can retrieve, manipulate and work with the data stored on your organization's database server within the Microsoft Access environment. For more on this, view Microsoft's page on SQL Server/Office integration.



## Microsoft Access Reports Tutorial

### Part 1: Getting Started

In our previous tutorials, you've learned a good deal about Microsoft Access. Together, we created a query, modified the query to make it more complex, and created a data entry form. We've learned the skills necessary to put information into a database and selectively remove the exact information we're seeking. In this tutorial, we're going to go a step further and learn how to create professionally formatted reports automatically from our database information. Returning to our familiar Northwind Company, we're going to design a nicely-formatted listing of employee home telephone numbers for the use of management.

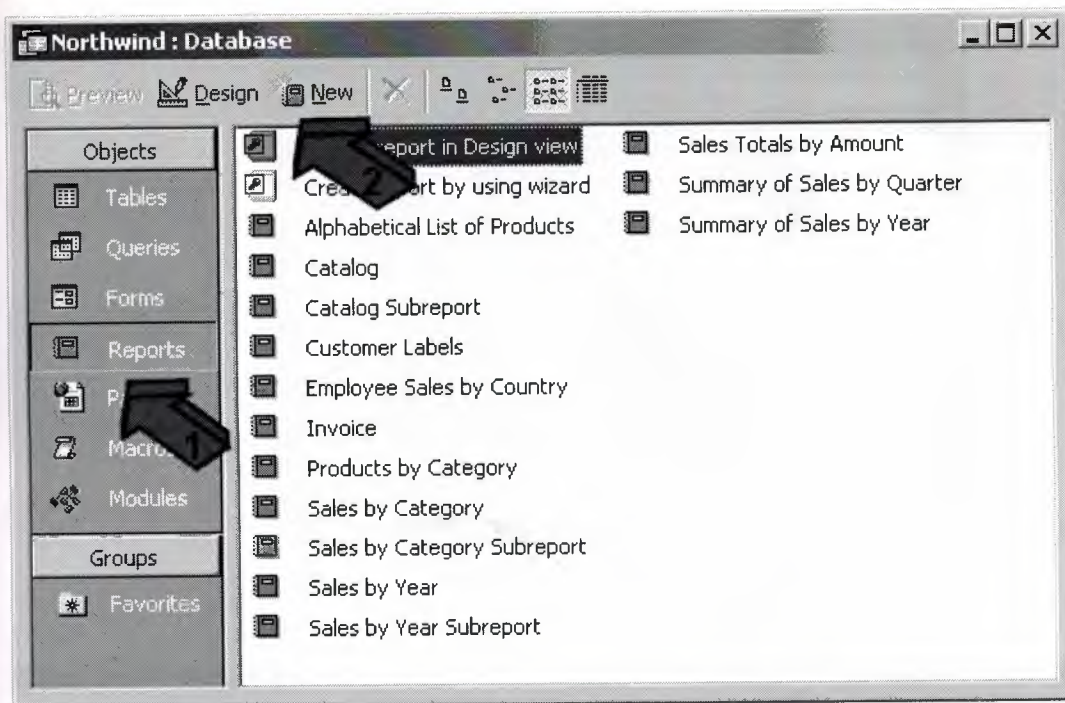
The sample images in this tutorial were created using Access 2000. If you are running an earlier version of Access, your screen images may appear slightly different. However, the same general principles still apply and you should be able to follow along. If you need a quick-start on the basics of Access before getting started, take a look at the article "Microsoft Access Fundamentals."

Once again, we're going to use the Northwind sample database. Before we get started, open up Microsoft Access and then open the Northwind database. If you need help with this step, please read the article "How to Install the Northwind Sample Database."

**1. Choose the Reports menu.** Once you've opened Northwind, you'll be presented with the main database menu shown below. Go ahead and click on the "Reports" selection and you'll see a list of the various reports Microsoft included in the sample database. If you'd like, feel free to double-click on a few of these and get a feel for what reports look like and the various types of information that they contain.

**2. Create a new report.** After you've satisfied your curiosity, go ahead and click on the "New" button and we'll begin the process of creating a report from scratch.

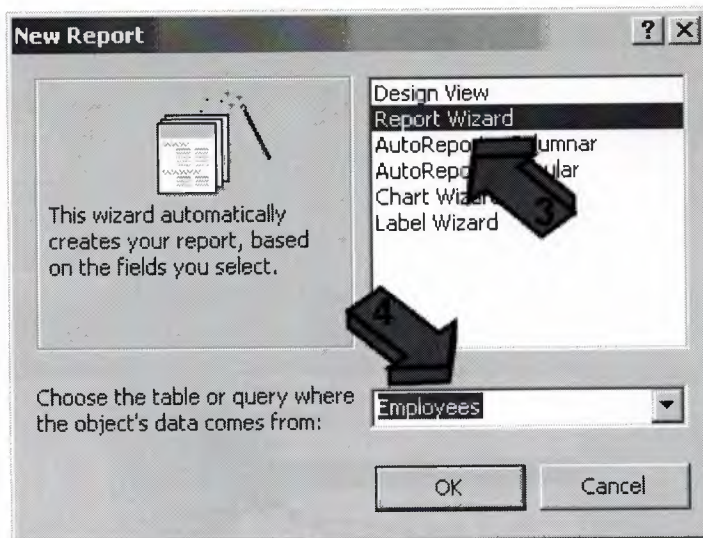




### Create a new report

3. **Select the Report Wizard.** The next screen that appears will ask you to select the method you wish to use to create the report. We're going to use the Report Wizard which will walk us through the creation process step-by-step. After you've mastered the wizard, you might want to return to this step and explore the flexibility provided by the other creation methods.

4. **Choose a table or query.** Before leaving this screen, we want to choose the source of data for our report. If you want to retrieve information from a single table, you can select it from the drop-down box below. Alternatively, for more complex reports, we can choose to base our report on the output of a query that we previously designed. For our example, all of the data we need is contained within the Employees table, so choose this table and click on OK.



### Select a creation method

Next, we'll select exactly which table data to include in the report and learn how to apply formatting to our finished product. Read on!

## Microsoft Access Reports Tutorial Part 2: Selecting the Data

**5. Select the fields to include.** Use the '>' button to move over the desired fields. Note that the order you place the fields in the right column determines the default order they will appear in your report. Remember that we're creating an employee telephone directory for our senior management. Let's keep the information contained in it simple -- the first and last name of each employee, their title and their home telephone number. Go ahead and select these fields. When you are satisfied, click the Next button.

**6. Select the grouping levels.** At this stage, you can select one or more grouping levels to refine the order in which our report data is presented. For example, we may wish to break down our telephone directory by department so that all of the members of each department are listed separately. However, due to the small number of employees in our database, this is not necessary for our report. Go ahead and simply click on the Next button to bypass this step. You may wish to return here later and experiment with grouping levels.



**Report Wizard**

Which fields do you want on your report?  
You can choose from more than one table or query.

Tables/Queries  
Table: Employees

Available Fields:

- BirthDate
- HireDate
- Address
- City
- Region
- PostalCode
- Country
- Extension

Selected Fields:

- FirstName
- LastName
- Title
- HomePhone

Buttons: Cancel, < Back, Next >, Finish

**Select the fields to include**

**Report Wizard**

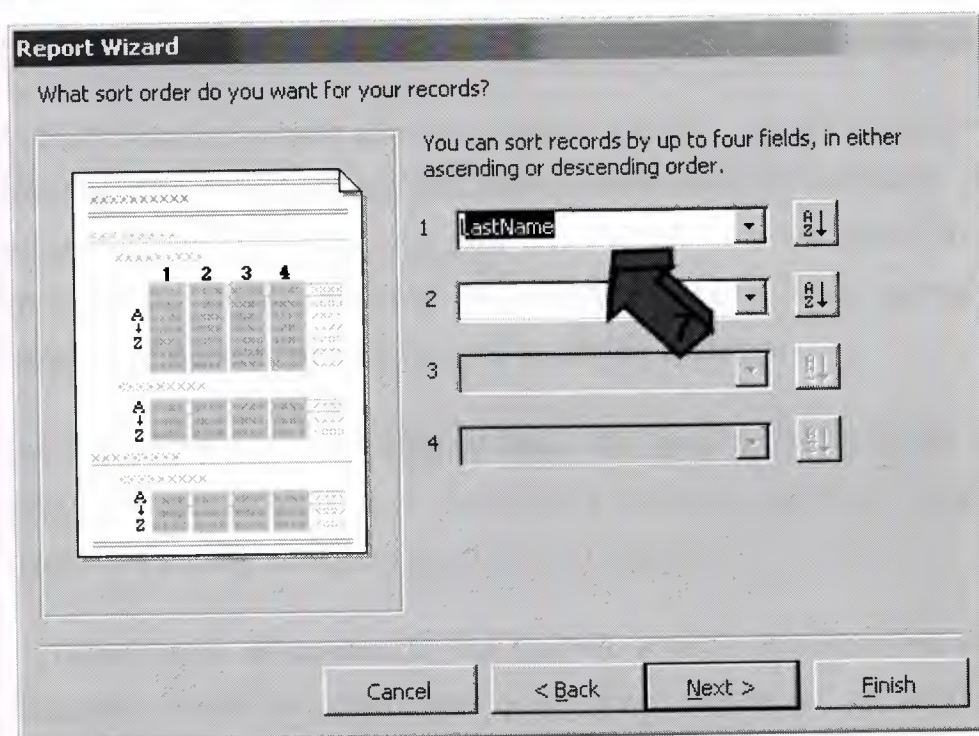
Do you want to add any grouping levels?

Fields: FirstName, LastName, Title, HomePhone

Buttons: Grouping Options..., Cancel, < Back, Next >, Finish

**Choose the grouping levels**

**7. Choose your sorting options.** In order to make reports useful, we often want to sort our results by one or more attributes. In the case of our telephone directory, the logical choice is to sort by the last name of each employee. Select this attribute from the first drop-down box and then click the Next button to continue.



**Choose the sorting options**

## Microsoft Access Reports Tutorial Part 3: Finishing Touches





## Creating a Simple Query in Microsoft Access

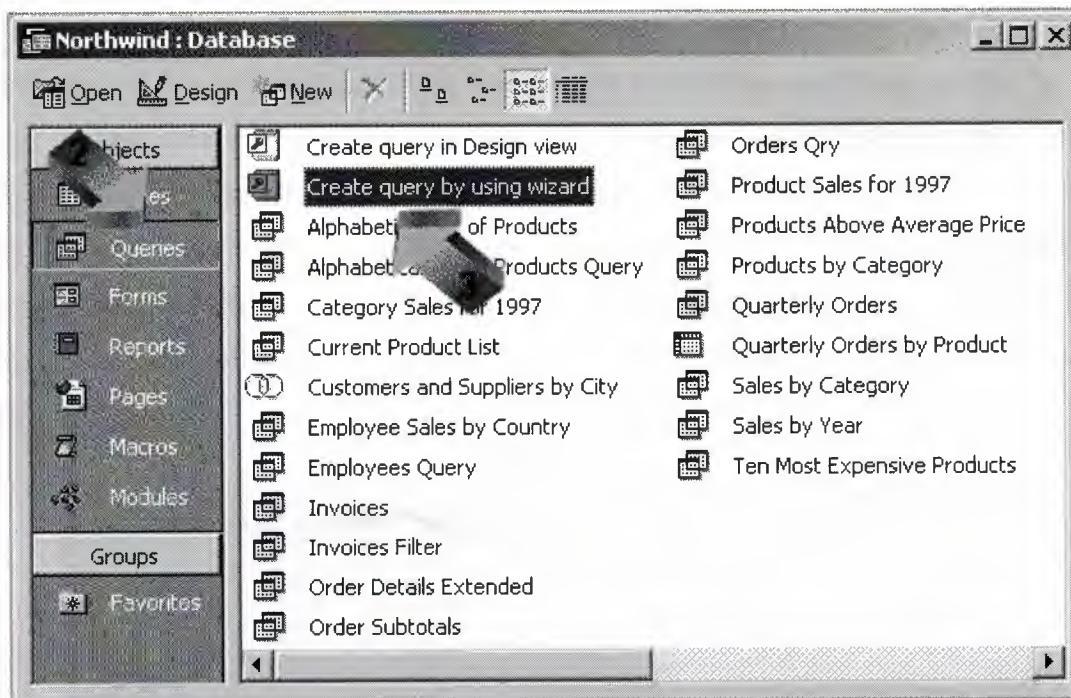
Have you ever wanted to combine information from multiple tables in your database in an efficient manner? Microsoft Access offers a powerful query function with an easy-to-learn interface that makes it a snap to extract exactly the information you need from your database. In this tutorial we'll explore the creation of a simple query.

In this example, as with all of our Access tutorials, we will use Access 2000 and the Northwind sample database included on the installation CD-ROM. If you're using an earlier version of Access, you may find that some of the menu choices and wizard screens are slightly different. However, the same basic principles apply to all versions of Access (as well as most database systems).

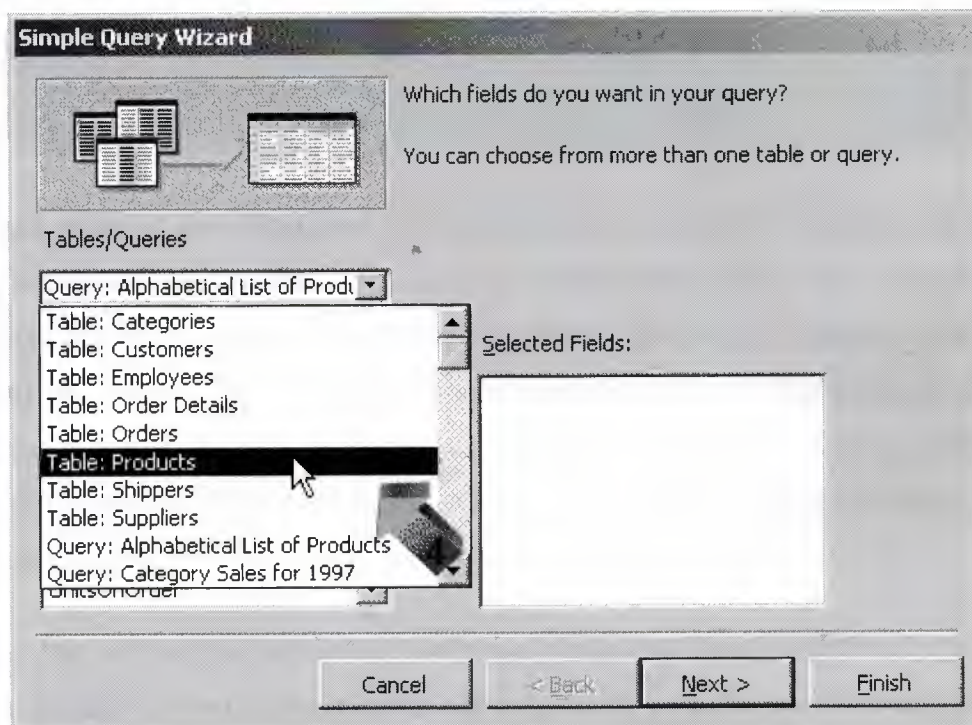
Let's explore the process step-by-step. Our goal in this tutorial is to create a query listing the names of all of our company's products, current inventory levels and the name and phone number of each product's supplier.

1. **Open your database.** If you haven't already installed the Northwind sample database, these instructions will assist you. Otherwise, go to the File tab, select Open and locate the Northwind database on your computer.
2. **Select the queries tab.** This will bring up a listing of the existing queries that Microsoft included in the sample database along with two options to create new queries.
3. **Double-click on "create query by using wizard".** The query wizard simplifies the creation of new queries. We'll use it in this tutorial to introduce the concept of query creation. In later tutorials we'll examine the Design view which facilitates the creation of more sophisticated queries.

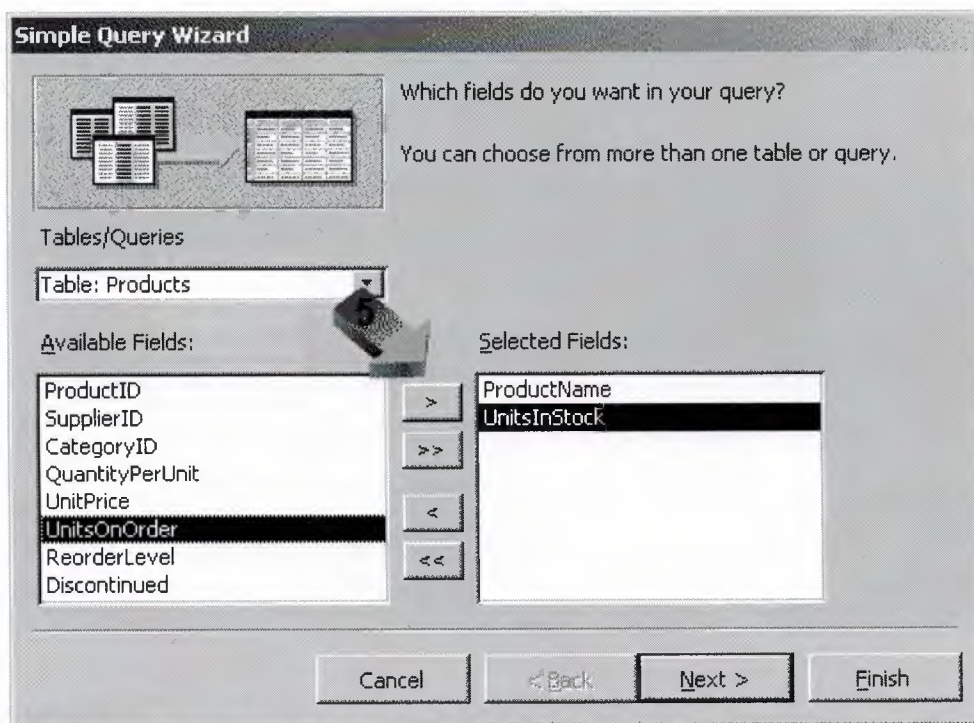




4. **Select the appropriate table from the pull-down menu.** When you select the pull-down menu, you'll be presented with a listing of all the tables and queries currently stored in your Access database. These are the valid data sources for your new query. In this example, we want to first select the Products table which contains information about the products we keep in our inventory.



5. **Choose the fields you wish to appear in the query results.** by either double-clicking on them or by single clicking first on the field name and then on the ">" icon. As you do this, the fields will move from the Available Fields listing to the Selected Fields listing. Notice that there are three other icons offered. The ">>" icon will select all available fields. The "<" icon allows you to remove the highlighted field from the Selected Fields list while the "<<" icon removes all selected fields. In this example, we want to select the ProductName, UnitsInStock, and UnitsOnOrder from the Product table.



6. **Repeat steps 4 and 5 to add information from additional tables, as desired.** In our example, we wanted to include information about the supplier. That information wasn't included in the Products table -- it's in the Suppliers table. Here's the power of a query! You can combine information from multiple tables and easily show relationships. In this example, we want to include the CompanyName and Phone fields from the Suppliers table. All you have to do is select the fields -- Access will line up the fields for you!

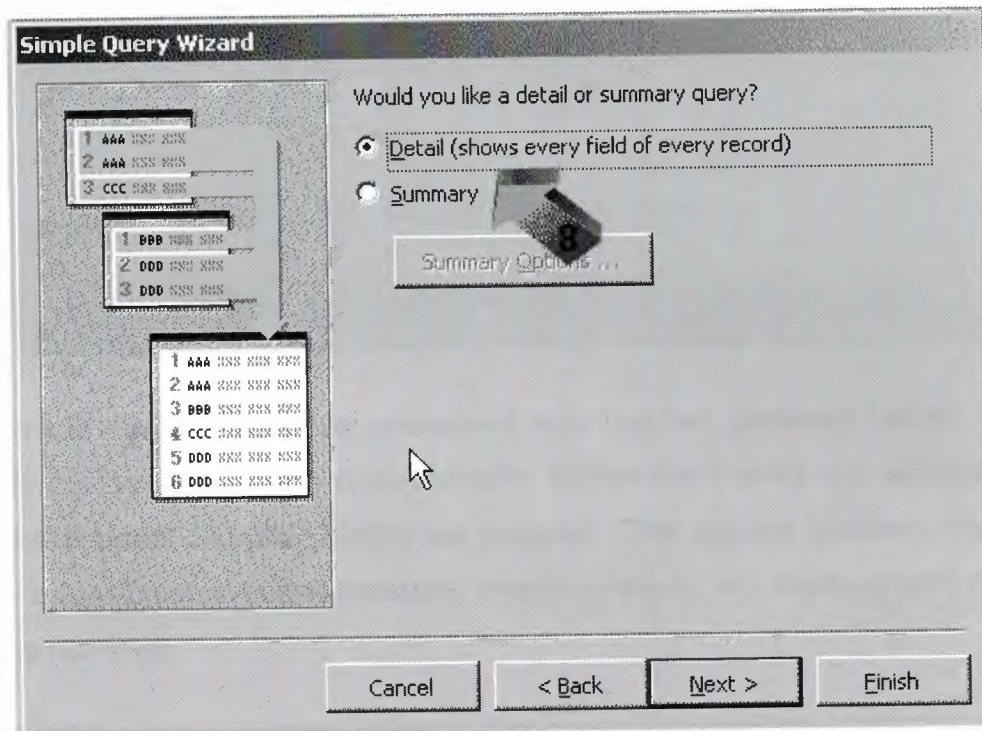
Note that this works because the Northwind database has predefined relationships between tables. If you're creating a new database, you'll need to establish these



relationships yourself. Read the article "Defining Relationships in Microsoft Access" for more information on this topic.

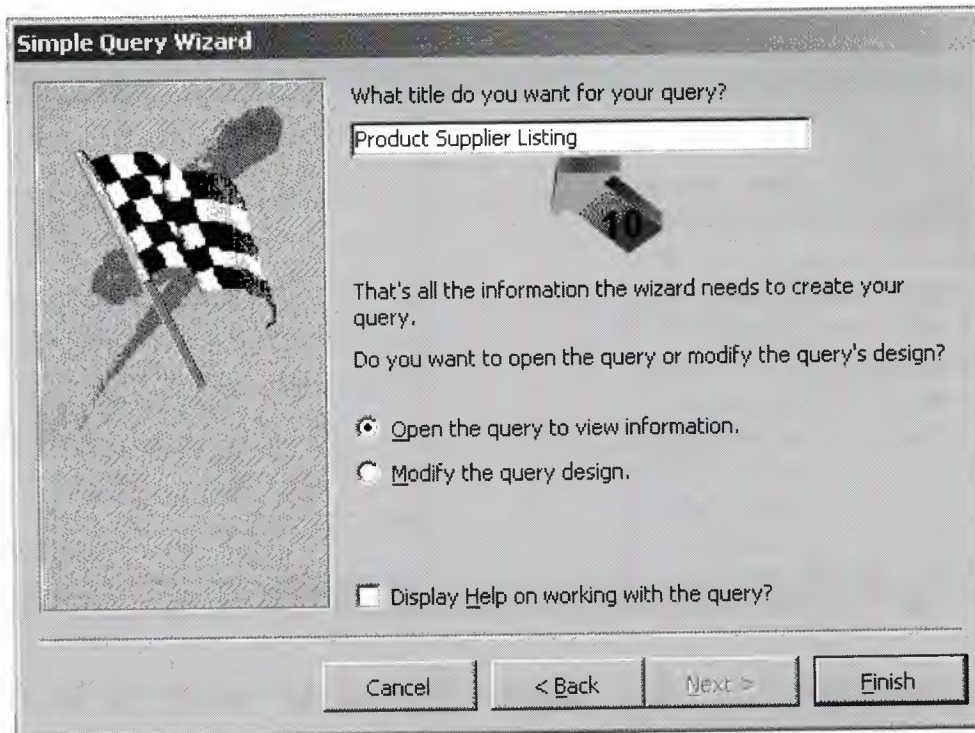
7. **Click on Next.**

8. **Choose the type of results you would like to produce.** We want to produce a full listing of products and their suppliers, so choose the Detail option here.

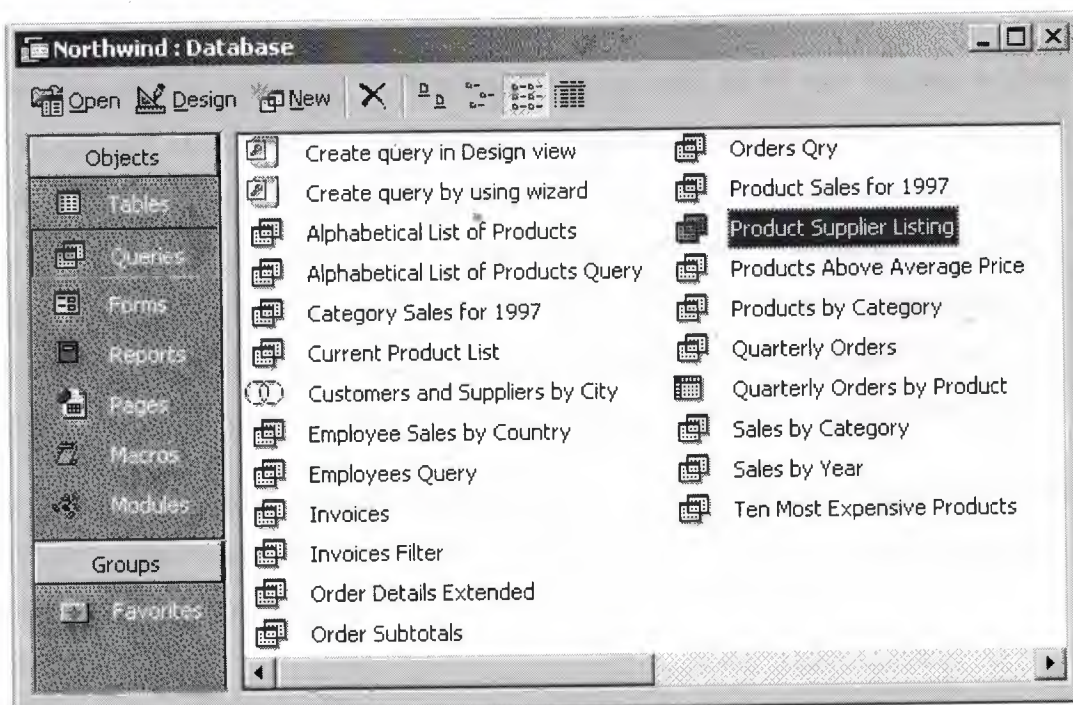


9. **Click on Next.**

10. **Give your query a title.** You're almost done! On the next screen you can give your query a title. Select something descriptive that will help you recognize this query later. We'll call this query "Product Supplier Listing."



11. **Click on Finish.** You'll be presented with the two windows below. The first window is the Query tab that we started with. Notice that there's one additional listing now -- the Product Supplier Listing we created. The second window contains our results -- a list of our company products, inventory levels and the supplier's name and telephone number!





Product Supplier Listing : Select Query					
	Product Name	Units In Stock	Units On Order	Company Name	Phone
►	Chai	39	0	Exotic Liquids	(171) 555-2222
	Chang	17	40	Exotic Liquids	(171) 555-2222
	Aniseed Syrup	13	70	Exotic Liquids	(171) 555-2222
	Chef Anton's Cajun Seasoning	53	0	New Orleans Cajun Delights	(100) 555-4822
	Chef Anton's Gumbo Mix	0	0	New Orleans Cajun Delights	(100) 555-4822
	Louisiana Fiery Hot Peppers	76	0	New Orleans Cajun Delights	(100) 555-4822
	Louisiana Hot Spiced Okra	4	100	New Orleans Cajun Delights	(100) 555-4822
	Grandma's Boysenberry Sauce	120	0	Grandma Kelly's Homestead	(313) 555-5735
	Uncle Bob's Organic Dried Pears	15	0	Grandma Kelly's Homestead	(313) 555-5735
	Northwoods Cranberry Sauce	6	0	Grandma Kelly's Homestead	(313) 555-5735
	Mishi Kobe Niku	29	0	Tokyo Traders	(03) 3555-5011
	Ikura	31	0	Tokyo Traders	(03) 3555-5011
	Longlife Tofu	4	20	Tokyo Traders	(03) 3555-5011
Record: 1 of 77					

Congratulations! You've successfully created your first query using Microsoft Access! Now you're armed with a powerful tool to apply to your database needs.

## Creating Forms in Microsoft Access

### Open your database

Microsoft Access forms provide a quick and easy way to modify and insert records into your databases. They offer an intuitive, graphical environment easily navigated by anyone familiar with standard computer techniques. Creating a form is a quite simple, pleasant experience. In this example, as with all of our Access tutorials, we will use Access 2003 and the Northwind sample database included on the installation CD-ROM. If you're using an earlier version of Access, you may find that some of the menu choices and wizard screens are slightly different. However, the same basic principles apply to all versions of Access (as well as most database systems). Let's begin! Our goal for this tutorial is to create a simple form that will allow data entry operators in our company to easily add new customers to our sales database.

If you haven't already installed the Northwind sample database, these instructions will assist you. Otherwise, go to the Help menu, then choose Sample Databases and Northwind Sample Databases.



**Click on the Forms tab under Objects**

This will bring up a list of the form objects currently stored in your database. Notice that there are a large number of pre-defined forms in this sample database. After you complete this tutorial, you might want to return to this screen and explore some of the more advanced features included in these forms.

**Click on the New icon to create a new form**

Click on the New icon to create a new form

**Select the creation method you wish to use**

Next, we're presented with a variety of different methods we can use to create a form. The AutoForm options quickly create a form based upon a table or query. Design View allows for the creation and formatting of elaborate forms using Access' form editing interface. The Chart Wizard and PivotTable Wizard create forms revolving around those two Microsoft formats. In this tutorial, we'll use the Form Wizard to walk through the process step-by-step.

**Select the data source and click OK.**

You can choose from any of the queries and tables in your database. If you recall our scenario, we wish to create a form to facilitate the addition of customers to our database. In order to accomplish this, we're going to select the Customers table from the pull-down menu.

**Select the form fields to be used and click Next.**

Next, you'll be presented with the screen below. Use this form to select the table/query fields you wish to appear on your form. To add fields one at a time, either double-click the field name or single-click the field name and single click the ">" button. To add all the fields at once, simply click the ">>" button. The "<" and "<<" buttons work in a similar manner to remove fields from the form. For our example, we will add all of the table's fields to the form.

**Select the form layout and click Next**

You can choose from either a columnar, tabular, datasheet or justified form layout. We'll use the justified layout to produce an organized form with a clean layout. You may wish to come back to this step later and explore the various layouts available.

**Select the form style and click Next.**

Microsoft Access includes a number of built-in styles to give your forms an attractive appearance. Click on each of the style names to see a preview of your form and choose the one you find most appealing.

**Provide a title for your form**

Select something easily recognizable -- this is how your form will appear in the database menu. Let's call our form "Customers" in this case. Select the next action and click Finish. You may open the form as a user will see it and begin viewing, modifying and/or entering new data. Alternatively, you may open the form in design view to make modifications to the form's appearance and properties. Let's do the latter and explore some of the options available to us.

**Edit Properties**

Click the Properties icon. This will bring up a menu of user-definable attributes that apply to our form. Edit the properties as necessary. Recall that our original goal was to create a form for data entry purposes. Most likely, we don't want to grant data entry employees full access to view or edit customer records. Setting the "Data Entry" property to Yes will only allow users to insert new records and modify records created during that session.

**Encrypting an Access Database**

Security-conscious database users have long called for the ability to use strong encryption in Microsoft Access. With the release of Access 2007, Microsoft answered these pleas and introduced a robust encryption feature that allows for the simple addition of a great deal of security to Access databases.

**What is encryption?**

Encryption provides you with the ability to protect your database file from prying eyes. It transforms the way data is stored on your disk so that individuals who do not know the database password can not open the database or use other techniques to view the file contents. Security professionals recommend the use of encryption to protect sensitive information.

## **How do I encrypt an Access 2007 database?**

Access 2007 users may encrypt databases stored in ACCDB format by password-protecting them.

Note that this feature is not available for database stored in the older MDB format. You may find the following articles useful when attempting to encrypt an Access database:

- Password-protecting an Access 2007 Database, Step-by-Step
- ACCDB Database Format
- Convert older Access databases to Access 2007

## **How do I decrypt an Access 2007 database?**

If you want to open an encrypted database for use and then reencrypt it when you are finished, Microsoft Access will handle the mechanics for you. Simply open the database as you normally would and enter the database password when prompted. Access will decrypt the database for your use and then save a new encrypted copy when you make changes.

If you want to remove encryption from an encrypted Access database, open the database in exclusive mode and then click "Decrypt Database" in the Database Tools group.

## **What type of encryption does Access 2007 use?**

Access 2007 uses the Microsoft Cryptographic API. This means that it will support any cryptographic algorithm available within Windows as a Cryptographic Service Provider (CSP). This is a great improvement over earlier versions of Access, which only supported a built-in, weak encryption algorithm.



## A brief history of Borland's Delphi

### Pascal

Delphi uses the language Pascal, a third generation structured language. It is what is called a **highly typed** language. This promotes a clean, consistent programming style, and, importantly, results in more reliable applications. Pascal has a considerable heritage:

### Beginnings

Pascal appeared relatively late in the history of programming languages. It probably benefited from this, learning from Fortran, Cobol and IBM's PL/1 that appeared in the early 1960's. Niklaus Wirth is claimed to have started developing Pascal in 1968, with a first implementation appearing on a CDC 6000 series computer in 1970.

Curiously enough, the **C** language did not appear until 1972. C sought to serve quite different needs to Pascal. C was designed as a high level language that still provided the low level access that assembly languages gave. Pascal was designed for the development of structured, maintainable applications.

### The 1970's

In 1975, Wirth teamed up with Jensen to produce the definitive Pascal reference book "Pascal User Manual and Report". Wirth moved on from Pascal in 1977 to work on **Modula** - the successor to Pascal.

### The 1980's

In 1982 ISO Pascal appears. The big event is in November 1983, when **Turbo Pascal** is released in a blaze of publicity. Turbo Pascal reaches release 4 by 1987. Turbo Pascal excelled on speed of compilation and execution, leaving the competition in its wake.

## From Turbo Pascal to Delphi

Delphi, **Borland's** powerful **Windows?** and **Linux?** programming development tool first appeared in 1995. It derived from the **Turbo Pascal?** product line.

As the opposition took heed of Turbo Pascal, and caught up, Borland took a gamble on an Object Oriented version, mostly based on the Pascal object orientation extensions. The risk paid off, with a lot of the success due to the thought underlying the design of the IDE (Integrated Development Environment), and the retention of fast compilation and execution.

This first version of Delphi was somewhat limited when compared to today's heavyweights, but succeeded on the strength of what it did do. And speed was certainly a key factor. Delphi went through rapid changes through the 1990's.

## Delphi for Microsoft .Net

From that first version, Delphi went through 7 further iterations before Borland decided to embrace the competition in the form of the **Microsoft? .Net** architecture with the stepping stone Delphi 8 and then fully with Delphi 2005 and 2006. Delphi however still remains, in the opinion of the author, the best development tool for stand alone Windows and Linux applications. Pascal is a cleaner and much more disciplined language than Basic, and adapted much better to Object Orientation than Basic.



## **A new direction**

Delphi is now provided by a development tools only company.

### ***Delphi For Beginners:***

**Your guide will try to explain exactly what is Delphi and what can it do for you.**

Dateline: 1999

### **Preparations.**

First of all, I will presume that you know what computers are, what can you do with them, and finally what does programming mean, in general. It would also be great if you already have basic knowledge of programming (Pascal perhaps?).

If this is not true, you wouldn't be here anyway (am I right?). I'll be very glad if I'm not! So sit back, relax and enjoy reading this article.

### **Delphi**

Borland Delphi is a development tool for Microsoft Windows applications. Delphi is powerful and easy to use tool for generating stand-alone graphical user interface (GUI) programs or 32-bit console applications (programs that have no GUI presence but instead run in what is commonly referred to as a "DOS box.")

When paired with Borland Kylix, Delphi users can build single-source applications for both Windows and Linux, which opens new opportunities and increases the potential return on development investments.\* Use the Cross-platform CLX component library and visual designers to build high-performance portable applications for Windows that can be easily re-compiled on Linux.

Delphi is the first programming language to shatter the barrier between high-level, easy-to-use rapid application development environments and low-level bits-and-bytes power tools.

When creating GUI applications with Delphi, you have all the power of a true compiled programming language (Object Pascal) wrapped up in a RAD environment. All the common parts of the Windows graphical user interface, like forms, buttons and lists

objects, are included in Delphi as components. This means that you don't have to write any code when adding them to your application. You simply draw them onto your form like in a paint program. You can also drop ActiveX controls on forms to create specialized programs such as Web browsers in a matter of minutes. Delphi allows the developer to design the entire interface visually, and quickly implement an event driven code with the click of the mouse.

Delphi ships in a variety of configurations aimed at both departmental and enterprise needs. With Delphi, you can write Windows programs more quickly and more easily than was possible ever before.

### **Pascal**

The best way of describing Delphi is an Object Pascal-based visual development environment. Delphi's environment is based on Object Pascal, a language that is as object oriented as C++, and in some cases, better. For developers with no Pascal experience, its templates for Pascal program structures speed the process of learning the language.

The compiler produces applications packaged in compact executable files, with no need for bulky runtime libraries (DLL's)-a notable benefit, I must say.

### **VCL**

Visual Component Library (self-contained binary piece of software that performs some specific predefined function), or VCL, is Delphi's object-oriented framework. In this rich library, you'll find classes for Windows objects such as windows, buttons, etc, and you'll also find classes for custom controls such as gauge, timer and multimedia player, along with non-visual objects such as string lists, database tables, and streams.

### **Databases**

Delphi can access many types of databases. Using forms and reports that you create, the BDE (Borland Database Engine) can access local databases, like Paradox and dBase, network SQL server databases, like InterBase, and SysBase, and any data source accessible through ODBC (open database connectivity).



## Hello World!

At the end let's see one of the smallest Delphi applications: the famous 'Hello World!' program.

This example is not for beginners – there is no main form of application or something like that. This is only a demonstration. In some of the future articles I will focus on topics like Delphi for Beginners - How to get started.

```
program HelloWorld;  
uses dialogs;  
begin  
  ShowMessage('Hello World!');  
end.
```



## A Glossary of Delphi Programming Technical Terms

Definitions of terms having to do with Delphi programming, Pascal, OOP, BDE and programming in general

### "Self"

Definition: Within the implementation of a method, the identifier Self references the object in which the method is called.

### type

TCar = **Class**

color : TColor;

procedure ChangeColor(newColor : TColor) ;

**end;**

...

**procedure** TCar.ChangeColor(newColor : TColor) ;

**begin**

*//self is "this" instance*

```
Self.color := newColor;  
end;
```

In class methods the identifier Self represents the class where the method is called.

### "Constructor"

Definition: A constructor is a special method that creates and initializes instance objects. The declaration of a constructor looks like a procedure declaration, but it begins with the reserved word constructor.

A class can have more than one constructor, but most have only one. It is conventional to call the constructor Create.

To create an object, call the constructor method on a class type.

### type

```
TCar = Class  
  constructor Create;  
end;
```

...

```
car := TCar.Create;
```

### "Reserved Word"

Definition: A special word reserved by a programming language or by a program.

You are not allowed to use reserved words as variable names.

A partial list of Delphi reserved words:

- and
- array
- as
- asm
- begin



- case
- class
- const
- constructor
- destructor
- dispinterface
- div
- do
- downto
- else
- end
- except
- exports
- file
- finalization
- finally
- for
- function
- goto
- if
- implementation
- in
- inherited
- initialization
- interface
- in
- is
- library
- nil
- not
- object
- of
- or
- out

- packed
- procedure
- program
- property
- raise
- record
- repeat
- resourcestring
- set
- string
- then
- to
- try
- type
- unit
- until
- uses
- var
- while
- with

In addition to the words above, private, protected, public, published, and automated act as reserved words within object type declarations, but are otherwise treated as directives.

### **"Class Method"**

Definition: A class method is a method that operates on classes instead of objects.

The definition of a class method must begin with the reserved word class.

The most common used class method in Delphi language is the "Create" constructor.

In the defining declaration of a class method, the identifier Self represents the class where the method is called (which could be a descendant of the class in which it is



defined). If the method is called in the class TCar, then Self is of the type class of TCar.

## "Method"

Definition: Procedure or function (routine) associated with a particular object.

Different classes may define methods with the same name (Car.Drive or Scooter.Drive).

Most methods operate on objects that are instances of a certain class.

A class method is a method (other than a constructor) that operates on classes instead of objects.

A call to a method specifies the object (or, if it is a class method, the class) that the method should operate on.

Examples:

### type

TCar = **Class**

*//method procedure*

**procedure** Drive;

*//method (function)*

**function** ChangeGear(newGear : integer) ;

**end;**

## "Object"

Definition: An object is a variable of class. More generally, a variable of any type.

An instance of a class or object, is a self-contained entity that consists of both properties, events and methods to manipulate the data.

Each object has its own values for the instance variables of its class and can respond to the methods as well as raise events defined by its class.

Also Known As: Instance variable

### "Canvas"

Definition: Canvas is the graphical drawing surface of an object. The canvas has a brush, a pen, a font, and an array of pixels. The canvas encapsulates the Windows device context.

In Delphi, the TCanvas class provides an abstract drawing space for objects that must render their own images.

### "Class"

Definition: A list of features representing data and associated code assembled into single entity. A class includes not only features listed in its definition but also features inherited from ancestors.

The terms class and type are usually (but not always) interchangeable; a class is a slightly different concept than a type, in that it emphasizes the classifications of structure and behavior.

Classes are related in a class hierarchy. One class may be a specialisation (a "subclass") of another (one of its "superclasses"). A class may be an abstract class or a concrete class.

The Visual Component Library (CVL) is a class hierarchy of Delphi components and object types.

Also Known As: Object Type

Examples:

#### **type**

TCar = **Class**

Year : integer;

Color : TColor;

**end;**



## **"Run Time"**

Definition: Run time is any time you are actually running the application in the operating system and interacting with the application as the user would.

In Delphi, "dynamically creating ..." means "creating at run-time".

## **"RTL"**

Definition: The raw power of Delphi is based on a considerable amount of its Run Time Library functions and procedures.

RTL is the collection of functions and procedures that are built into Delphi.

Also Known As: Run Time Library; VCL Routines

## **"Routine"**

Definition: Self-contained statement blocks that can be called from different locations in a program. In Delphi: function or procedure.

Also Known As: Subroutine

## **"Recursion "**

Definition: Recursion is a very simple, yet useful and powerful programmer's tool. As we know, routines can, and frequently do, call other routines.

A routine that activates/calls itself is called recursive. Recursion is a general method of solving problems by reducing them to simpler problems of a similar type.

A recursive subroutine constantly calls itself, each time in a simpler situation, until it gets to the trivial case, at which point it stops.

## **"Procedure"**

Definition: A procedure is a routine that does not return a value (unlike a function).

Procedure header gives the name of a procedure followed by a list of formal parameters.

In a unit, a routine may have a header declared in the interface part, and then again in the implementation part. The second appearance of the header may be an exact duplicate of the header in the interface part, or may be only the name of the routine.

Examples: `[blockquote shade=yes] procedure TestMe(parameter: TCustomType[br] begin[br] ... end; [/blockquote]_z_delphi_z_);`

### **"Pointer"**

Definition: A pointer is a variable that holds the address of another variable (or routine) in memory.

A pointer can be used to indirectly manipulate the object.

### **"Parameter"**

Definition: Represents one value that is supplied by one function (the calling function) that wishes to make use of the services of another function (the called function).

In Delphi, every parameter is classified as value, variable, constant, or out.

Also Known As: Argument

Examples: `[blockquote] "year" and "name" are parameters for the "TestMe" function`

`procedure TestMe(const year: integer; var name : string) ; [/blockquote]`

### **"OLE"**

Definition: OLE is a compound document standard developed by Microsoft Corporation. It enables you to create objects with one application and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them.

With OLE, data from a server application is stored in a container application. The data is stored in an OLE Object.

Also Known As: Object Linking and Embedding

## **"MDI"**

Definition: A Windows API that enables programmers to easily create applications with multiple windows.

Each MDI application has a single main (frame) window, and any number of child windows (documents). All child windows are displayed within the main window - this is common in applications such as spreadsheets or word processors.

The child window's document title merges with the parent window's title bar when the child window is maximized.

Although many programmers still use MDI, Microsoft recommends using a newer API called Single Document Interface (SDI).

Also Known As: Multiple Document Interface

## **"IDE"**

Definition: IDE (Integrated Development Environment) is the user interface (GUI) where you can design, compile and debug your Delphi projects.

Also Known As: Integrated Development Environment

## **"GUI"**

Definition: A GUI (usually pronounced GOO-ee) is a graphical (rather than purely textual) user interface to a computer.

Applications typically use the elements of the GUI that come with the operating system and add their own graphical user interface elements and ideas. When creating an application, Delphi facilitate writing a graphical user interface.

Each GUI element (for example a Button or an EditBox) is defined as a class from which you can create object instances for your application.

Also Known As: Graphical User Interface

Alternate Spellings: goo-ee



## "Function"

Definition: A function is a routine that returns a value when it executes.

It can be passed and it can return a value. Functions that are part of a class are usually called methods.

You can code your own functions or use built-in functions provided by Delphi RTL (run time library).

Examples:

```
function YearsOld(const BirthYear:integer): integer;  
var  
Year, Month, Day : Word;  
begin  
DecodeDate(Date, Year, Month, Day) ;  
Result := Year - BirthYear;  
end;
```

## "Freeware"

Definition: Copyrighted software given away for free by the author. Although it is available for free, the author retains the copyright, which means that you cannot do anything with it that is not expressly allowed by the author. Usually, the author allows people to use the software, but not sell it.

## "Exception"

Definition: An event happening during execution of a program that disrupts the normal flow of control. Exceptions are raised when a runtime error occurs in an application, such as attempting to divide by zero.

Also, an exception is an object that contains information about what error occurred and where it happened.

## **"Design Time"**

**Definition:** We work with forms and controls, set their properties, and write code for their events at design time, which is any time we're building an application in the Delphi's IDE.

Design-time is when you use the IDE to design your application, using the form, the Object Inspector, Component palette, Code editor, and so forth; as opposed to run-time, when the application you design is actually running.

## **"Compiler"**

**Definition:** A compiler is a program that performs the process of compilation. When you press F9 in Delphi IDE your current project gets compiled and run.

## **"Compilation"**

**Definition:** Compilation is the process of translating source code into an object program, which is composed of machine instructions along with the data needed by those instructions. Virtually all of the software on your computer was created by this process.

Compiled programs (Delphi applications for example) run faster than "interpreted" - which is the line-by-line translation of source code to machine instructions (Visual Basic applications for example).

## **"Comment"**

**Definition:** The purpose of adding comments to Delphi code is to provide more program readability using understandable description of what your code is doing.

A comment is a note to yourself or another programmer; it is ignored by the compiler.

There are several ways to construct comments:

{ Text between a left brace and a right brace constitutes a comment. }

(\* Text between a left-parenthesis-plus-asterisk and an asterisk-plus-right-parenthesis also constitutes a comment. \*)

// Any text between a double-slash and the end of the line constitutes a comment.

Also Known As: REM meaning "Remark in Basic"

## **"COM"**

Definition: The Component Object Model (COM) enables programmers to develop objects that can be accessed by any COM-compliant application. Both OLE and ActiveX are based on COM.

The key aspect of COM is that it enables communication between clients and servers through interfaces. Information about these interfaces is usually included in a type library.

COM allows you to create COM objects that are not specific to any language, and in some cases, even platforms. For instance, COM objects can be ported to a Unix system. COM also allows you to create COM Objects that will be instantiated on a different machine across the world if you so desired.

Although often associated with Microsoft, COM is an open standard that specifies how components work together and interoperate.

Also Known As: Component Object Model

## **"Callback Routine"**

Definition: A callback routine is a routine (function or procedure) in your program that Windows calls. More generally, a callback is a means of sending a function as a parameter into another function. When the callback function has completed, control is passed back to the original function.

For example, EnumFonts is a Windows routine that calls a given callback function for every font installed in the system.

## **"BDE"**

Definition: The core database engine and connectivity software behind Borland products, as well as Paradox for Windows and Visual dBASE for Windows. The



included set of database drivers enables consistent access to standard data sources: Paradox, dBASE, FoxPro, Access, and text databases.

Many Delphi components use this database engine to access and deliver data. BDE maintains information about your PC's environment in the BDE configuration file (usually called IDAPI.CFG). Use the BDE Administrator to change the settings in this configuration file.

Also Known As: Borland Database Engine, IDAPI

## **"Application"**

Definition: An application is the executable file and all related files that a program needs to function which serve a common purpose or purposes, as distinguished from the design and source code of the project.

Software applications can be divided into two general classes: systems software and applications software. Systems software consists of low-level programs that interact with the computer at a very basic level. This includes operating systems, compilers, and utilities for managing computer resources.

In contrast, applications software (also called end-user programs) includes database programs, word processors, spreadsheets, etc. Figuratively speaking, applications software sits on top of systems software because it is unable to run without the operating system and system utilities.

In general we use Delphi to produce applications software.

## **"API"**

Definition: A set of routines, protocols, and tools for building software applications. A wide variety of software from operating systems to individual components are said to have an API.

A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Also Known As: Application Programming Interface

## **"Alias"**

Definition: A name that specifies the location of database tables accessed using the BDE. The terms alias and database are \*synonymous\* when talking about the BDE. An alias specifies driver parameters and database locations, such as Driver Type, Server Name, User Name and others.

## **"Algorithm"**

Definition: An algorithm is a set of precisely defined steps guaranteed to arrive at an answer to a problem or set of problems. As this implies, a set of steps that might never end is not an algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

## **"ActiveX"**

Definition: A technology that allows various software components to communicate and interact, even though they are not written in the same language. ActiveX controls can be embedded in Web pages to produce animation and other multimedia effects, interactive objects and sophisticated applications.

An ActiveX control is a COM-based software component that integrates into and extends the functionality of any host application. ActiveX controls implement a set of predefined COM interfaces.

The ActiveX page of the component palette includes several ActiveX controls. You can use them like any standard VCL component, dropping them on forms and setting their properties using the Object Inspector.

## **"ASCII"**

Definition: ASCII assigns each English character and basic punctuation mark its own number from 0 to 127. Since the code is standard, every computer should be able to translate it into serviceable, if unglamorous, copy. So, when you're unsure what program - or what computer - is on the receiving end of a document, your safest bet is to save your file as plain ASCII text.

Examples: The capital letter A has an ASCII value of 65. The ASCII code for a space is 32.

You can reference a character by its ASCII code prefixed with a number sign (#).

Example: To put the symbol for American cents into a character C, for example, you could code "c := #155;".

Pronunciation: ask-ee

Also Known As: American Standard Code for Information Interchange

## **Understanding Delphi Project Files (.DPR)**

### **New: Delphi Project**

Since it is quite common for Delphi applications to share code or previously customized forms, Delphi organizes applications into what is called projects.

A project is made up of the visual interface along with the code that activates the interface. Each project can have multiple forms, allowing us to build applications that have multiple windows. The code that is needed for a form in our project is stored in a separate Unit file that Delphi automatically associates to the form. General code that we want to be shared by all the forms in our application is placed in unit files as well. Simply put, a Delphi project is a collection of files that make up an application.

What this means is that each project is made of one or more Form files (files with the *.dfm* extension) and one or more Unit files (*.pas* extension).

We can also add resource files, and they are compiled into *.RES* files and linked when we compile the project.

### **Project File**

Each project is made up of a single project file (*.dpr*). Project files contain directions for building an application. This is normally a set of simple routines which open the main form and any other forms that are set to be opened automatically and then starts the program by calling the *Initialize*, *CreateForm* and *Run* methods of the



global Application object (which is actually a form of zero width and height, so it never actually appears on the screen).

Note: The global variable *Application*, of type TApplication, is in every Delphi Windows application. Application encapsulates your application as well as provides many functions that occur in the background of the program. For instance, Application would handle how you would call a help file from the menu of your program.

## Project Unit

Use Project - View Source to display the project file for the current project.

Although you can look and edit the Project File, in most cases, you'll let Delphi maintain the DPR file. The main reason to view the project file is so we can see the units and forms that make up the project, and which form is specified as the application's main form.

Another reason to work with the project file is when we are creating a DLL rather than a stand-alone application or need some start-up code, such as a splash screen before the main form is created by Delphi.

Here is the default project file for a new application (containing one form: "Form1"):

```
program Project1;  
uses  
    Forms,  
    Unit1 in 'Unit1.pas' {Form1};  
{$R *.RES}  
begin  
    Application.Initialize;  
    Application.CreateForm(TForm1, Form1) ;  
    Application.Run;  
end.
```

The **program** [link url=/od/delhiprogrammingglossary/g/reservedword.htm]keyword identifies this unit as a program's main source unit. You can see that the unit name,

Project1, follows the program keyword (Delphi gives the project a default name until you save the project with a more meaningful name). When we run a project file from the IDE, Delphi uses the name of the Project file for the name of the EXE file that it creates.

Delphi reads the **uses** clause of the project file to determine which units are part of a project.

The .dpr file is linked with the .pas file with the compile directive `{$R *.RES}` (in this case '\*' represents the root of the .pas filename rather than "any file"). This compiler directive tells Delphi to include this project's resource file. The project's resource file contains such items as the project's icon image.

The **begin..end** block is the main source-code block for the project.

Although **Initialize** is the first method called in the main project source code, it is not the first code that is executed in an application. The application first executes the **"initialization" section of all the units** used by the application.

The **Application.CreateForm** statement loads the form specified in its argument. Delphi adds an **Application.CreateForm** statement to the project file for each form you add to the project. This code's job is to first allocate memory for the form. The statements are listed in the order the forms are added to the project. This is the order that the forms will be created in memory at runtime. If you want to change this order, do not edit the project source code. Use the Project|Options menu command.

The **Application.Run** statement starts your application. This instruction tells the predeclared object called **Application** to begin processing the events that occur during the run of a program.

### **An Example: Hide Main Form / Hide Taskbar Button**

The **Application** object's **ShowMainForm** property determines whether or not a form will show at startup. The only condition of setting this property is that it has to be called before the **Application.Run** line.

```
//Presume: Form1 is the MAIN FORM  
Application.CreateForm(TForm1, Form1) ;  
Application.ShowMainForm := False;  
Application.Run;
```

## **Understanding the Birth, Life and Death of a Delphi Form**

### **Life-Cycle of a Delphi Form**

In Windows, most elements of the user interface are windows. In Delphi, every project has at least one window - program's main window. All windows of a Delphi application are based on TForm object.

#### **Form**

Form objects are the basic building blocks of a Delphi application, the actual windows with which a user interacts when they run the application. Forms have their own properties, events, and methods with which you can control their appearance and behavior. A form is actually a Delphi component, but unlike other components, a form doesn't appear on the component palette.

We normally create a form object by starting a new application (File | New Application). This newly created form will be, by default, the application's main form - the first form created at runtime.

Note: To add an additional form to Delphi project, we select File|New Form.

There are, of course, other ways to add a "new" form to a Delphi project.

#### **Birth**

##### **OnCreate**

The OnCreate event is fired when a TForm is first created, that is, only once. The statement responsible for creating the form is in the project's source (if the form is set to be automatically created by the project). When a form is being created and its Visible property is True, the following events occur in the order listed: OnCreate, OnShow, OnActivate, OnPaint.



You should use the OnCreate event handler to do, for example, initialization chores like allocating string lists.

Any objects created in the OnCreate event should be freed by the OnDestroy event.

OnCreate -> OnShow -> OnActivate -> OnPaint -> OnResize -> OnPaint ...

### **OnShow**

This event indicates that the form is being displayed. OnShow is called just before a form becomes visible. Besides main forms, this event happens when we set forms Visible property to True, or call the Show or ShowModal method.

### **OnActivate**

This event is called when the program activates the form - that is, when the form receives the input focus. Use this event to change which control actually gets focus if it is not the one desired.

### **OnPaint, OnResize**

Events like OnPaint and OnResize are always called after the form is initially created, but are also called repeatedly. OnPaint occurs before any controls on the form are painted (use it for special painting on the form).

### **Life**

As we have seen the birth of a form is not so interesting as the life and death can be. When your form is created and all the controls are waiting for events to handle, the program is running until someone tries to close the form!

### **Death**

An event-driven application stops running when all its forms are closed and no code is executing. If a hidden form still exists when the last visible form is closed, your application will appear to have ended (because no forms are visible), but will in fact continue to run until all the hidden forms are closed. Just think of a situation where the main form gets hidden early and all other forms are closed.

... OnCloseQuery -> OnClose -> OnDeactivate -> OnHide -> OnDestroy

## OnCloseQuery

When we try to close the form using the Close method or by other means (Alt+F4), the OnCloseQuery event is called. Thus, event handler for this event is the place to intercept a form's closing and prevent it. We use the OnCloseQuery to ask the users if they are sure that they really want the form to close.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean) ;  
begin  
  if MessageDlg('Really close this window?', mtConfirmation, [mbOk, mbCancel], 0) =  
  mrCancel then CanClose := False;  
end;
```

An OnCloseQuery event handler contains a CanClose variable that determines whether a form is allowed to close. The OnCloseQuery event handler may set the value of CloseQuery to False (via the CanClose parameter), thus aborting the Close method.

## OnClose

If OnCloseQuery indicates that the form should be closed, the OnClose event is called.

The OnClose event gives us one last chance to prevent the form from closing. The OnClose event handler has an Action parameter, with the following four possible values:

- **caNone**. The form is not allowed to close. Just as if we have set the CanClose to False in the OnCloseQuery.
- **caHide**. Instead of closing the form you hide it.
- **caFree**. The form is closed, so it's allocated memory is freed by Delphi.
- **caMinimize**. The form is minimized, rather than closed. This is the default action for MDI child forms. Note: When a user shuts down Windows, the OnCloseQuery event is activated, not the OnClose. If you want to prevent Windows from shutting down, put your code in the OnCloseQuery event handler, of course CanClose=False will not do the trick.

## OnDestroy

After the OnClose method has been processed and the form is to be closed, the OnDestroy event is called. Use this event for operations opposite to those in the OnCreate event. OnDestroy is therefore used to deallocate objects related to the form and free the corresponding memory.

Of course, when the main form for a project closes, the application terminates.

## Understanding and Using Functions and Procedures

Have you ever found yourself writing the same code over and over to perform some common task within event handlers? Yes! It's time for you to learn about programs within a program. Let's call those mini programs subroutines.

### Intro to subroutines

Subroutines are an important part of any programming language, and Delphi is no exception. In Delphi, there are generally two types of subroutines: a **function** and a **procedure**. The usual difference between a function and a procedure is that **a function can return a value, and a procedure generally will not do so**. A function is normally called as a part of an expression.

Take a look at the following examples:

```
procedure SayHello(const sWhat:string) ;
```

```
begin
```

```
    ShowMessage('Hello ' + sWhat) ;
```

```
end;
```

```
function YearsOld(const BirthYear:integer): integer;
```

```
var
```

```
    Year, Month, Day : Word;
```

```
begin
```

```
    DecodeDate(Date, Year, Month, Day) ;
```



```
Result := Year - BirthYear;  
end;
```

Once subroutines have been defined, we can call them one or more times:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
begin  
    SayHello('Delphi User') ;  
end;
```

```
procedure TForm1.Button2Click(Sender: TObject) ;  
begin  
    SayHello('Zarko Gajic') ;  
    ShowMessage('You are ' + IntToStr(YearsOld(1973)) + ' years old!') ;  
end;
```

## Functions and Procedures

As we can see, both functions and procedures act like mini programs.

In particular, they can have their own type, constants and variable declarations inside them.

Take a closer look at a (miscellaneous) SomeCalc function:

```
function SomeCalc  
    (const sStr: string;  
     const iYear, iMonth: integer;  
     var iDay: integer): boolean;  
begin  
    ...  
end;
```

Every procedure or function begins with a *header* that identifies the procedure or function and lists the *parameters* the routine uses, if any. The parameters are listed within parentheses. Each parameter has an identifying name and usually has a type. A semicolon separates parameters in a parameter list from one another.

sStr, iYear and iMonth are called *constant parameters*. Constant parameters cannot be changed by the function (or procedure). The iDay is passed as a *var parameter*, and we can make changes to it, inside the subroutine.

Functions, since they return values, must have a *return type* declared at the end of the header. The return value of a function is given by the (final) assignment to its name. Since every function implicitly has a local **variable Result** of the same type as the functions return value, assigning to Result has the same effect as assigning to the name of the function.

### Positioning and Calling Subroutines

Subroutines are always placed inside the implementation section of the unit. Such subroutines can be called (used) by any event handler or subroutine in the same unit that is defined after it.

Note: the uses clause of a unit tells you which units it can call. If we want a specific subroutine in a Unit1 to be usable by the event handlers or subroutines in another unit (say Unit2), we have to:

- Add Unit1 to the uses clause of Unit2
- Place a copy of the header of the subroutine in the interface section of the Unit1.

This means that subroutines whose headers are given in the interface section are *global in scope*.

When we call a function (or a procedure) inside its own unit, we use its name with whatever parameters are needed. On other hand, if we call a global subroutine (defined in some other unit, e.g. MyUnit) we use the name of the unit followed by a period.

...

*//SayHello procedure is defined inside this unit*

SayHello('Delphi User') ;

*//YearsOld function is defined inside MyUnit unit*

```
Dummy := MyUnit.YearsOld(1973) ;
```

Note: functions or procedures can have their own subroutines **embedded** inside them. An embedded subroutine is local to the container subroutine and cannot be used by other parts of the program. Something like:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
  function IsSmall(const sStr:string):boolean;  
  begin  
    //IsSmall returns True if sStr is in lowercase, False otherwise  
    Result:=LowerCase(sStr)=sStr;  
  end;  
begin  
  //IsSmall can only be uses inside Button1 OnClick event  
  if IsSmall(Edit1.Text) then  
    ShowMessage('All small caps in Edit1.Text')  
  else  
    ShowMessage('Not all small caps in Edit1.Text') ;  
end;
```

## Understanding and Using Decisions

```
if language = Delphi then  
begin  
  Use(language)  
end  
else  
  Skip(language) ;
```

## Branching

If you want to **control the flow of code execution** depending on what the program has already done or what it has just encountered you need to use one of the two Delphi Pascal branching statements: **if statements** and **case statements**.



## The IF THEN ELSE statement

The if statement is used to test for a condition and then execute sections of code based on whether that condition is True or False. The condition is described with a Boolean expression, If the condition is True, the code flow branches one way. If the condition is False, the flow branches in another direction. Let's see this behavior on an example:

```
var iNumber : Integer;  
begin  
    //some value must be  
    //assigned to iNumber here!  
    if iNumber = 0 then  
        ShowMessage('Zero value encountered!') ;  
    end;
```

If the number (assigned to iNumber variable) is 0, the expression `iNumber = 0` evaluates to True and the message is displayed; otherwise, nothing is displayed.

If we want more than one thing to happen when the tested condition is True, we can write multiple statements in a **begin ... end** block.

```
var iNumber : Integer;  
begin  
    //some value must be  
    //assigned to iNumber here!  
    if iNumber = 0 then  
        begin  
            ShowMessage('Zero value encountered!') ;  
            Exit; // exit from the current procedure  
        end;  
        //if iNumber is 0 the following  
        //code will never be executed  
        ShowMessage('Nobody likes 0, ha!') ;  
    end;
```

More often, we will want to process multiple statements if a condition is True or False.

```

var iNumber : Integer;
begin
    //some value must be
    //assigned to iNumber here!
    if iNumber < 0 then
        begin
            //statements ...
            ShowMessage('Your number is negative!') ;
            //statements ...
        end
    else
        begin
            //statements ...
            ShowMessage('Your number is positive or zero!') ;
            //statements ...
        end;
    end;
end;

```

Note: Each statement in the begin..end block ends with a semicolon. We cannot have a semicolon before or after the else keyword. The if-then-else statement, is a single statement, therefore we cannot place a semicolon in the middle of it.

An if statement can be quite complex. The condition can be turned into a series of conditions (using the and, or and not Boolean operators), or the if statement can nest a second if statement.

```

var
    iNumber : Integer;
begin
    if iNumber = 0 then
        begin
            ShowMessage('Zero number not allowed!') ;
            exit;
        end
    else

```

*//no need to use begin-end here*

**if** iNumber < 0 **then**

    ShowMessage('Your number is negative!')

**else**

    ShowMessage('Your number is positive!') ;

**end;**

**Note:** When you write nested if statements choose a consistent, clear indentation style. This will help you and anyone else who reads your code see the logic of the if statement and how the code flows when your application runs.

### **The CASE statement**

Although, we can use the if statement for very complex (nested) condition testing, the case statement is usually easier to read (debug!) and the code runs more quickly.

The case statement makes it clear that a program has reached a point with many branches; multiple if-then statements do not.

**var**

    iNumber : Integer;

**begin**

*//some value must be*

*//assigned to iNumber here!*

**case** iNumber **of**

        0 : ShowMessage('Zero value') ;

        1..10 : ShowMessage('Less than 11, greater than 0') ;

        -1, -2, -3 : ShowMessage('Number is -1 or -2 or -3') ;

**else**

        ShowMessage('I do not care') ;

**end;**

**end;**

What follows the case keyword is usually called the selector. The selector is a variable or expression taken from either the char type or any integer type (an ordinal



type). String type are *invalid!*. However, the StringToCaseSelect custom function enables you to use the Case statement with string type variables

As you can see, the individual case statements use a single constant, a group of constants (separated by comma), or a range of constants (double dot separated). We can even add an else keyword to take care of all the remaining cases at once.

Note 1: Only one case statement will be executed, we cannot have overlapping conditions in the case statements.

Note 2: If you want to include more than one statement in the part following the colon (:), place the begin and end keywords around the multiple statements.

## Understanding and Using Loops

### Repeating operations in Delphi Pascal

The loop is a common element in all programming languages. Object Pascal has three control structures that execute blocks of code repeatedly: for, repeat ... until and while ... do.

#### The FOR loop

Suppose we need to repeat an operation a fixed number of times.

```
// show 1,2,3,4,5 message boxes
```

```
var j: integer;
```

```
begin
```

```
  for j := 1 to 5 do
```

```
  begin
```

```
    ShowMessage('Box: '+IntToStr(j)) ;
```

```
  end;
```

```
end;
```

The value of a control variable (j), which is really just a counter, determines how many times a for statement runs. The keyword for sets up a counter. In the preceding example, the starting value for the counter is set to 1.

The ending value is set to 5.

When the for statement begins running the counter variable is set to the starting value. Delphi then checks whether the value for the counter is less than the ending value. If the value is greater, nothing is done (program execution jumps to the line of code immediately following the for loop code block). If the starting value is less than the ending value, the body of the loop is executed (here: the message box is displayed). Finally, Delphi adds 1 to the counter and starts the process again.

Sometimes it is necessary to count backward. The **downto** keyword specifies that the value of a counter should be decremented by one each time the loop executes (it is not possible to specify an increment / decrement other than one). An example of a for loop that counts backward.

```
var j: integer;
begin
  for j := 5 downto 1 do
    begin
      ShowMessage('T minus ' + IntToStr(j) + 'seconds') ;
    end;
    ShowMessage('For sequence executed!') ;
  end;
```

Note: it's important that you never change the value of the control variable in the middle of the loop. Doing so will cause errors.

### **Nested FOR loops**

Writing a for loop within another for loop (nesting loops) is very useful when you want to fill / display data in a table or a grid.

```
var k,j: integer;
begin
  //this double loop is executed 4x4=16 times
  for k:= 1 to 4 do
    for j:= 4 downto 1 do
      ShowMessage('Box: ' + IntToStr(k)+ ',' + IntToStr(j)) ;
    end;
```

The rule for nesting for-next loops is simple: the inner loop (j counter) must be completed before the next statement for the outer loop is encountered (k counter). We can have triply or quadruply nested loops, or even more.

**Note:** Generally, the begin and end keywords are not strictly required, as you can see. If begin and end are not used, the statement immediately following the for statement is considered the body of the loop.

### The FOR-IN loop

If you have Delphi 2005 or any newer version, you can use the "new" for-element-in-collection style iteration over containers. The following example demonstrates iteration over string expressions: for each char in string check if the character is either 'a' or 'e' or 'i'.

**const**

s = 'About Delphi Programming';

**var**

c : char;

**begin**

for c in s do

begin

if c in ['a','e','i'] then

begin

*// do something*

end;

end;

end;

### The WHILE and REPEAT loops

Sometimes we won't know exactly how many times a loop should cycle. What if we want to repeat an operation until we reach a specific goal?

The most important difference between the while-do loop and the repeat-until loop is that the code of the repeat statement is always executed at least once.



The general pattern when we write a repeat (and while) type of loop in Delphi is as follows:

```
repeat
begin
    statements;
end;
until condition = true

while condition = true do
begin
    statements;
end;
```

Here is the code to show 5 successive message boxes using repeat-until:

```
var
    j: integer;
begin
    j:=0;
    repeat
        begin
            j := j + 1;
            ShowMessage('Box:'+IntToStr(j)) ;
        end;
    until j > 5;
end;
```

As you can see, the repeat statement evaluates a condition at the end of the loop (therefore repeat loop is executed for sure at least once).

The while statement, on the other hand, evaluates a condition at the beginning of the loop. Since the test is being done at the top, we will usually need to make sure that the condition makes sense before the loop is processed, if this is not true the compiler may decide to remove the loop from the code.

```
var j: integer;
begin
```

```

j:=0;
while j < 5 do
begin
  j:=j+1;
  ShowMessage('Box:'+IntToStr(j)) ;
end;
end;

```

## Break and Continue

The Break and Continue procedures can be used to control the flow of repetitive statements: The Break procedure causes the flow of control to exit a for, while, or repeat statement and continue at the next statement following the loop statement. Continue allows the flow of control to proceed to the next iteration of repeating operation.

## Understanding Typed Constants in Delphi

### How to implement persistent values between function calls.

When Delphi invokes an event handler, the old values of local variables are wiped out. What if we want to keep track of how many times a button has been clicked? We could have the values persist by using a unit-level variable, but it is generally a good idea to reserve unit-level variables only for sharing information. What we need are usually called static variables or typed constants in Delphi.

### Variable or constant?

Typed constants can be compared to initialized variables-variables whose values are defined on entry to their block (usually event handler). Such a variable is initialized only when the program starts running. After that, the value of a typed constant persists between successive calls to their procedures.

Using typed constants is a very clean way of implementing automatically initialized variables.

To implement these variables without typed constants, we'll need to create an initialization section that sets the value of each initialized variable.

### Variable typed constants

Although we declare typed constants in the const section of a procedure, it is important to remember that they are not constants. At any point in your application, if you have access to the identifier for a typed constant you'll be able to modify its value.

To see typed constants at work, put a button on a blank form, and assign the following code to the OnClick event handler:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
const  
    clicks : Integer = 1; //not a true constant  
begin  
    Form1.Caption := IntToStr(clicks) ;  
    clicks := clicks + 1;  
end;
```

Notice that every time you click on the button, forms caption increments steadily.

Now try the following code:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
var  
    clicks : Integer;  
begin  
    Form1.Caption := IntToStr(clicks) ;  
    clicks := clicks + 1;  
end;
```

We are now using uninitialized variable for the clicks counter. Notice that weird value in the forms caption after you click on the button.



## Constant typed constants

You have to agree that idea of modifiable constants sound a bit strange. In 32 bit versions of Delphi Borland decided to discourage their use, but support them for Delphi 1 legacy code.

We can enable or disable Assignable typed constants on the Compiler page of the Project Options dialog box.

If you've disabled Assignable typed constants for a given project, when you attempt to compile previous code Delphi will give you 'Left side cannot be assigned to' error upon compilation. You can, however, create assignable typed constant by declaring:

```
{SJ+}
```

```
const clicks : Integer = 1;
```

```
{SJ-}
```

Therefore, the first example code looks like:

```
procedure TForm1.Button1Click(Sender: TObject) ;
```

```
const
```

```
{SJ+}
```

```
    clicks : Integer = 1; //not a true constant
```

```
{SJ-}
```

```
begin
```

```
    Form1.Caption := IntToStr(clicks) ;
```

```
    clicks := clicks + 1;
```

```
end;
```

## **Running Delphi Applications With Parameters**

**How to pass command-line parameters to your application and how to handle them.**

In the days of DOS it was a common practice run applications (command line programs) with some kind of parameters that will specify what we want to do. Even now, in the world of Windows, we can go to MS-Dos prompt and run MS-DOS based program like DIR /?. That '/' after program name (DIR) will give us some help regarding the usage of the DIR command.

In this article, we will find out how to respond to command line parameters passed to a Delphi application.

### **Parameters**

We can pass the parameter from the command line in Windows or from the development environment in Delphi under Run-Parameters menu option.

We will use Parameters dialog box to pass command-line parameters to an application when we run it (for testing purposes - from within Delphi), just as if we were running the application from the Windows Explorer.

### **ParamCount, ParamStr()**

Simply put, the ParamCount function returns the number of parameters passed to the program on the command line, and ParamStr returns a specified parameter from the command-line.

While application is running, the parameters are available to us so we can retrieve them within a specific section of the application (usually from the OnActivate event handler of the main form).

Note: In a program, the CmdLine variable contains a string with command-line arguments specified when the application was started. We can use CmdLine to access the entire paramstring passed to an application.

We'll start with a simple application. Start up a new project and place a Button component on Form. In the button's OnClick event handler, write the following code:

#### **Procedure**

```
TForm1.Button1Click(Sender: TObject) ;  
begin  
  ShowMessage(ParamStr(0)) ;  
end;
```

When you run the program and click the button, a message box appears with the path and file name of the executing program.

We can see, that even if we haven't passed any parameters to our application ParamStr function "works", the reason is that the array value 0 stores the file name of the executable application including path information.

Now, choose Parameters from the Run Menu and add 'Delphi Programming' to the drop down list (without apostrophes).

Note: when you pass parameters to your application separate them with spaces or tabs. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

We will be looping through the amount of parameters using ParamCount() to get the value of parameters passed, with ParamStr(i).

Change the button's OnClick event handler to:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
var  
  j:integer;  
begin  
  for j := 1 to ParamCount do  
    ShowMessage(ParamStr(j)) ;  
end;
```



When you run the program and click the button, a message box appears displaying 'Delphi' (first parameter) and 'Programming' (second parameter).

Note: Working with parameters passed to the console mode application is the same.

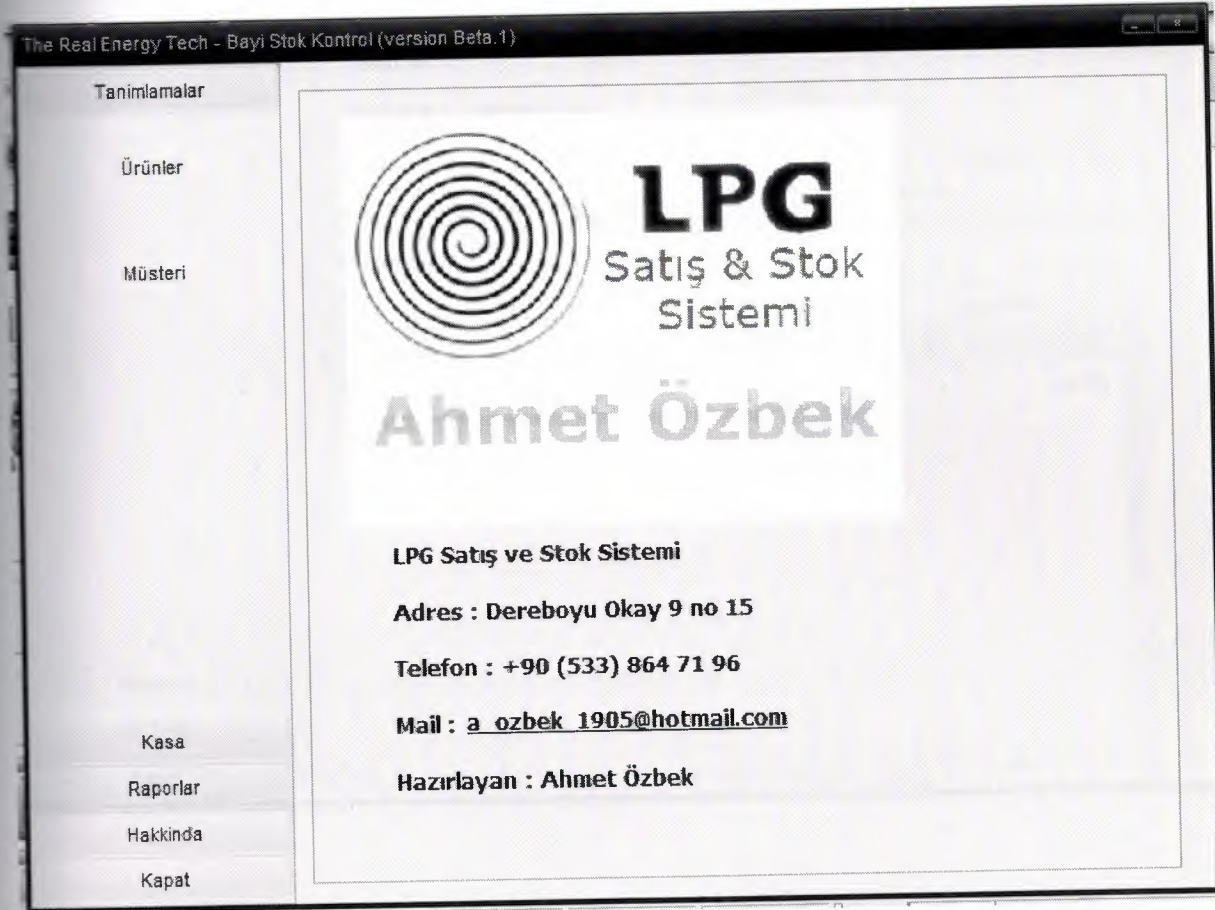
That's it, simple as only Delphi can be!

## CHAPTER THREE

### DEVELOPMENT OF LPG DISTRIBUTOR STOCK & SALE SYSTEM

The software development of LPG distributor stock & sale system is considered. The system is designed using delphi programme.

#### Main Menu



After we run the program main window welcomes us. In this window there are some menu items on the left.

This buttons helps us to navigate the system easily.

## Products Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Ürünler Müsteriler

Ürünler

Müşteri

Yeni

Düzenle

Sil

Kaydet

İptal

Arama

Ürün No

Ürün No : 1

Ürün Adı : Gapgaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Birim Fiyat : 35,00 YTL

Stok Miktarı : 990

Ürün Adı	Açıklama	Birim Fiyat	Stok Miktar
Gapgaz 25 Kg	Ev Tüpü 25 kg	35	990
LPG 1Lt	Litre ile Satış	1	100000

Kasa

Raporlar

Hakkında

Kapat

After we clicked the “products” button an inline window appears on the content panel. In this window we can record the stocks with all the details and we can search products.



## Customers Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Ürünler Müsteriler

Ürünler

Müşteri

Kasa

Raporlar

Hakkında

Kapat

Müşteri No : 1

Firma Adı : Melek A.Ş.

Müşteri Adı Soyadı : Burak Melek

Telefon : 5338662213

Faks :

Adres : fhkjshfhkjshfsjhfsjfhdsjft

Açıklama :

Yeni

Düzenle

Sil

Kaydet

İptal

Arama Firma Adı

Yetkili Adı	Firma Adı	Telefon	Faks
► Burak Melek	Melek A.Ş.	5338662213	

İşlemler

To collect the orders properly we need to record the customers. So we can record whom we sold the products.

## Orders Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Kasa

Alis

Satis

Raporlar

Hakkında

Kapat

Alis Satis

Arama

Ürün No : 1

Ürün No

Ürün Adı : Gaggaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Müşteri Adı (Firma)

Melek A.Ş

Birim Fiyat : 35

Stok Miktarı : 990

Onayla

Miktar

100

Ödeme Türü

Kredi Kartı

Onayla

Birim Fiyat

48,00 YTL

Açıklama

iptal

4.800,00 YTL

Önceki işlem

Ödeme Tarihi

14.08.2007

Durum

Henüz Ödenmedi

Alacak Kaydet

iptal

Yazdır

In this window we record the orders. First we choose the product by searching with any information and select customer from the combo box. And then we click the "confirm" button. Now we can enter the quantity, unit price and payment type. Then we click the confirm button twice. After all we select the payment date and payment situation.



## Purchase Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Kasa

Alis

Satis

Raporlar

Hakkında

Kapat

Alis Satis

Arama

Ürün No

Ürün No : 1

Ürün Adı : Gaggaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Birim Fiyat : 35

Stok Miktarı : 990

Onayla

Miktar

1

Birim Fiyat

42,00 YTL

42,00 YTL

Ödeme Türü

Çek

Açıklama

Onayla

iptal

Önceki işlem

Ödeme Tarihi

13.08.2007

Durum

Henüz Ödenmedi

Borc Kaydet

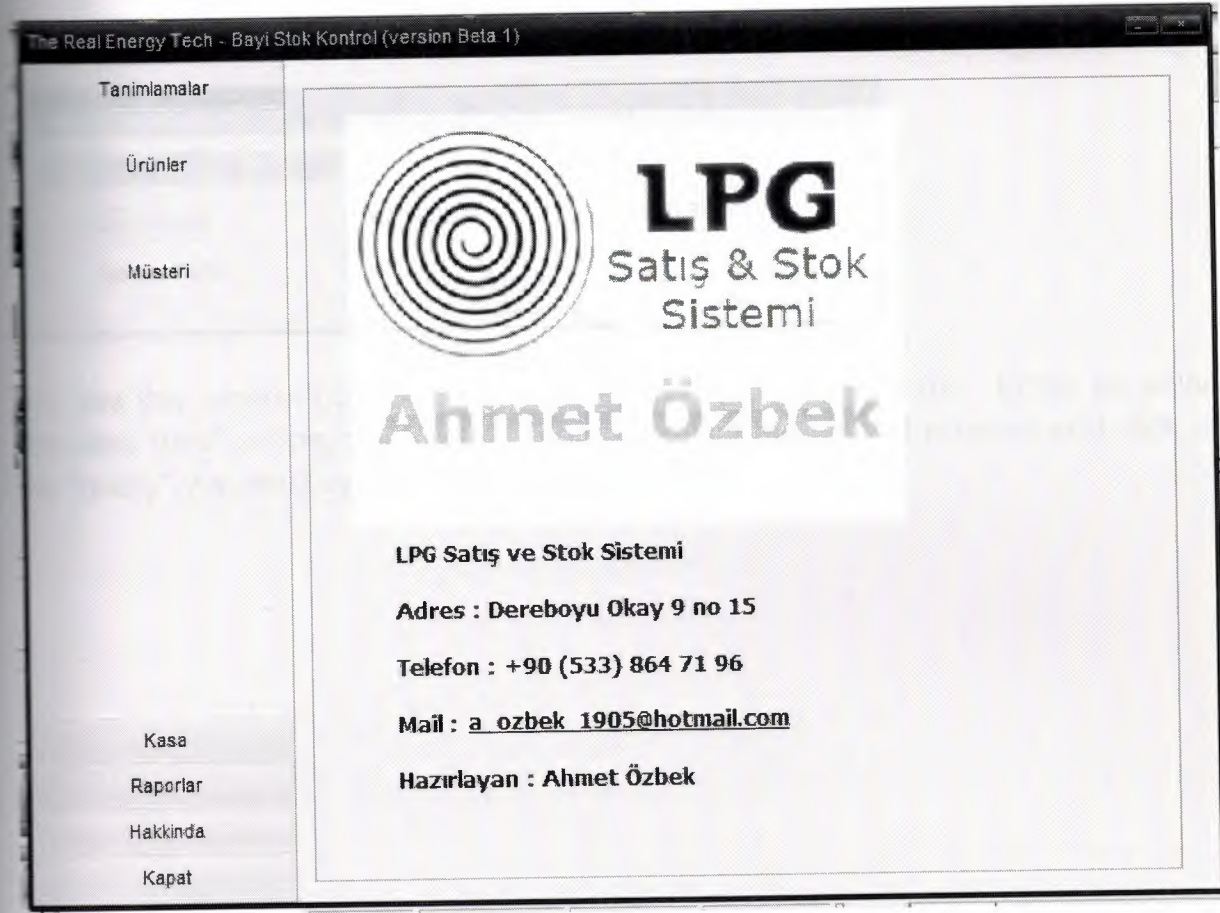
iptal

Yazdır

In this window we record the purchases. First we choose the product by searching with any information. And then we click the "confirm" button. Now we can enter the quantity, unit price and payment type. Then we click the confirm button twice. After all we select the payment date and payment situation.

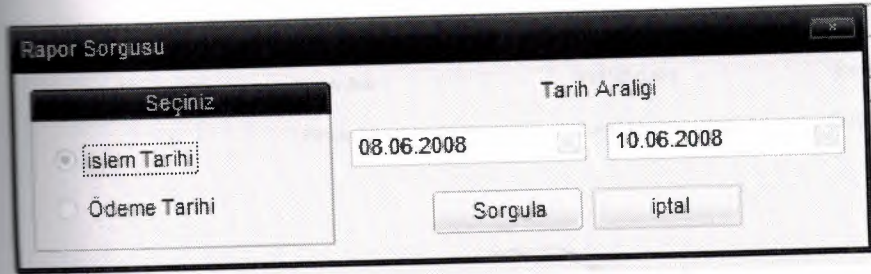


## About window



About window has the same view of the main window. Here we can see some texts about the program and programmer.

## Query window



Rapor Sorgusu

Seçiniz

☒ İşlem Tarihi

☐ Ödeme Tarihi

Tarih Aralığı

08.06.2008 10.06.2008

Sorgula İptal

We use this window both on “purchase reports” and “sale reports”. Either we select “process date” or “payment date”. Then we select the date of process and click on the “query”. As result system generates the report.

## Customers List

Müsteri Listesi				
Musteri No :	Firma Adi :	Yetkili Adi :	Telefon :	Faks :
1	Melek A.Ö	Burak Melek	5338662213	

The list of customers.



## Stocks List

Stok Listesi					
Stok No :	Stok Adi :	Aciklama :	Birim Fiyat (YTL) : Stok Miktari :		
1	Gapgaz 25 Kg	Ev Tüpü 25 kg	35	990	
2	LPG 1Lt	Litre ile Satıyb	1	100000	

## The list of Stocks

## CONCLUSION

TECHNOLOGY HAS AFFECTED THE REFERENCE and information culture in libraries. With the increasing scope of information transfer, users have higher service expectations of library and information science professionals. The emergence of a digital information environment has changed the century-old role of the reference professional. After the rise of the Internet, many skeptics foresaw the end of a need for librarians, particularly those working in traditional positions such as reference. Nevertheless, data from the Bureau of Labor Statistics indicates an increase in the number of information professionals by the year 2008. Reference professionals are becoming more--not less--essential. Graduate programs must examine the curriculum for reference and information access professionals. Greater access to information sources by users has highlighted the need for reference and information professionals to develop new skills including more technological knowledge, a better understanding of user information-seeking, new instructional techniques, and better communication skills. In addition to live classroom instruction, most schools offer reference and information access courses to a more diverse student body by employing distance-learning technologies.

## REFERENCES

- [1] Mastering Borland Delphi 2005 (Mastering) by Marco Cantu' (Paperback - Aug 19, 2005)
- [2] Inside Delphi 2006 (Wordware Delphi Developer's Library) by Ivan Hladni (Paperback - Nov 25, 2005)
- [3] Introducing Delphi Programming: Theory through Practice by John Barrow, Linda Miller, Katherine Malan, and Helene Gelderblom
- [4] [www.delphiturk.com](http://www.delphiturk.com)
- [5] <http://www.torry.net>
- [6] [www.about.com](http://www.about.com)



## APPENDIX

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, bsSkinData, BusinessSkinForm, bsSkinCtrls, bsSkinGrids,  
bsDBGrids, DB, ADODB, ComCtrls, bsSkinTabs, StdCtrls, Mask,  
bsSkinBoxCtrls, bsdbctrls, bsMessages, OleCtrls,  
ShockwaveFlashObjects\_TLB, ExtCtrls, jpeg;

type

TForm1 = class(TForm)

bsBusinessSkinForm1: TbsBusinessSkinForm;

bsSkinData1: TbsSkinData;

bsCompressedStoredSkin1: TbsCompressedStoredSkin;

bsSkinButtonsBar1: TbsSkinButtonsBar;  
  
bsSkinPageControl1: TbsSkinPageControl;  
  
bsSkinTabSheet1: TbsSkinTabSheet;  
  
bsSkinTabSheet2: TbsSkinTabSheet;  
  
ADOConnection1: TADOConnection;  
  
ADOTable1: TADOTable;  
  
DataSource1: TDataSource;  
  
bsSkinDBGrid1: TbsSkinDBGrid;  
  
bsSkinScrollBar1: TbsSkinScrollBar;  
  
bsSkinPanel1: TbsSkinPanel;  
  
bsSkinTextLabel1: TbsSkinTextLabel;  
  
bsSkinDBText1: TbsSkinDBText;  
  
bsSkinDBEdit1: TbsSkinDBEdit;  
  
bsSkinDBEdit2: TbsSkinDBEdit;  
  
bsSkinDBCurrencyEdit1: TbsSkinDBCurrencyEdit;  
  
bsSkinDBSpinEdit1: TbsSkinDBSpinEdit;  
  
bsSkinButton1: TbsSkinButton;  
  
bsSkinButton2: TbsSkinButton;  
  
bsSkinButton3: TbsSkinButton;  
  
bsSkinButton4: TbsSkinButton;

bsSkinButton5: TbsSkinButton;  
  
bsSkinStdLabel1: TbsSkinStdLabel;  
  
bsSkinComboBox1: TbsSkinComboBox;  
  
bsSkinEdit1: TbsSkinEdit;  
  
bsSkinPanel2: TbsSkinPanel;  
  
bsSkinTextLabel2: TbsSkinTextLabel;  
  
bsSkinDBText2: TbsSkinDBText;  
  
bsSkinDBEdit3: TbsSkinDBEdit;  
  
bsSkinDBEdit4: TbsSkinDBEdit;  
  
bsSkinButton6: TbsSkinButton;  
  
bsSkinButton8: TbsSkinButton;  
  
bsSkinButton9: TbsSkinButton;  
  
bsSkinButton10: TbsSkinButton;  
  
bsSkinComboBox2: TbsSkinComboBox;  
  
bsSkinStdLabel2: TbsSkinStdLabel;  
  
bsSkinEdit2: TbsSkinEdit;  
  
bsSkinDBGrid2: TbsSkinDBGrid;  
  
bsSkinMessage1: TbsSkinMessage;  
  
bsSkinButton7: TbsSkinButton;  
  
ADOTable2: TADOTable;



DataSource2: TDataSource;

bsSkinStdLabel3: TbsSkinStdLabel;

bsSkinDBMemo1: TbsSkinDBMemo;

bsSkinStdLabel4: TbsSkinStdLabel;

bsSkinDBMemo2: TbsSkinDBMemo;

bsSkinDBEdit5: TbsSkinDBEdit;

bsSkinDBEdit6: TbsSkinDBEdit;

bsSkinPageControl2: TbsSkinPageControl;

bsSkinTabSheet3: TbsSkinTabSheet;

bsSkinTabSheet4: TbsSkinTabSheet;

bsSkinPanel3: TbsSkinPanel;

bsSkinPanel4: TbsSkinPanel;

bsSkinPanel5: TbsSkinPanel;

bsSkinTextLabel3: TbsSkinTextLabel;

bsSkinStdLabel5: TbsSkinStdLabel;

bsSkinComboBox3: TbsSkinComboBox;

bsSkinEdit3: TbsSkinEdit;

bsSkinDBText3: TbsSkinDBText;

bsSkinDBText4: TbsSkinDBText;

bsSkinDBText5: TbsSkinDBText;

bsSkinDBText6: TbsSkinDBText;  
  
bsSkinDBText7: TbsSkinDBText;  
  
bsSkinButton11: TbsSkinButton;  
  
ADOTable3: TADOTable;  
  
DataSource3: TDataSource;  
  
bsSkinDBSpinEdit2: TbsSkinDBSpinEdit;  
  
bsSkinStdLabel6: TbsSkinStdLabel;  
  
bsSkinDBCurrencyEdit2: TbsSkinDBCurrencyEdit;  
  
bsSkinStdLabel7: TbsSkinStdLabel;  
  
bsSkinDBCurrencyEdit3: TbsSkinDBCurrencyEdit;  
  
bsSkinDBComboBox1: TbsSkinDBComboBox;  
  
bsSkinStdLabel8: TbsSkinStdLabel;  
  
bsSkinStdLabel9: TbsSkinStdLabel;  
  
bsSkinDBMemo3: TbsSkinDBMemo;  
  
bsSkinButton12: TbsSkinButton;  
  
bsSkinButton13: TbsSkinButton;  
  
bsSkinButton14: TbsSkinButton;  
  
bsSkinButton15: TbsSkinButton;  
  
ADOTable4: TADOTable;  
  
DataSource4: TDataSource;

bsSkinDBDateEdit1: TbsSkinDBDateEdit;  
bsSkinStdLabel10: TbsSkinStdLabel;  
bsSkinDBComboBox2: TbsSkinDBComboBox;  
bsSkinStdLabel11: TbsSkinStdLabel;  
bsSkinButton16: TbsSkinButton;  
bsSkinButton17: TbsSkinButton;  
bsSkinPanel8: TbsSkinPanel;  
bsSkinStdLabel12: TbsSkinStdLabel;  
bsSkinStdLabel13: TbsSkinStdLabel;  
bsSkinButton18: TbsSkinButton;  
bsSkinDBDateEdit2: TbsSkinDBDateEdit;  
bsSkinDBComboBox3: TbsSkinDBComboBox;  
bsSkinButton19: TbsSkinButton;  
bsSkinButton20: TbsSkinButton;  
bsSkinPanel7: TbsSkinPanel;  
bsSkinStdLabel14: TbsSkinStdLabel;  
bsSkinStdLabel15: TbsSkinStdLabel;  
bsSkinStdLabel16: TbsSkinStdLabel;  
bsSkinStdLabel17: TbsSkinStdLabel;  
bsSkinDBSpinEdit3: TbsSkinDBSpinEdit;



bsSkinDBCurrencyEdit4: TbsSkinDBCurrencyEdit;  
bsSkinDBCurrencyEdit5: TbsSkinDBCurrencyEdit;  
bsSkinDBComboBox4: TbsSkinDBComboBox;  
bsSkinDBMemo4: TbsSkinDBMemo;  
bsSkinButton21: TbsSkinButton;  
bsSkinButton22: TbsSkinButton;  
bsSkinButton23: TbsSkinButton;  
bsSkinPanel6: TbsSkinPanel;  
bsSkinTextLabel4: TbsSkinTextLabel;  
bsSkinStdLabel18: TbsSkinStdLabel;  
bsSkinDBText8: TbsSkinDBText;  
bsSkinDBText9: TbsSkinDBText;  
bsSkinDBText10: TbsSkinDBText;  
bsSkinDBText11: TbsSkinDBText;  
bsSkinDBText12: TbsSkinDBText;  
bsSkinComboBox4: TbsSkinComboBox;  
bsSkinEdit4: TbsSkinEdit;  
bsSkinButton24: TbsSkinButton;  
ADOTable5: TADOTable;  
DataSource5: TDataSource;

DataSource6: TDataSource;

ADOTable6: TADOTable;

ADOTable7: TADOTable;

DataSource7: TDataSource;

bsSkinDBLookupComboBox1: TbsSkinDBLookupComboBox;

bsSkinStdLabel19: TbsSkinStdLabel;

bsSkinPanel9: TbsSkinPanel;

bsSkinTextLabel5: TbsSkinTextLabel;

bsSkinLinkLabel2: TbsSkinLinkLabel;

bsSkinButton25: TbsSkinButton;

Image1: TImage;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure bsSkinButton1Click(Sender: TObject);

procedure bsSkinButton2Click(Sender: TObject);

procedure bsSkinButton4Click(Sender: TObject);

procedure bsSkinButton5Click(Sender: TObject);

procedure bsSkinButton3Click(Sender: TObject);

procedure bsSkinComboBox1Change(Sender: TObject);

procedure bsSkinEdit1Change(Sender: TObject);

```
procedure bsSkinButton6Click(Sender: TObject);

procedure bsSkinButton7Click(Sender: TObject);

procedure bsSkinButton8Click(Sender: TObject);

procedure bsSkinButton9Click(Sender: TObject);

procedure bsSkinButton10Click(Sender: TObject);

procedure bsSkinComboBox2Change(Sender: TObject);

procedure bsSkinEdit2Change(Sender: TObject);

procedure bsSkinButtonsBar1Sections0Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections0Items1Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections1Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections1Items1Click(Sender: TObject);

procedure bsSkinComboBox3Change(Sender: TObject);

procedure bsSkinEdit3Change(Sender: TObject);

procedure bsSkinButton11Click(Sender: TObject);

procedure bsSkinDBSpinEdit2Change(Sender: TObject);

procedure bsSkinDBCurrencyEdit2Change(Sender: TObject);

procedure bsSkinButton12Click(Sender: TObject);

procedure bsSkinButton13Click(Sender: TObject);

procedure bsSkinButton14Click(Sender: TObject);

procedure bsSkinButton16Click(Sender: TObject);
```



```

procedure bsSkinButton17Click(Sender: TObject);

procedure bsSkinComboBox4Change(Sender: TObject);

procedure bsSkinEdit4Change(Sender: TObject);

procedure bsSkinButton24Click(Sender: TObject);

procedure bsSkinDBSpinEdit3Change(Sender: TObject);

procedure bsSkinButton21Click(Sender: TObject);

procedure bsSkinButton22Click(Sender: TObject);

procedure bsSkinButton23Click(Sender: TObject);

procedure bsSkinButton19Click(Sender: TObject);

procedure bsSkinDBCurrencyEdit4Change(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items3Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items2Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items1Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections3Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections4Click(Sender: TObject);

procedure bsSkinButton25Click(Sender: TObject);

private
{ Private declarations }

public

```

```

{ Public declarations }

end;

var
    Form1: TForm1;

implementation

uses Unit3, Unit4, Unit5, Unit2;

{$R *.dfm}

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if bsskinmessage1.MessageDlg('Programi kapatmak istediginize emin misiniz
    ?',mtconfirmation,[mbyes,mbno],0)=mryes then

        application.Terminate

    else

        application.Run;

        form2.hide;

end;

```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
tmpstr:string;
```

```
begin
```

```
  TmpStr:=ExtractFileDir(ParamStr(0));
```

```
  if TmpStr[Length(TmpStr)]<>'\' Then TmpStr:=TmpStr+'\';
```

```
  adoconnection1.Connected:=false;
```

```
  adoconnection1.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data  
  Source='+tmpstr+'db.mdb;Persist Security Info=False';
```

```
  adoconnection1.Connected:=true;
```

```
  adotable1.Active:=true;
```

```
  adotable2.Active:=true;
```

```
  adotable3.Active:=true;
```

```
  adotable4.Active:=true;
```

```
  adotable5.Active:=true;
```

```
  adotable6.Active:=true;
```

```
  adotable7.Active:=true;
```

```
  // stok grid duzenlemeler //
```

```
  bsskindbgrid1.Columns[0].Visible:=false;
```



```
bsskindbgrid1.Columns[1].Title.caption:='Ürün Adı';  
  
bsskindbgrid1.Columns[1].Width:=100;  
  
bsskindbgrid1.Columns[2].Title.caption:='Açıklama';  
  
bsskindbgrid1.Columns[2].Width:=150;  
  
bsskindbgrid1.Columns[3].Title.caption:='Birim Fiyat';  
  
bsskindbgrid1.Columns[3].Width:=90;  
  
bsskindbgrid1.Columns[4].Title.caption:='Stok Miktar';  
  
bsskindbgrid1.Columns[4].Width:=90;  
  
// muster grid duzenlemeler //  
  
bsskindbgrid2.Columns[0].Visible:=false;  
  
bsskindbgrid2.Columns[1].Title.caption:='Yetkili Adı';  
  
bsskindbgrid2.Columns[1].Width:=150;  
  
bsskindbgrid2.Columns[2].Title.caption:='Firma Adı';  
  
bsskindbgrid2.Columns[2].Width:=100;  
  
bsskindbgrid2.Columns[3].Title.caption:='Telefon';  
  
bsskindbgrid2.Columns[3].Width:=90;  
  
bsskindbgrid2.Columns[4].Title.caption:='Faks';  
  
bsskindbgrid2.Columns[4].Width:=90;  
  
bsskindbgrid2.Columns[5].Visible:=false;  
  
bsskindbgrid2.Columns[6].Visible:=false;
```

**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**LPG DISTRIBUTOR STOCK & SALE SYSTEM**

**Graduation Project**

**COM 400**

**Student : Ahmet ÖZBEK**

**Supervisor : Assoc. Prof. Rahib ABİYEV**

**Nicosia – 2008**

## ACKNOWLEDGEMENTS

*"Firstly, I would like to thank to my supervisor Mr Rahib ABIYEV, Elbrus IMANOV, Dr.Kaan UYAR, Okan DONANGİL, Ümit İLHAN for their great advise and recomendation for finishing my project properly also, teaching and guiding me in others lectures.*

*I am greatly indepted to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my brother Mahmut ÖZBEK and my uncle Ahmet Şerif ÇULCU did for me during my educational life, also I want to say thanks to my mother Saliha ÖZBEK. I dedicate my project to them.*

*I thank all the staff of the faculty of engineering for giving facilities to practise, teaching and solving problem in my complete undergraduation program*

*I thank my friends Burak MELEK and Kerim ALTANHAN for their help, they get tired with me, and they helped me and give morale everytime.*

*I thank them with my all*

*Finally, I promise to do my best in my life as an bachelor of engineer after finishing my undergraduate program"*



# **TABLE OF CONTENTS**

<b>ACKNOWLEDGEMENT</b>	i
<b>TABLE OF CONTENTS</b>	ii
<b>ABSTRACT</b>	iii
<b>INTRODUCTION</b>	1
 <b>CHAPTER ONE – MICROSOFT ACCESS DATABASE</b>	 2
1.1 Microsoft Access Database Fundamentals	2
1.2 Microsoft Access Reports Tutorial	5
1.3 Creating a Simple Query in Microsoft Access	11
1.4 Creating Forms in Microsoft Access	16
1.5 How do I encrypt an Access 2007 database?	18
<b>CHAPTER ONE – DELPHI PROGRAMMING LANGUAGE</b>	19
2.1 A brief history of Borland's Delphi	19
2.2 Delphi For Beginners:	21
2.3 A Glossary of Delphi Programming Technical Terms	23
2.4 Understanding Delphi Project Files (.DPR)	37
2.4.1 New: Delphi Project	37
2.4.2 Project File	37
2.4.3 Project Unit	38
2.4.4 An Example: Hide Main Form / Hide Taskbar Button	39
2.5 Understanding the Birth, Life and Death of a Delphi Form	40
2.6 Understanding and Using Functions and Procedures	43
2.7 Understanding and Using Loops	50
2.8 Understanding Typed Constants in Delphi	54
2.9 Running Delphi Applications With Parameters	57
<b>CHAPTER THREE – DEVELOPMENT OF LPG DISTRIBUTOR STOCK &amp; SALE SYSTEM</b>	60
<b>CONCLUSION</b>	69
<b>REFERENCES</b>	70
<b>APPENDIX</b>	71

## **ABSTRACT**

The aim of this project is to register petrol station program that contain registration, all applications and also customers, sales, stocks and lpg application. The program was prepared by using Delphi programming and using MS Access database.

This project consist of so many forms and menus. The main form of the arrive the others forms . Which are include information about the sales,stocks and customers.

I think to give this system to GAPGAZ A.Ş.

To show results, show the efficiency of the program of LPG sale and stock in program of the using in other chapters.

## INTRODUCTION

Information technology (IT), as defined by the Information Technology Association of America (ITAA), is "the study, design, development, implementation, support or management of computer-based information systems, particularly software applications and computer hardware." IT deals with the use of electronic computers and computer software to convert, store, protect, process, transmit and retrieve information, securely.

Recently it has become popular to broaden the term to explicitly include the field of electronic communication so that people tend to use the abbreviation ICT (Information and Communications Technology), it is common for this to be referred to as IT & T in the Australasia region, standing for Information Technology and *Telecommunications*.

Today, the term information technology has ballooned to encompass many aspects of computing and technology, and the term is more recognizable than ever before. The information technology umbrella can be quite large, covering many fields. IT professionals perform a variety of duties that range from installing applications to designing complex computer networks and information databases. A few of the duties that IT professionals perform may include data management, networking, engineering computer hardware, database and software design, as well as the management and administration of entire systems.

The aim of this project is to develop a simple Stock Management System for small companies. The project consists of introduction, three chapters and conclusion.

Chapter One; describes general terms of Microsoft Access Database and the processes of creating a database.

Chapter Two; describes the main lines of Borland Delphi Programming Language such as reserved words, simple codes, methods and basic events.

Chapter Three; is the User's Manual of the program that gives information about the system developed as Stock Management System.



## **Microsoft Access Database Fundamentals**

Are you overwhelmed by the large quantities of data that need to be tracked in your organization? Perhaps you're currently using a paper filing system, text documents or a spreadsheet to keep track of your critical information. If you're searching for a more flexible data management system, a database might be just the salvation you're looking for.

What is a database? Quite simply, it's an organized collection of data. A database management system (DBMS) such as Access, FileMaker Pro, Oracle or SQL Server provides you with the software tools you need to organize that data in a flexible manner. It includes facilities to add, modify or delete data from the database, ask questions (or queries) about the data stored in the database and produce reports summarizing selected contents.

Microsoft Access provides users with one of the simplest and most flexible DBMS solutions on the market today. Regular users of Microsoft products will enjoy the familiar Windows "look and feel" as well as the tight integration with other Microsoft Office family products. An abundance of wizards lessen the complexity of administrative tasks and the ever-present Microsoft Office Helper (you know... the paper clip!) is available for those who care to use it. Before purchasing Access, be sure that your system meets Microsoft's minimum system requirements. To further our discussion, let's first examine three of the major components of Access that most database users will encounter - tables, queries, forms. Once we've completed that we'll look at the added benefits of reports, web integration and SQL Server integration.

Tables comprise the fundamental building blocks of any database. If you're familiar with spreadsheets, you'll find database tables extremely similar.

	Employee ID	Last Name	First Name	Title	Title Of	Birth Date
*	1	Davolio	Nancy	Sales Representative	Ms.	08-Dec-1968
+	2	Fuller	Andrew	Vice President, Sales	Dr.	19-Feb-1962
+	3	Leverling	Janet	Sales Representative	Ms.	30-Aug-1963
+	4	Peacock	Margaret	Sales Representative	Mrs.	19-Sep-1958
+	5	Buchanan	Steven	Sales Manager	Mr.	04-Mar-1955
+	6	Suyama	Michael	Sales Representative	Mr.	02-Jul-1963
+	7	King	Robert	Sales Representative	Mr.	29-May-1960
+	8	Callahan	Laura	Inside Sales Coordinator	Ms.	09-Jan-1958
+	9	Dodsworth	Anne	Sales Representative	Ms.	02-Jul-1969

Record: 10 of 10

The table above contains the employee information for our organization -- characteristics like name, date of birth and title. Examine the construction of the table and you'll find that each column of the table corresponds to a specific employee characteristic (or **attribute** in database terms). Each row corresponds to one particular employee and contains his or her information. That's all there is to it! If it helps, think of each one of these tables as a spreadsheet-style listing of information.

Product	Unit Price	Quantity	Discount	Extended Price
Spegesild	\$12.00	2	25%	\$18.00
Chartreuse verte	\$18.00	21	25%	\$283.50
Rössle Sauerkraut	\$45.60	15	25%	\$513.00
*			0%	

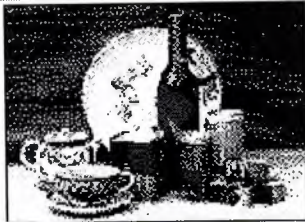
Subtotal: \$814.50  
Freight: \$29.46  
Total: \$843.96

Record: 1 of 830

Reports provide the capability to quickly produce attractively formatted summaries of the data contained in one or more tables and/or queries. Through the use of wizards, database users can create reports in literally a matter of minutes. As an example, let's return to our Northwind database. In this case, suppose that our company wishes to produce a catalog to share our product information with current and prospective clients. In previous sections, we learned that this sort of information



could be retrieved from our database through the judicious use of queries. However, recall that this information was presented in a tabular form -- not exactly the most attractive marketing material! Reports allow the inclusion of graphics, attractive formatting and pagination. Take a look at the sample report in the illustration below:

<b>Beverages</b>			
<i>Soft drinks, coffees, teas, beers, and ales</i>			
			
<b>Product Name:</b>	<b>Product ID:</b>	<b>Quantity Per Unit:</b>	<b>Unit Price:</b>
<b>Chai</b>	1	10 boxes x 20 bags	\$18.00
<b>Chang</b>	2	24 - 12 oz bottles	\$19.00
<b>Chartreuse verte</b>	39	750 cc per bottle	\$18.00
<b>Côte de Blaye</b>	38	12 - 75 cl bottles	\$263.50
<b>Guaraná Fantástica</b>	24	12 - 355 ml cans	\$4.50
<b>Ipoh Coffee</b>	43	16 - 500 g tins	\$46.00
<b>Lakkaikööri</b>	76	500 ml	\$18.00
<b>Laughing Lumberjack Lager</b>	67	24 - 12 oz bottles	\$14.00

Microsoft Access also provides native support for the World Wide Web. Posting data to the web is a breeze. If you have a formatted report that you would like to share with Internet or Intranet users, you can simply export it to an HTML file and publish it to your organization's web server. For those with more complex tastes, the advanced features of Access 2000 provide interactive data manipulation capabilities to web users.

Finally, no discussion of Microsoft Access is complete without mentioning its capability to tightly integrate with SQL Server, Microsoft's professional database server product. If you're in an organization that utilizes SQL Server, you'll be pleased to learn that you can retrieve, manipulate and work with the data stored on your organization's database server within the Microsoft Access environment. For more on this, view Microsoft's page on SQL Server/Office integration.



# Microsoft Access Reports Tutorial

## Part 1: Getting Started

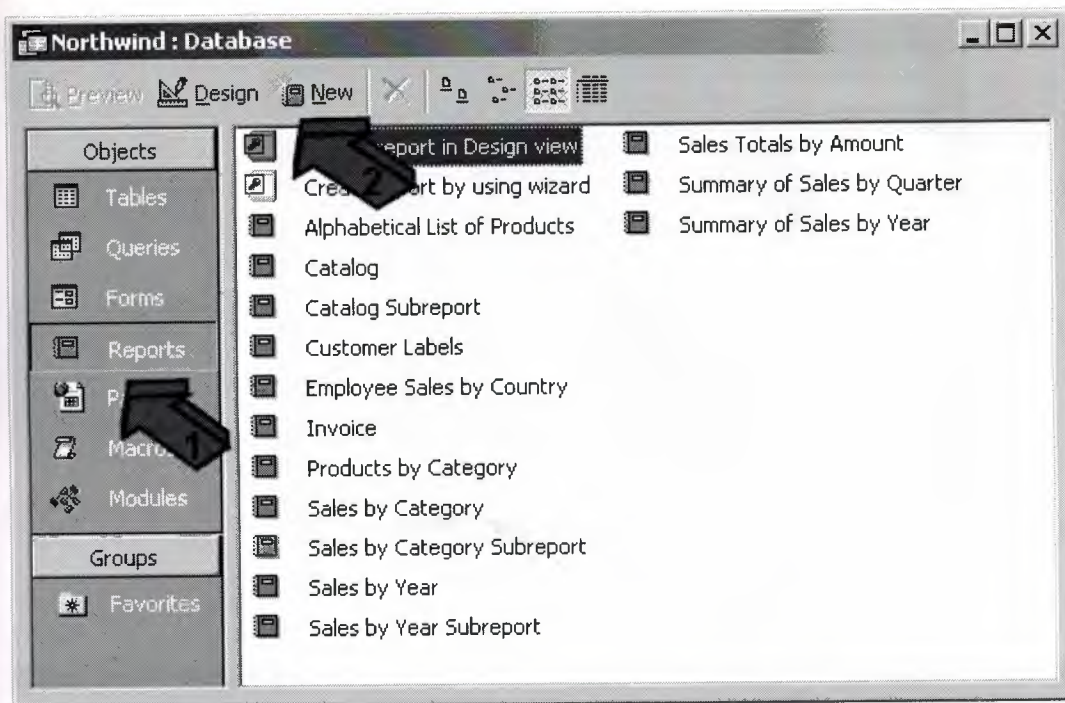
In our previous tutorials, you've learned a good deal about Microsoft Access. Together, we created a query, modified the query to make it more complex, and created a data entry form. We've learned the skills necessary to put information into a database and selectively remove the exact information we're seeking. In this tutorial, we're going to go a step further and learn how to create professionally formatted reports automatically from our database information. Returning to our familiar Northwind Company, we're going to design a nicely-formatted listing of employee home telephone numbers for the use of management.

The sample images in this tutorial were created using Access 2000. If you are running an earlier version of Access, your screen images may appear slightly different. However, the same general principles still apply and you should be able to follow along. If you need a quick-start on the basics of Access before getting started, take a look at the article "Microsoft Access Fundamentals."

Once again, we're going to use the Northwind sample database. Before we get started, open up Microsoft Access and then open the Northwind database. If you need help with this step, please read the article "How to Install the Northwind Sample Database."

**1. Choose the Reports menu.** Once you've opened Northwind, you'll be presented with the main database menu shown below. Go ahead and click on the "Reports" selection and you'll see a list of the various reports Microsoft included in the sample database. If you'd like, feel free to double-click on a few of these and get a feel for what reports look like and the various types of information that they contain.

**2. Create a new report.** After you've satisfied your curiosity, go ahead and click on the "New" button and we'll begin the process of creating a report from scratch.

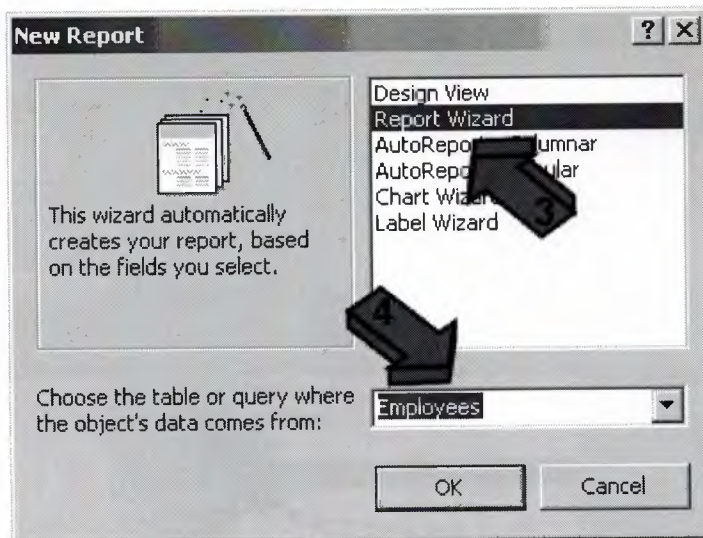


### Create a new report

3. **Select the Report Wizard.** The next screen that appears will ask you to select the method you wish to use to create the report. We're going to use the Report Wizard which will walk us through the creation process step-by-step. After you've mastered the wizard, you might want to return to this step and explore the flexibility provided by the other creation methods.

4. **Choose a table or query.** Before leaving this screen, we want to choose the source of data for our report. If you want to retrieve information from a single table, you can select it from the drop-down box below. Alternatively, for more complex reports, we can choose to base our report on the output of a query that we previously designed. For our example, all of the data we need is contained within the Employees table, so choose this table and click on OK.





### Select a creation method

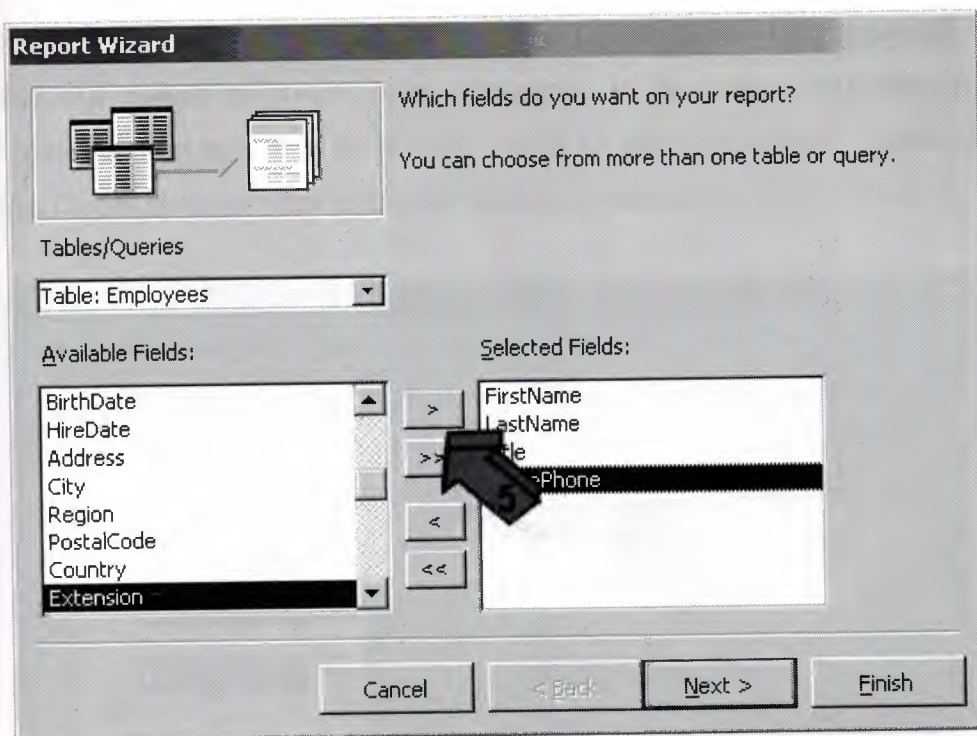
Next, we'll select exactly which table data to include in the report and learn how to apply formatting to our finished product. Read on!

## Microsoft Access Reports Tutorial Part 2: Selecting the Data

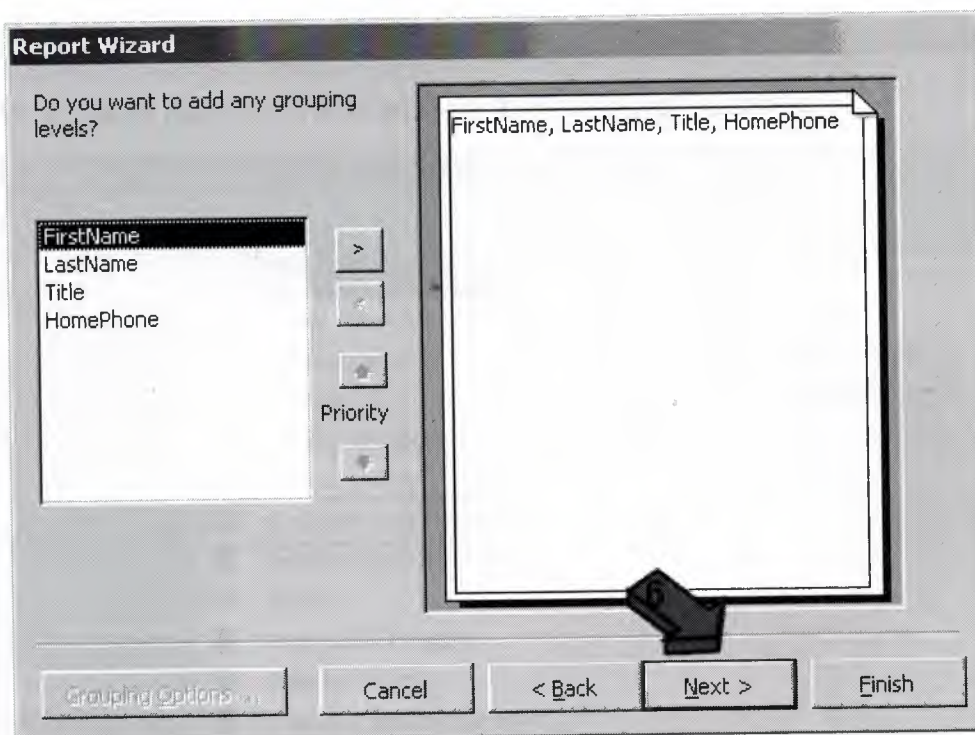
**5. Select the fields to include.** Use the '>' button to move over the desired fields. Note that the order you place the fields in the right column determines the default order they will appear in your report. Remember that we're creating an employee telephone directory for our senior management. Let's keep the information contained in it simple -- the first and last name of each employee, their title and their home telephone number. Go ahead and select these fields. When you are satisfied, click the Next button.

**6. Select the grouping levels.** At this stage, you can select one or more grouping levels to refine the order in which our report data is presented. For example, we may wish to break down our telephone directory by department so that all of the members of each department are listed separately. However, due to the small number of employees in our database, this is not necessary for our report. Go ahead and simply click on the Next button to bypass this step. You may wish to return here later and experiment with grouping levels.



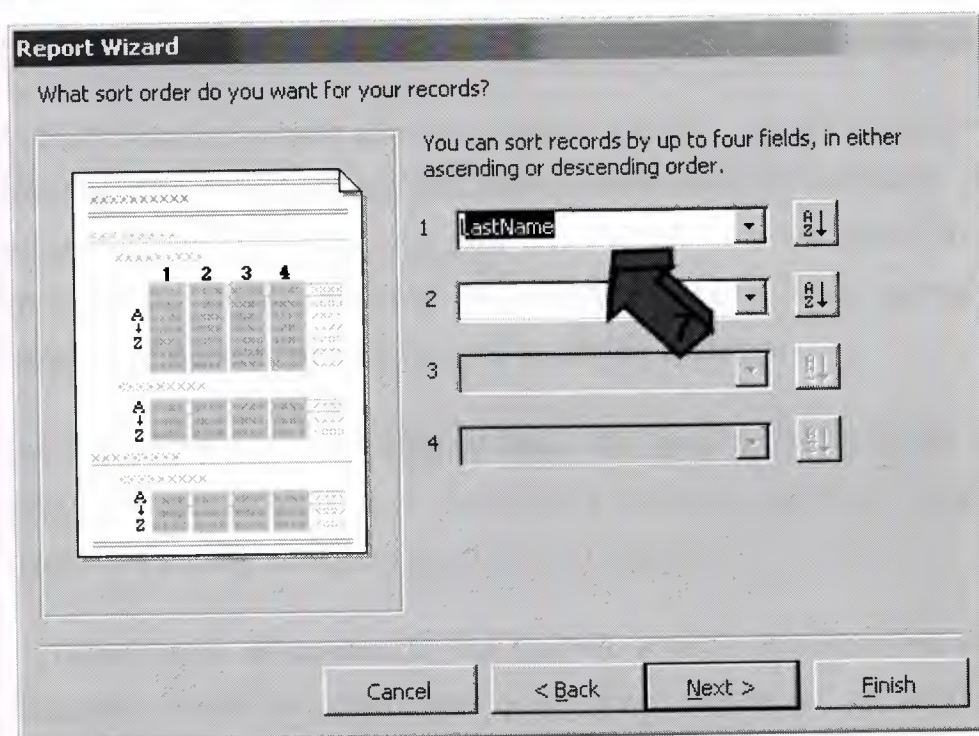


**Select the fields to include**



**Choose the grouping levels**

**7. Choose your sorting options.** In order to make reports useful, we often want to sort our results by one or more attributes. In the case of our telephone directory, the logical choice is to sort by the last name of each employee. Select this attribute from the first drop-down box and then click the Next button to continue.



**Choose the sorting options**

### Microsoft Access Reports Tutorial Part 3: Finishing Touches





## Creating a Simple Query in Microsoft Access

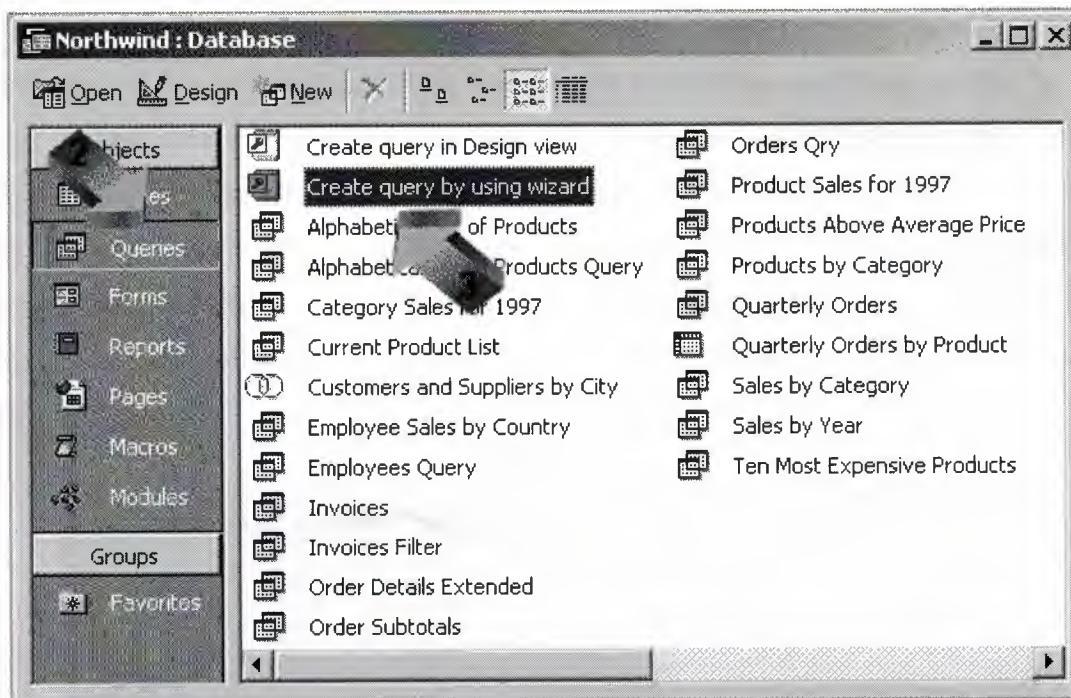
Have you ever wanted to combine information from multiple tables in your database in an efficient manner? Microsoft Access offers a powerful query function with an easy-to-learn interface that makes it a snap to extract exactly the information you need from your database. In this tutorial we'll explore the creation of a simple query.

In this example, as with all of our Access tutorials, we will use Access 2000 and the Northwind sample database included on the installation CD-ROM. If you're using an earlier version of Access, you may find that some of the menu choices and wizard screens are slightly different. However, the same basic principles apply to all versions of Access (as well as most database systems).

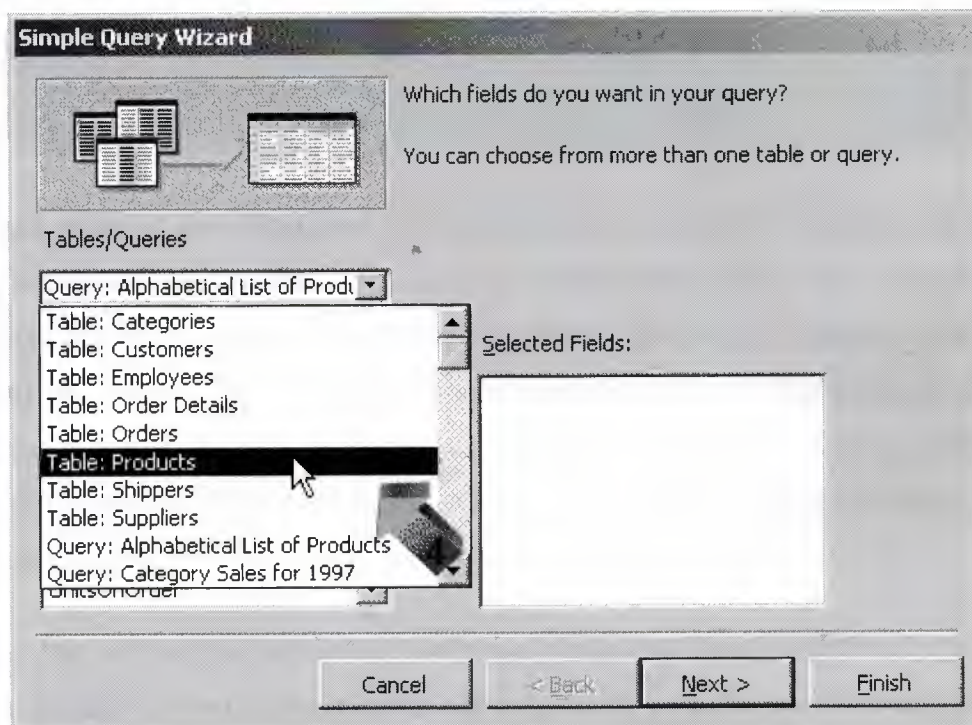
Let's explore the process step-by-step. Our goal in this tutorial is to create a query listing the names of all of our company's products, current inventory levels and the name and phone number of each product's supplier.

1. **Open your database.** If you haven't already installed the Northwind sample database, these instructions will assist you. Otherwise, go to the File tab, select Open and locate the Northwind database on your computer.
2. **Select the queries tab.** This will bring up a listing of the existing queries that Microsoft included in the sample database along with two options to create new queries.
3. **Double-click on "create query by using wizard".** The query wizard simplifies the creation of new queries. We'll use it in this tutorial to introduce the concept of query creation. In later tutorials we'll examine the Design view which facilitates the creation of more sophisticated queries.

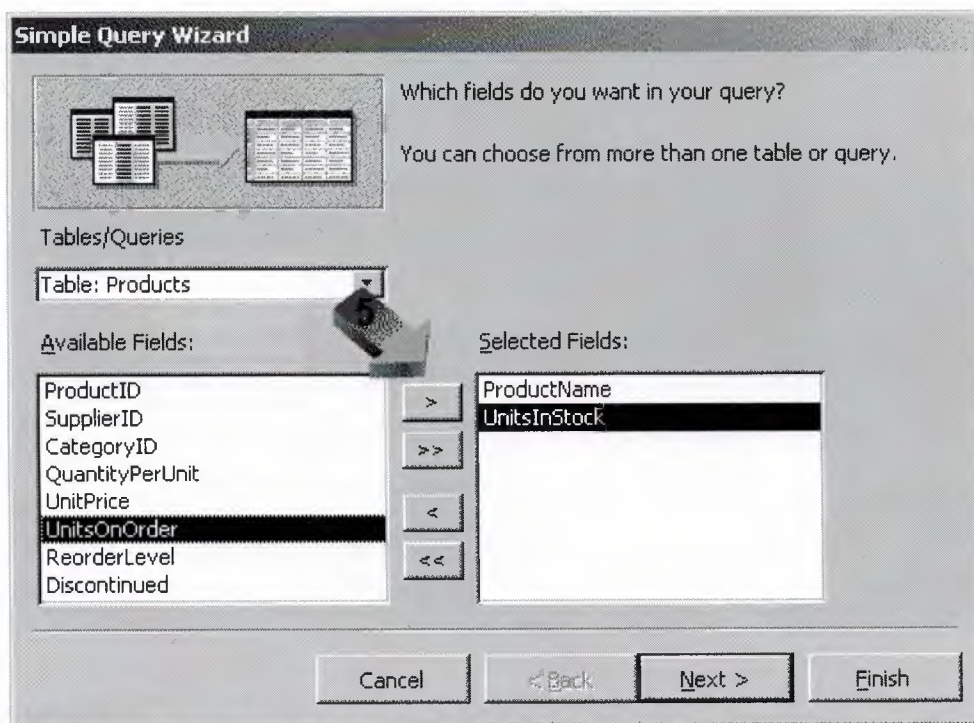




4. **Select the appropriate table from the pull-down menu.** When you select the pull-down menu, you'll be presented with a listing of all the tables and queries currently stored in your Access database. These are the valid data sources for your new query. In this example, we want to first select the Products table which contains information about the products we keep in our inventory.



5. **Choose the fields you wish to appear in the query results.** by either double-clicking on them or by single clicking first on the field name and then on the ">" icon. As you do this, the fields will move from the Available Fields listing to the Selected Fields listing. Notice that there are three other icons offered. The ">>" icon will select all available fields. The "<" icon allows you to remove the highlighted field from the Selected Fields list while the "<<" icon removes all selected fields. In this example, we want to select the ProductName, UnitsInStock, and UnitsOnOrder from the Product table.



6. **Repeat steps 4 and 5 to add information from additional tables, as desired.** In our example, we wanted to include information about the supplier. That information wasn't included in the Products table -- it's in the Suppliers table. Here's the power of a query! You can combine information from multiple tables and easily show relationships. In this example, we want to include the CompanyName and Phone fields from the Suppliers table. All you have to do is select the fields -- Access will line up the fields for you!

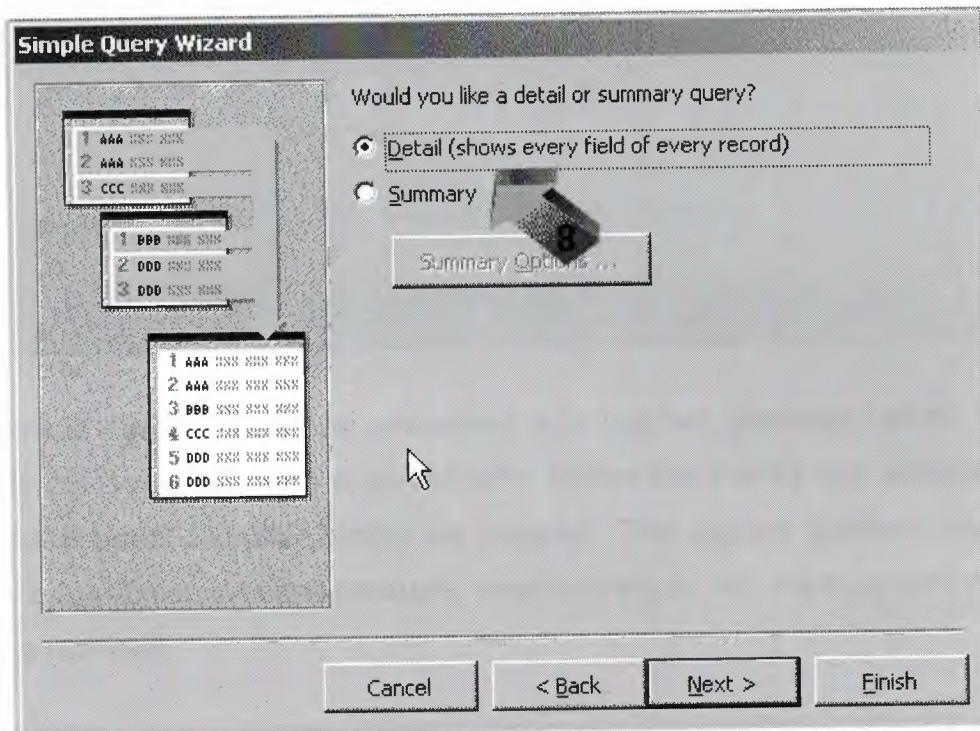
Note that this works because the Northwind database has predefined relationships between tables. If you're creating a new database, you'll need to establish these



relationships yourself. Read the article "Defining Relationships in Microsoft Access" for more information on this topic.

7. **Click on Next.**

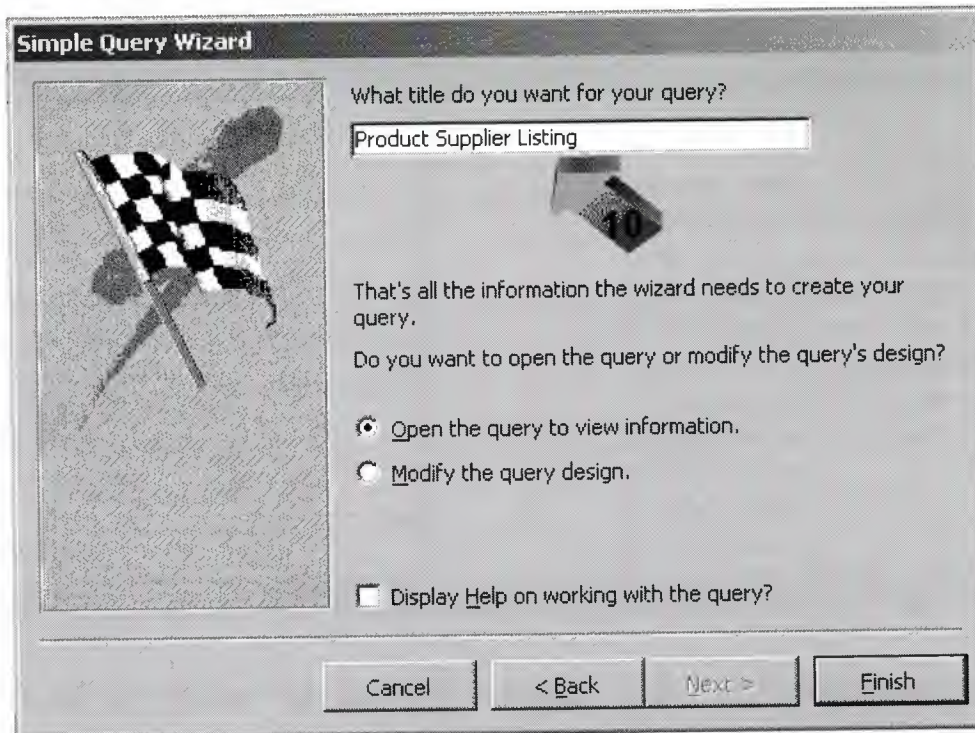
8. **Choose the type of results you would like to produce.** We want to produce a full listing of products and their suppliers, so choose the Detail option here.



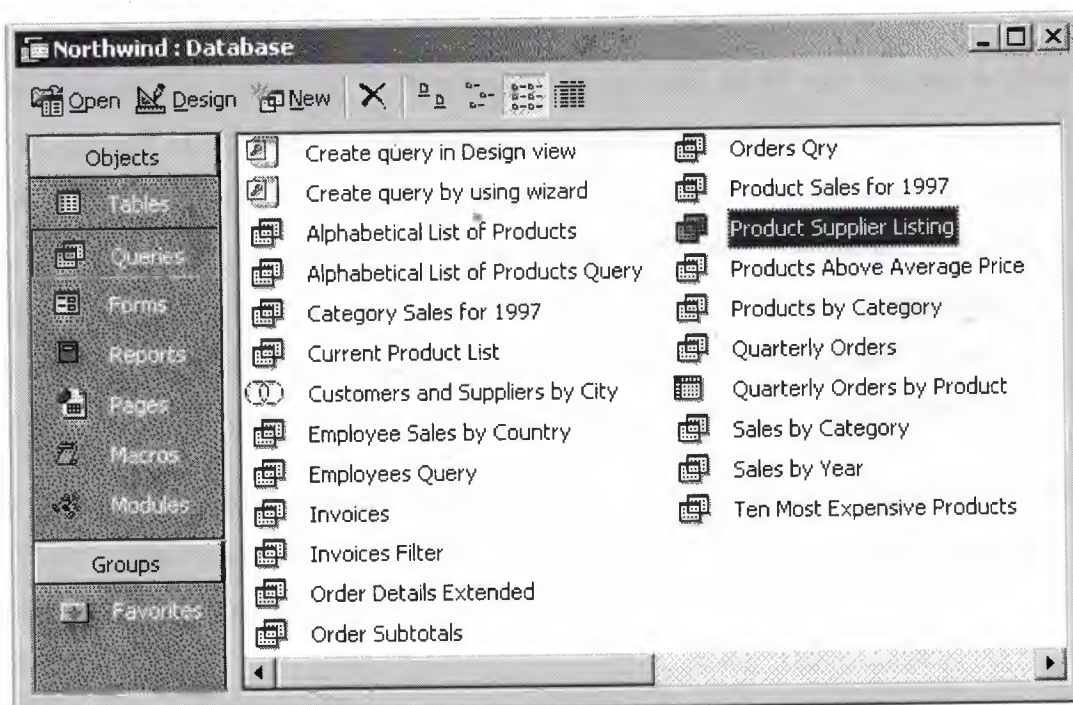
9. **Click on Next.**

10. **Give your query a title.** You're almost done! On the next screen you can give your query a title. Select something descriptive that will help you recognize this query later. We'll call this query "Product Supplier Listing."





11. **Click on Finish.** You'll be presented with the two windows below. The first window is the Query tab that we started with. Notice that there's one additional listing now -- the Product Supplier Listing we created. The second window contains our results -- a list of our company products, inventory levels and the supplier's name and telephone number!



Product Supplier Listing : Select Query					
	Product Name	Units In Stock	Units On Order	Company Name	Phone
►	Chai	39	0	Exotic Liquids	(171) 555-2222
	Chang	17	40	Exotic Liquids	(171) 555-2222
	Aniseed Syrup	13	70	Exotic Liquids	(171) 555-2222
	Chef Anton's Cajun Seasoning	53	0	New Orleans Cajun Delights	(100) 555-4822
	Chef Anton's Gumbo Mix	0	0	New Orleans Cajun Delights	(100) 555-4822
	Louisiana Fiery Hot Peppers	76	0	New Orleans Cajun Delights	(100) 555-4822
	Louisiana Hot Spiced Okra	4	100	New Orleans Cajun Delights	(100) 555-4822
	Grandma's Boysenberry Sauce	120	0	Grandma Kelly's Homestead	(313) 555-5735
	Uncle Bob's Organic Dried Pears	15	0	Grandma Kelly's Homestead	(313) 555-5735
	Northwoods Cranberry Sauce	6	0	Grandma Kelly's Homestead	(313) 555-5735
	Mishi Kobe Niku	29	0	Tokyo Traders	(03) 3555-5011
	Ikura	31	0	Tokyo Traders	(03) 3555-5011
	Longlife Tofu	4	20	Tokyo Traders	(03) 3555-5011
Record: 1 of 77					

Congratulations! You've successfully created your first query using Microsoft Access! Now you're armed with a powerful tool to apply to your database needs.

## Creating Forms in Microsoft Access

### Open your database

Microsoft Access forms provide a quick and easy way to modify and insert records into your databases. They offer an intuitive, graphical environment easily navigated by anyone familiar with standard computer techniques. Creating a form is a quite simple, pleasant experience. In this example, as with all of our Access tutorials, we will use Access 2003 and the Northwind sample database included on the installation CD-ROM. If you're using an earlier version of Access, you may find that some of the menu choices and wizard screens are slightly different. However, the same basic principles apply to all versions of Access (as well as most database systems). Let's begin! Our goal for this tutorial is to create a simple form that will allow data entry operators in our company to easily add new customers to our sales database.

If you haven't already installed the Northwind sample database, these instructions will assist you. Otherwise, go to the Help menu, then choose Sample Databases and Northwind Sample Databases.



**Click on the Forms tab under Objects**

This will bring up a list of the form objects currently stored in your database. Notice that there are a large number of pre-defined forms in this sample database. After you complete this tutorial, you might want to return to this screen and explore some of the more advanced features included in these forms.

**Click on the New icon to create a new form**

Click on the New icon to create a new form

**Select the creation method you wish to use**

Next, we're presented with a variety of different methods we can use to create a form. The AutoForm options quickly create a form based upon a table or query. Design View allows for the creation and formatting of elaborate forms using Access' form editing interface. The Chart Wizard and PivotTable Wizard create forms revolving around those two Microsoft formats. In this tutorial, we'll use the Form Wizard to walk through the process step-by-step.

**Select the data source and click OK.**

You can choose from any of the queries and tables in your database. If you recall our scenario, we wish to create a form to facilitate the addition of customers to our database. In order to accomplish this, we're going to select the Customers table from the pull-down menu.

**Select the form fields to be used and click Next.**

Next, you'll be presented with the screen below. Use this form to select the table/query fields you wish to appear on your form. To add fields one at a time, either double-click the field name or single-click the field name and single click the ">" button. To add all the fields at once, simply click the ">>" button. The "<" and "<<" buttons work in a similar manner to remove fields from the form. For our example, we will add all of the table's fields to the form.

**Select the form layout and click Next**

You can choose from either a columnar, tabular, datasheet or justified form layout. We'll use the justified layout to produce an organized form with a clean layout. You may wish to come back to this step later and explore the various layouts available.



**Select the form style and click Next.**

Microsoft Access includes a number of built-in styles to give your forms an attractive appearance. Click on each of the style names to see a preview of your form and choose the one you find most appealing.

**Provide a title for your form**

Select something easily recognizable -- this is how your form will appear in the database menu. Let's call our form "Customers" in this case. Select the next action and click Finish. You may open the form as a user will see it and begin viewing, modifying and/or entering new data. Alternatively, you may open the form in design view to make modifications to the form's appearance and properties. Let's do the latter and explore some of the options available to us.

**Edit Properties**

Click the Properties icon. This will bring up a menu of user-definable attributes that apply to our form. Edit the properties as necessary. Recall that our original goal was to create a form for data entry purposes. Most likely, we don't want to grant data entry employees full access to view or edit customer records. Setting the "Data Entry" property to Yes will only allow users to insert new records and modify records created during that session.

**Encrypting an Access Database**

Security-conscious database users have long called for the ability to use strong encryption in Microsoft Access. With the release of Access 2007, Microsoft answered these pleas and introduced a robust encryption feature that allows for the simple addition of a great deal of security to Access databases.

**What is encryption?**

Encryption provides you with the ability to protect your database file from prying eyes. It transforms the way data is stored on your disk so that individuals who do not know the database password can not open the database or use other techniques to view the file contents. Security professionals recommend the use of encryption to protect sensitive information.

## **How do I encrypt an Access 2007 database?**

Access 2007 users may encrypt databases stored in ACCDB format by password-protecting them.

Note that this feature is not available for database stored in the older MDB format. You may find the following articles useful when attempting to encrypt an Access database:

- Password-protecting an Access 2007 Database, Step-by-Step
- ACCDB Database Format
- Convert older Access databases to Access 2007

## **How do I decrypt an Access 2007 database?**

If you want to open an encrypted database for use and then reencrypt it when you are finished, Microsoft Access will handle the mechanics for you. Simply open the database as you normally would and enter the database password when prompted. Access will decrypt the database for your use and then save a new encrypted copy when you make changes.

If you want to remove encryption from an encrypted Access database, open the database in exclusive mode and then click "Decrypt Database" in the Database Tools group.

## **What type of encryption does Access 2007 use?**

Access 2007 uses the Microsoft Cryptographic API. This means that it will support any cryptographic algorithm available within Windows as a Cryptographic Service Provider (CSP). This is a great improvement over earlier versions of Access, which only supported a built-in, weak encryption algorithm.

## A brief history of Borland's Delphi

### Pascal

Delphi uses the language Pascal, a third generation structured language. It is what is called a **highly typed** language. This promotes a clean, consistent programming style, and, importantly, results in more reliable applications. Pascal has a considerable heritage:

### Beginnings

Pascal appeared relatively late in the history of programming languages. It probably benefited from this, learning from Fortran, Cobol and IBM's PL/1 that appeared in the early 1960's. Niklaus Wirth is claimed to have started developing Pascal in 1968, with a first implementation appearing on a CDC 6000 series computer in 1970.

Curiously enough, the **C** language did not appear until 1972. C sought to serve quite different needs to Pascal. C was designed as a high level language that still provided the low level access that assembly languages gave. Pascal was designed for the development of structured, maintainable applications.

### The 1970's

In 1975, Wirth teamed up with Jensen to produce the definitive Pascal reference book "Pascal User Manual and Report". Wirth moved on from Pascal in 1977 to work on **Modula** - the successor to Pascal.

### The 1980's

In 1982 ISO Pascal appears. The big event is in November 1983, when **Turbo Pascal** is released in a blaze of publicity. Turbo Pascal reaches release 4 by 1987. Turbo Pascal excelled on speed of compilation and execution, leaving the competition in its wake.



## From Turbo Pascal to Delphi

Delphi, **Borland's** powerful **Windows?** and **Linux?** programming development tool first appeared in 1995. It derived from the **Turbo Pascal?** product line.

As the opposition took heed of Turbo Pascal, and caught up, Borland took a gamble on an Object Oriented version, mostly based on the Pascal object orientation extensions. The risk paid off, with a lot of the success due to the thought underlying the design of the IDE (Integrated Development Environment), and the retention of fast compilation and execution.

This first version of Delphi was somewhat limited when compared to today's heavyweights, but succeeded on the strength of what it did do. And speed was certainly a key factor. Delphi went through rapid changes through the 1990's.

## Delphi for Microsoft .Net

From that first version, Delphi went through 7 further iterations before Borland decided to embrace the competition in the form of the **Microsoft? .Net** architecture with the stepping stone Delphi 8 and then fully with Delphi 2005 and 2006. Delphi however still remains, in the opinion of the author, the best development tool for stand alone Windows and Linux applications. Pascal is a cleaner and much more disciplined language than Basic, and adapted much better to Object Orientation than Basic.

## **A new direction**

Delphi is now provided by a development tools only company.

### ***Delphi For Beginners:***

**Your guide will try to explain exactly what is Delphi and what can it do for you.**

Dateline: 1999

### **Preparations.**

First of all, I will presume that you know what computers are, what can you do with them, and finally what does programming mean, in general. It would also be great if you already have basic knowledge of programming (Pascal perhaps?).

If this is not true, you wouldn't be here anyway (am I right?). I'll be very glad if I'm not! So sit back, relax and enjoy reading this article.

### **Delphi**

Borland Delphi is a development tool for Microsoft Windows applications. Delphi is powerful and easy to use tool for generating stand-alone graphical user interface (GUI) programs or 32-bit console applications (programs that have no GUI presence but instead run in what is commonly referred to as a "DOS box.")

When paired with Borland Kylix, Delphi users can build single-source applications for both Windows and Linux, which opens new opportunities and increases the potential return on development investments.\* Use the Cross-platform CLX component library and visual designers to build high-performance portable applications for Windows that can be easily re-compiled on Linux.

Delphi is the first programming language to shatter the barrier between high-level, easy-to-use rapid application development environments and low-level bits-and-bytes power tools.

When creating GUI applications with Delphi, you have all the power of a true compiled programming language (Object Pascal) wrapped up in a RAD environment. All the common parts of the Windows graphical user interface, like forms, buttons and lists

objects, are included in Delphi as components. This means that you don't have to write any code when adding them to your application. You simply draw them onto your form like in a paint program. You can also drop ActiveX controls on forms to create specialized programs such as Web browsers in a matter of minutes. Delphi allows the developer to design the entire interface visually, and quickly implement an event driven code with the click of the mouse.

Delphi ships in a variety of configurations aimed at both departmental and enterprise needs. With Delphi, you can write Windows programs more quickly and more easily than was possible ever before.

### **Pascal**

The best way of describing Delphi is an Object Pascal-based visual development environment. Delphi's environment is based on Object Pascal, a language that is as object oriented as C++, and in some cases, better. For developers with no Pascal experience, its templates for Pascal program structures speed the process of learning the language.

The compiler produces applications packaged in compact executable files, with no need for bulky runtime libraries (DLL's)-a notable benefit, I must say.

### **VCL**

Visual Component Library (self-contained binary piece of software that performs some specific predefined function), or VCL, is Delphi's object-oriented framework. In this rich library, you'll find classes for Windows objects such as windows, buttons, etc, and you'll also find classes for custom controls such as gauge, timer and multimedia player, along with non-visual objects such as string lists, database tables, and streams.

### **Databases**

Delphi can access many types of databases. Using forms and reports that you create, the BDE (Borland Database Engine) can access local databases, like Paradox and dBase, network SQL server databases, like InterBase, and SysBase, and any data source accessible through ODBC (open database connectivity).



## Hello World!

At the end let's see one of the smallest Delphi applications: the famous 'Hello World!' program.

This example is not for beginners – there is no main form of application or something like that. This is only a demonstration. In some of the future articles I will focus on topics like Delphi for Beginners - How to get started.

```
program HelloWorld;  
uses dialogs;  
begin  
  ShowMessage('Hello World!');  
end.
```



## A Glossary of Delphi Programming Technical Terms

Definitions of terms having to do with Delphi programming, Pascal, OOP, BDE and programming in general

### "Self"

Definition: Within the implementation of a method, the identifier Self references the object in which the method is called.

### type

TCar = **Class**

color : TColor;

procedure ChangeColor(newColor : TColor) ;

**end;**

...

**procedure** TCar.ChangeColor(newColor : TColor) ;

**begin**

*//self is "this" instance*

```
Self.color := newColor;  
end;
```

In class methods the identifier Self represents the class where the method is called.

### "Constructor"

Definition: A constructor is a special method that creates and initializes instance objects. The declaration of a constructor looks like a procedure declaration, but it begins with the reserved word constructor.

A class can have more than one constructor, but most have only one. It is conventional to call the constructor Create.

To create an object, call the constructor method on a class type.

### type

```
TCar = Class  
  constructor Create;  
end;
```

...

```
car := TCar.Create;
```

### "Reserved Word"

Definition: A special word reserved by a programming language or by a program.

You are not allowed to use reserved words as variable names.

A partial list of Delphi reserved words:

- and
- array
- as
- asm
- begin

- case
- class
- const
- constructor
- destructor
- dispinterface
- div
- do
- downto
- else
- end
- except
- exports
- file
- finalization
- finally
- for
- function
- goto
- if
- implementation
- in
- inherited
- initialization
- interface
- in
- is
- library
- nil
- not
- object
- of
- or
- out



- packed
- procedure
- program
- property
- raise
- record
- repeat
- resourcestring
- set
- string
- then
- to
- try
- type
- unit
- until
- uses
- var
- while
- with

In addition to the words above, private, protected, public, published, and automated act as reserved words within object type declarations, but are otherwise treated as directives.

### **"Class Method"**

Definition: A class method is a method that operates on classes instead of objects.

The definition of a class method must begin with the reserved word class.

The most common used class method in Delphi language is the "Create" constructor.

In the defining declaration of a class method, the identifier Self represents the class where the method is called (which could be a descendant of the class in which it is

defined). If the method is called in the class TCar, then Self is of the type class of TCar.

## "Method"

Definition: Procedure or function (routine) associated with a particular object.

Different classes may define methods with the same name (Car.Drive or Scooter.Drive).

Most methods operate on objects that are instances of a certain class.

A class method is a method (other than a constructor) that operates on classes instead of objects.

A call to a method specifies the object (or, if it is a class method, the class) that the method should operate on.

Examples:

### type

TCar = **Class**

*//method procedure*

**procedure** Drive;

*//method (function)*

**function** ChangeGear(newGear : integer) ;

**end;**

## "Object"

Definition: An object is a variable of class. More generally, a variable of any type.

An instance of a class or object, is a self-contained entity that consists of both properties, events and methods to manipulate the data.

Each object has its own values for the instance variables of its class and can respond to the methods as well as raise events defined by its class.

Also Known As: Instance variable

### "Canvas"

Definition: Canvas is the graphical drawing surface of an object. The canvas has a brush, a pen, a font, and an array of pixels. The canvas encapsulates the Windows device context.

In Delphi, the TCanvas class provides an abstract drawing space for objects that must render their own images.

### "Class"

Definition: A list of features representing data and associated code assembled into single entity. A class includes not only features listed in its definition but also features inherited from ancestors.

The terms class and type are usually (but not always) interchangeable; a class is a slightly different concept than a type, in that it emphasizes the classifications of structure and behavior.

Classes are related in a class hierarchy. One class may be a specialisation (a "subclass") of another (one of its "superclasses"). A class may be an abstract class or a concrete class.

The Visual Component Library (CVL) is a class hierarchy of Delphi components and object types.

Also Known As: Object Type

Examples:

#### **type**

TCar = **Class**

Year : integer;

Color : TColor;

**end;**



## **"Run Time"**

Definition: Run time is any time you are actually running the application in the operating system and interacting with the application as the user would.

In Delphi, "dynamically creating ..." means "creating at run-time".

## **"RTL"**

Definition: The raw power of Delphi is based on a considerable amount of its Run Time Library functions and procedures.

RTL is the collection of functions and procedures that are built into Delphi.

Also Known As: Run Time Library; VCL Routines

## **"Routine"**

Definition: Self-contained statement blocks that can be called from different locations in a program. In Delphi: function or procedure.

Also Known As: Subroutine

## **"Recursion "**

Definition: Recursion is a very simple, yet useful and powerful programmer's tool. As we know, routines can, and frequently do, call other routines.

A routine that activates/calls itself is called recursive. Recursion is a general method of solving problems by reducing them to simpler problems of a similar type.

A recursive subroutine constantly calls itself, each time in a simpler situation, until it gets to the trivial case, at which point it stops.

## **"Procedure"**

Definition: A procedure is a routine that does not return a value (unlike a function).

Procedure header gives the name of a procedure followed by a list of formal parameters.

In a unit, a routine may have a header declared in the interface part, and then again in the implementation part. The second appearance of the header may be an exact duplicate of the header in the interface part, or may be only the name of the routine.

Examples: `[blockquote shade=yes] procedure TestMe(parameter: TCustomType[br] begin[br] ... end; [/blockquote]_z_delphi_z_);`

### **"Pointer"**

Definition: A pointer is a variable that holds the address of another variable (or routine) in memory.

A pointer can be used to indirectly manipulate the object.

### **"Parameter"**

Definition: Represents one value that is supplied by one function (the calling function) that wishes to make use of the services of another function (the called function).

In Delphi, every parameter is classified as value, variable, constant, or out.

Also Known As: Argument

Examples: `[blockquote] "year" and "name" are parameters for the "TestMe" function`

`procedure TestMe(const year: integer; var name : string) ; [/blockquote]`

### **"OLE"**

Definition: OLE is a compound document standard developed by Microsoft Corporation. It enables you to create objects with one application and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them.

With OLE, data from a server application is stored in a container application. The data is stored in an OLE Object.

Also Known As: Object Linking and Embedding

## **"MDI"**

Definition: A Windows API that enables programmers to easily create applications with multiple windows.

Each MDI application has a single main (frame) window, and any number of child windows (documents). All child windows are displayed within the main window - this is common in applications such as spreadsheets or word processors.

The child window's document title merges with the parent window's title bar when the child window is maximized.

Although many programmers still use MDI, Microsoft recommends using a newer API called Single Document Interface (SDI).

Also Known As: Multiple Document Interface

## **"IDE"**

Definition: IDE (Integrated Development Environment) is the user interface (GUI) where you can design, compile and debug your Delphi projects.

Also Known As: Integrated Development Environment

## **"GUI"**

Definition: A GUI (usually pronounced GOO-ee) is a graphical (rather than purely textual) user interface to a computer.

Applications typically use the elements of the GUI that come with the operating system and add their own graphical user interface elements and ideas. When creating an application, Delphi facilitate writing a graphical user interface.

Each GUI element (for example a Button or an EditBox) is defined as a class from which you can create object instances for your application.

Also Known As: Graphical User Interface

Alternate Spellings: goo-ee



## "Function"

Definition: A function is a routine that returns a value when it executes.

It can be passed and it can return a value. Functions that are part of a class are usually called methods.

You can code your own functions or use built-in functions provided by Delphi RTL (run time library).

Examples:

```
function YearsOld(const BirthYear:integer): integer;
```

```
var
```

```
Year, Month, Day : Word;
```

```
begin
```

```
DecodeDate(Date, Year, Month, Day) ;
```

```
Result := Year - BirthYear;
```

```
end;
```

## "Freeware"

Definition: Copyrighted software given away for free by the author. Although it is available for free, the author retains the copyright, which means that you cannot do anything with it that is not expressly allowed by the author. Usually, the author allows people to use the software, but not sell it.

## "Exception"

Definition: An event happening during execution of a program that disrupts the normal flow of control. Exceptions are raised when a runtime error occurs in an application, such as attempting to divide by zero.

Also, an exception is an object that contains information about what error occurred and where it happened.

## **"Design Time"**

**Definition:** We work with forms and controls, set their properties, and write code for their events at design time, which is any time we're building an application in the Delphi's IDE.

Design-time is when you use the IDE to design your application, using the form, the Object Inspector, Component palette, Code editor, and so forth; as opposed to run-time, when the application you design is actually running.

## **"Compiler"**

**Definition:** A compiler is a program that performs the process of compilation. When you press F9 in Delphi IDE your current project gets compiled and run.

## **"Compilation"**

**Definition:** Compilation is the process of translating source code into an object program, which is composed of machine instructions along with the data needed by those instructions. Virtually all of the software on your computer was created by this process.

Compiled programs (Delphi applications for example) run faster than "interpreted" - which is the line-by-line translation of source code to machine instructions (Visual Basic applications for example).

## **"Comment"**

**Definition:** The purpose of adding comments to Delphi code is to provide more program readability using understandable description of what your code is doing.

A comment is a note to yourself or another programmer; it is ignored by the compiler.

There are several ways to construct comments:

{ Text between a left brace and a right brace constitutes a comment. }

(\* Text between a left-parenthesis-plus-asterisk and an asterisk-plus-right-parenthesis also constitutes a comment. \*)

// Any text between a double-slash and the end of the line constitutes a comment.

Also Known As: REM meaning "Remark in Basic"

## **"COM"**

Definition: The Component Object Model (COM) enables programmers to develop objects that can be accessed by any COM-compliant application. Both OLE and ActiveX are based on COM.

The key aspect of COM is that it enables communication between clients and servers through interfaces. Information about these interfaces is usually included in a type library.

COM allows you to create COM objects that are not specific to any language, and in some cases, even platforms. For instance, COM objects can be ported to a Unix system. COM also allows you to create COM Objects that will be instantiated on a different machine across the world if you so desired.

Although often associated with Microsoft, COM is an open standard that specifies how components work together and interoperate.

Also Known As: Component Object Model

## **"Callback Routine"**

Definition: A callback routine is a routine (function or procedure) in your program that Windows calls. More generally, a callback is a means of sending a function as a parameter into another function. When the callback function has completed, control is passed back to the original function.

For example, EnumFonts is a Windows routine that calls a given callback function for every font installed in the system.

## **"BDE"**

Definition: The core database engine and connectivity software behind Borland products, as well as Paradox for Windows and Visual dBASE for Windows. The



included set of database drivers enables consistent access to standard data sources: Paradox, dBASE, FoxPro, Access, and text databases.

Many Delphi components use this database engine to access and deliver data. BDE maintains information about your PC's environment in the BDE configuration file (usually called IDAPI.CFG). Use the BDE Administrator to change the settings in this configuration file.

Also Known As: Borland Database Engine, IDAPI

## **"Application"**

Definition: An application is the executable file and all related files that a program needs to function which serve a common purpose or purposes, as distinguished from the design and source code of the project.

Software applications can be divided into two general classes: systems software and applications software. Systems software consists of low-level programs that interact with the computer at a very basic level. This includes operating systems, compilers, and utilities for managing computer resources.

In contrast, applications software (also called end-user programs) includes database programs, word processors, spreadsheets, etc. Figuratively speaking, applications software sits on top of systems software because it is unable to run without the operating system and system utilities.

In general we use Delphi to produce applications software.

## **"API"**

Definition: A set of routines, protocols, and tools for building software applications. A wide variety of software from operating systems to individual components are said to have an API.

A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Also Known As: Application Programming Interface

## **"Alias"**

Definition: A name that specifies the location of database tables accessed using the BDE. The terms alias and database are \*synonymous\* when talking about the BDE. An alias specifies driver parameters and database locations, such as Driver Type, Server Name, User Name and others.

## **"Algorithm"**

Definition: An algorithm is a set of precisely defined steps guaranteed to arrive at an answer to a problem or set of problems. As this implies, a set of steps that might never end is not an algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

## **"ActiveX"**

Definition: A technology that allows various software components to communicate and interact, even though they are not written in the same language. ActiveX controls can be embedded in Web pages to produce animation and other multimedia effects, interactive objects and sophisticated applications.

An ActiveX control is a COM-based software component that integrates into and extends the functionality of any host application. ActiveX controls implement a set of predefined COM interfaces.

The ActiveX page of the component palette includes several ActiveX controls. You can use them like any standard VCL component, dropping them on forms and setting their properties using the Object Inspector.

## **"ASCII"**

Definition: ASCII assigns each English character and basic punctuation mark its own number from 0 to 127. Since the code is standard, every computer should be able to translate it into serviceable, if unglamorous, copy. So, when you're unsure what program - or what computer - is on the receiving end of a document, your safest bet is to save your file as plain ASCII text.

Examples: The capital letter A has an ASCII value of 65. The ASCII code for a space is 32.

You can reference a character by its ASCII code prefixed with a number sign (#).

Example: To put the symbol for American cents into a character C, for example, you could code "c := #155;".

Pronunciation: ask-ee

Also Known As: American Standard Code for Information Interchange

## **Understanding Delphi Project Files (.DPR)**

### **New: Delphi Project**

Since it is quite common for Delphi applications to share code or previously customized forms, Delphi organizes applications into what is called projects.

A project is made up of the visual interface along with the code that activates the interface. Each project can have multiple forms, allowing us to build applications that have multiple windows. The code that is needed for a form in our project is stored in a separate Unit file that Delphi automatically associates to the form. General code that we want to be shared by all the forms in our application is placed in unit files as well. Simply put, a Delphi project is a collection of files that make up an application.

What this means is that each project is made of one or more Form files (files with the *.dfm* extension) and one or more Unit files (*.pas* extension).

We can also add resource files, and they are compiled into *.RES* files and linked when we compile the project.

### **Project File**

Each project is made up of a single project file (*.dpr*). Project files contain directions for building an application. This is normally a set of simple routines which open the main form and any other forms that are set to be opened automatically and then starts the program by calling the *Initialize*, *CreateForm* and *Run* methods of the



global Application object (which is actually a form of zero width and height, so it never actually appears on the screen).

Note: The global variable *Application*, of type TApplication, is in every Delphi Windows application. Application encapsulates your application as well as provides many functions that occur in the background of the program. For instance, Application would handle how you would call a help file from the menu of your program.

## Project Unit

Use Project - View Source to display the project file for the current project.

Although you can look and edit the Project File, in most cases, you'll let Delphi maintain the DPR file. The main reason to view the project file is so we can see the units and forms that make up the project, and which form is specified as the application's main form.

Another reason to work with the project file is when we are creating a DLL rather than a stand-alone application or need some start-up code, such as a splash screen before the main form is created by Delphi.

Here is the default project file for a new application (containing one form: "Form1"):

```
program Project1;  
uses  
    Forms,  
    Unit1 in 'Unit1.pas' {Form1};  
{$R *.RES}  
begin  
    Application.Initialize;  
    Application.CreateForm(TForm1, Form1);  
    Application.Run;  
end.
```

The **program** [link url=/od/delhiprogrammingglossary/g/reservedword.htm]keyword identifies this unit as a program's main source unit. You can see that the unit name,

Project1, follows the program keyword (Delphi gives the project a default name until you save the project with a more meaningful name). When we run a project file from the IDE, Delphi uses the name of the Project file for the name of the EXE file that it creates.

Delphi reads the **uses** clause of the project file to determine which units are part of a project.

The .dpr file is linked with the .pas file with the compile directive `{$R *.RES}` (in this case '\*' represents the root of the .pas filename rather than "any file"). This compiler directive tells Delphi to include this project's resource file. The project's resource file contains such items as the project's icon image.

The **begin..end** block is the main source-code block for the project.

Although **Initialize** is the first method called in the main project source code, it is not the first code that is executed in an application. The application first executes the **"initialization" section of all the units** used by the application.

The **Application.CreateForm** statement loads the form specified in its argument. Delphi adds an **Application.CreateForm** statement to the project file for each form you add to the project. This code's job is to first allocate memory for the form. The statements are listed in the order the forms are added to the project. This is the order that the forms will be created in memory at runtime. If you want to change this order, do not edit the project source code. Use the Project|Options menu command.

The **Application.Run** statement starts your application. This instruction tells the predeclared object called **Application** to begin processing the events that occur during the run of a program.

### **An Example: Hide Main Form / Hide Taskbar Button**

The **Application** object's **ShowMainForm** property determines whether or not a form will show at startup. The only condition of setting this property is that it has to be called before the **Application.Run** line.

```
//Presume: Form1 is the MAIN FORM  
Application.CreateForm(TForm1, Form1) ;  
Application.ShowMainForm := False;  
Application.Run;
```

## **Understanding the Birth, Life and Death of a Delphi Form**

### **Life-Cycle of a Delphi Form**

In Windows, most elements of the user interface are windows. In Delphi, every project has at least one window - program's main window. All windows of a Delphi application are based on TForm object.

#### **Form**

Form objects are the basic building blocks of a Delphi application, the actual windows with which a user interacts when they run the application. Forms have their own properties, events, and methods with which you can control their appearance and behavior. A form is actually a Delphi component, but unlike other components, a form doesn't appear on the component palette.

We normally create a form object by starting a new application (File | New Application). This newly created form will be, by default, the application's main form - the first form created at runtime.

Note: To add an additional form to Delphi project, we select File|New Form.

There are, of course, other ways to add a "new" form to a Delphi project.

#### **Birth**

##### **OnCreate**

The OnCreate event is fired when a TForm is first created, that is, only once. The statement responsible for creating the form is in the project's source (if the form is set to be automatically created by the project). When a form is being created and its Visible property is True, the following events occur in the order listed: OnCreate, OnShow, OnActivate, OnPaint.



You should use the OnCreate event handler to do, for example, initialization chores like allocating string lists.

Any objects created in the OnCreate event should be freed by the OnDestroy event.

OnCreate -> OnShow -> OnActivate -> OnPaint -> OnResize -> OnPaint ...

### **OnShow**

This event indicates that the form is being displayed. OnShow is called just before a form becomes visible. Besides main forms, this event happens when we set forms Visible property to True, or call the Show or ShowModal method.

### **OnActivate**

This event is called when the program activates the form - that is, when the form receives the input focus. Use this event to change which control actually gets focus if it is not the one desired.

### **OnPaint, OnResize**

Events like OnPaint and OnResize are always called after the form is initially created, but are also called repeatedly. OnPaint occurs before any controls on the form are painted (use it for special painting on the form).

### **Life**

As we have seen the birth of a form is not so interesting as the life and death can be. When your form is created and all the controls are waiting for events to handle, the program is running until someone tries to close the form!

### **Death**

An event-driven application stops running when all its forms are closed and no code is executing. If a hidden form still exists when the last visible form is closed, your application will appear to have ended (because no forms are visible), but will in fact continue to run until all the hidden forms are closed. Just think of a situation where the main form gets hidden early and all other forms are closed.

... OnCloseQuery -> OnClose -> OnDeactivate -> OnHide -> OnDestroy

## OnCloseQuery

When we try to close the form using the Close method or by other means (Alt+F4), the OnCloseQuery event is called. Thus, event handler for this event is the place to intercept a form's closing and prevent it. We use the OnCloseQuery to ask the users if they are sure that they really want the form to close.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean) ;  
begin  
  if MessageDlg('Really close this window?', mtConfirmation, [mbOk, mbCancel], 0) =  
  mrCancel then CanClose := False;  
end;
```

An OnCloseQuery event handler contains a CanClose variable that determines whether a form is allowed to close. The OnCloseQuery event handler may set the value of CloseQuery to False (via the CanClose parameter), thus aborting the Close method.

## OnClose

If OnCloseQuery indicates that the form should be closed, the OnClose event is called.

The OnClose event gives us one last chance to prevent the form from closing. The OnClose event handler has an Action parameter, with the following four possible values:

- **caNone**. The form is not allowed to close. Just as if we have set the CanClose to False in the OnCloseQuery.
- **caHide**. Instead of closing the form you hide it.
- **caFree**. The form is closed, so it's allocated memory is freed by Delphi.
- **caMinimize**. The form is minimized, rather than closed. This is the default action for MDI child forms. Note: When a user shuts down Windows, the OnCloseQuery event is activated, not the OnClose. If you want to prevent Windows from shutting down, put your code in the OnCloseQuery event handler, of course CanClose=False will not do the trick.

## OnDestroy

After the OnClose method has been processed and the form is to be closed, the OnDestroy event is called. Use this event for operations opposite to those in the OnCreate event. OnDestroy is therefore used to deallocate objects related to the form and free the corresponding memory.

Of course, when the main form for a project closes, the application terminates.

## Understanding and Using Functions and Procedures

Have you ever found yourself writing the same code over and over to perform some common task within event handlers? Yes! It's time for you to learn about programs within a program. Let's call those mini programs subroutines.

### Intro to subroutines

Subroutines are an important part of any programming language, and Delphi is no exception. In Delphi, there are generally two types of subroutines: a **function** and a **procedure**. The usual difference between a function and a procedure is that a **function can return a value, and a procedure generally will not do so**. A function is normally called as a part of an expression.

Take a look at the following examples:

```
procedure SayHello(const sWhat:string) ;
```

```
begin
```

```
    ShowMessage('Hello ' + sWhat) ;
```

```
end;
```

```
function YearsOld(const BirthYear:integer): integer;
```

```
var
```

```
    Year, Month, Day : Word;
```

```
begin
```

```
    DecodeDate(Date, Year, Month, Day) ;
```



```
Result := Year - BirthYear;  
end;
```

Once subroutines have been defined, we can call them one or more times:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
begin  
    SayHello('Delphi User') ;  
end;
```

```
procedure TForm1.Button2Click(Sender: TObject) ;  
begin  
    SayHello('Zarko Gajic') ;  
    ShowMessage('You are ' + IntToStr(YearsOld(1973)) + ' years old!') ;  
end;
```

## Functions and Procedures

As we can see, both functions and procedures act like mini programs.

In particular, they can have their own type, constants and variable declarations inside them.

Take a closer look at a (miscellaneous) SomeCalc function:

```
function SomeCalc  
    (const sStr: string;  
     const iYear, iMonth: integer;  
     var iDay: integer): boolean;  
begin  
    ...  
end;
```

Every procedure or function begins with a *header* that identifies the procedure or function and lists the *parameters* the routine uses, if any. The parameters are listed within parentheses. Each parameter has an identifying name and usually has a type. A semicolon separates parameters in a parameter list from one another.

sStr, iYear and iMonth are called *constant parameters*. Constant parameters cannot be changed by the function (or procedure). The iDay is passed as a *var parameter*, and we can make changes to it, inside the subroutine.

Functions, since they return values, must have a *return type* declared at the end of the header. The return value of a function is given by the (final) assignment to its name. Since every function implicitly has a local **variable Result** of the same type as the functions return value, assigning to Result has the same effect as assigning to the name of the function.

### Positioning and Calling Subroutines

Subroutines are always placed inside the implementation section of the unit. Such subroutines can be called (used) by any event handler or subroutine in the same unit that is defined after it.

Note: the uses clause of a unit tells you which units it can call. If we want a specific subroutine in a Unit1 to be usable by the event handlers or subroutines in another unit (say Unit2), we have to:

- Add Unit1 to the uses clause of Unit2
- Place a copy of the header of the subroutine in the interface section of the Unit1.

This means that subroutines whose headers are given in the interface section are *global in scope*.

When we call a function (or a procedure) inside its own unit, we use its name with whatever parameters are needed. On other hand, if we call a global subroutine (defined in some other unit, e.g. MyUnit) we use the name of the unit followed by a period.

...

*//SayHello procedure is defined inside this unit*

SayHello('Delphi User') ;

*//YearsOld function is defined inside MyUnit unit*

```
Dummy := MyUnit.YearsOld(1973) ;
```

Note: functions or procedures can have their own subroutines **embedded** inside them. An embedded subroutine is local to the container subroutine and cannot be used by other parts of the program. Something like:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
    function IsSmall(const sStr:string):boolean;  
    begin  
        //IsSmall returns True if sStr is in lowercase, False otherwise  
        Result:=LowerCase(sStr)=sStr;  
    end;  
begin  
    //IsSmall can only be uses inside Button1 OnClick event  
    if IsSmall(Edit1.Text) then  
        ShowMessage('All small caps in Edit1.Text')  
    else  
        ShowMessage('Not all small caps in Edit1.Text') ;  
end;
```

### Understanding and Using Decisions

```
if language = Delphi then  
begin  
    Use(language)  
end  
else  
    Skip(language) ;
```

### Branching

If you want to **control the flow of code execution** depending on what the program has already done or what it has just encountered you need to use one of the two Delphi Pascal branching statements: **if statements** and **case statements**.



## The IF THEN ELSE statement

The if statement is used to test for a condition and then execute sections of code based on whether that condition is True or False. The condition is described with a Boolean expression, If the condition is True, the code flow branches one way. If the condition is False, the flow branches in another direction. Let's see this behavior on an example:

```
var iNumber : Integer;  
begin  
  //some value must be  
  //assigned to iNumber here!  
  if iNumber = 0 then  
    ShowMessage('Zero value encountered!') ;  
  end;
```

If the number (assigned to iNumber variable) is 0, the expression `iNumber = 0` evaluates to True and the message is displayed; otherwise, nothing is displayed.

If we want more than one thing to happen when the tested condition is True, we can write multiple statements in a **begin ... end** block.

```
var iNumber : Integer;  
begin  
  //some value must be  
  //assigned to iNumber here!  
  if iNumber = 0 then  
    begin  
      ShowMessage('Zero value encountered!') ;  
      Exit; // exit from the current procedure  
    end;  
    //if iNumber is 0 the following  
    //code will never be executed  
    ShowMessage('Nobody likes 0, ha!') ;  
  end;
```

More often, we will want to process multiple statements if a condition is True or False.

```

var iNumber : Integer;
begin
    //some value must be
    //assigned to iNumber here!
    if iNumber < 0 then
        begin
            //statements ...
            ShowMessage('Your number is negative!') ;
            //statements ...
        end
    else
        begin
            //statements ...
            ShowMessage('Your number is positive or zero!') ;
            //statements ...
        end;
    end;
end;

```

Note: Each statement in the begin..end block ends with a semicolon. We cannot have a semicolon before or after the else keyword. The if-then-else statement, is a single statement, therefore we cannot place a semicolon in the middle of it.

An if statement can be quite complex. The condition can be turned into a series of conditions (using the and, or and not Boolean operators), or the if statement can nest a second if statement.

```

var
    iNumber : Integer;
begin
    if iNumber = 0 then
        begin
            ShowMessage('Zero number not allowed!') ;
            exit;
        end
    else

```

*//no need to use begin-end here*

**if** iNumber < 0 **then**

    ShowMessage('Your number is negative!')

**else**

    ShowMessage('Your number is positive!') ;

**end;**

**Note:** When you write nested if statements choose a consistent, clear indentation style. This will help you and anyone else who reads your code see the logic of the if statement and how the code flows when your application runs.

### **The CASE statement**

Although, we can use the if statement for very complex (nested) condition testing, the case statement is usually easier to read (debug!) and the code runs more quickly.

The case statement makes it clear that a program has reached a point with many branches; multiple if-then statements do not.

**var**

    iNumber : Integer;

**begin**

*//some value must be*

*//assigned to iNumber here!*

**case** iNumber **of**

        0 : ShowMessage('Zero value') ;

        1..10 : ShowMessage('Less than 11, greater than 0') ;

        -1, -2, -3 : ShowMessage('Number is -1 or -2 or -3') ;

**else**

        ShowMessage('I do not care') ;

**end;**

**end;**

What follows the case keyword is usually called the selector. The selector is a variable or expression taken from either the char type or any integer type (an ordinal



type). String type are *invalid!*. However, the StringToCaseSelect custom function enables you to use the Case statement with string type variables

As you can see, the individual case statements use a single constant, a group of constants (separated by comma), or a range of constants (double dot separated). We can even add an else keyword to take care of all the remaining cases at once.

Note 1: Only one case statement will be executed, we cannot have overlapping conditions in the case statements.

Note 2: If you want to include more than one statement in the part following the colon (:), place the begin and end keywords around the multiple statements.

## Understanding and Using Loops

### Repeating operations in Delphi Pascal

The loop is a common element in all programming languages. Object Pascal has three control structures that execute blocks of code repeatedly: for, repeat ... until and while ... do.

#### The FOR loop

Suppose we need to repeat an operation a fixed number of times.

```
// show 1,2,3,4,5 message boxes
```

```
var j: integer;
```

```
begin
```

```
  for j := 1 to 5 do
```

```
  begin
```

```
    ShowMessage('Box: '+IntToStr(j)) ;
```

```
  end;
```

```
end;
```

The value of a control variable (j), which is really just a counter, determines how many times a for statement runs. The keyword for sets up a counter. In the preceding example, the starting value for the counter is set to 1.

The ending value is set to 5.

When the for statement begins running the counter variable is set to the starting value. Delphi then checks whether the value for the counter is less than the ending value. If the value is greater, nothing is done (program execution jumps to the line of code immediately following the for loop code block). If the starting value is less than the ending value, the body of the loop is executed (here: the message box is displayed). Finally, Delphi adds 1 to the counter and starts the process again.

Sometimes it is necessary to count backward. The **downto** keyword specifies that the value of a counter should be decremented by one each time the loop executes (it is not possible to specify an increment / decrement other than one). An example of a for loop that counts backward.

```
var j: integer;
begin
  for j := 5 downto 1 do
    begin
      ShowMessage('T minus ' + IntToStr(j) + 'seconds') ;
    end;
    ShowMessage('For sequence executed!') ;
  end;
```

Note: it's important that you never change the value of the control variable in the middle of the loop. Doing so will cause errors.

### **Nested FOR loops**

Writing a for loop within another for loop (nesting loops) is very useful when you want to fill / display data in a table or a grid.

```
var k,j: integer;
begin
  //this double loop is executed 4x4=16 times
  for k:= 1 to 4 do
    for j:= 4 downto 1 do
      ShowMessage('Box: ' + IntToStr(k) + ' ' + IntToStr(j)) ;
    end;
```

The rule for nesting for-next loops is simple: the inner loop (j counter) must be completed before the next statement for the outer loop is encountered (k counter). We can have triply or quadruply nested loops, or even more.

**Note:** Generally, the begin and end keywords are not strictly required, as you can see. If begin and end are not used, the statement immediately following the for statement is considered the body of the loop.

### The FOR-IN loop

If you have Delphi 2005 or any newer version, you can use the "new" for-element-in-collection style iteration over containers. The following example demonstrates iteration over string expressions: for each char in string check if the character is either 'a' or 'e' or 'i'.

**const**

s = 'About Delphi Programming';

**var**

c : char;

**begin**

for c in s do

begin

if c in ['a','e','i'] then

begin

*// do something*

end;

end;

end;

### The WHILE and REPEAT loops

Sometimes we won't know exactly how many times a loop should cycle. What if we want to repeat an operation until we reach a specific goal?

The most important difference between the while-do loop and the repeat-until loop is that the code of the repeat statement is always executed at least once.



The general pattern when we write a repeat (and while) type of loop in Delphi is as follows:

```
repeat
begin
    statements;
end;
until condition = true

while condition = true do
begin
    statements;
end;
```

Here is the code to show 5 successive message boxes using repeat-until:

```
var
    j: integer;
begin
    j:=0;
    repeat
        begin
            j := j + 1;
            ShowMessage('Box:'+IntToStr(j)) ;
        end;
    until j > 5;
end;
```

As you can see, the repeat statement evaluates a condition at the end of the loop (therefore repeat loop is executed for sure at least once).

The while statement, on the other hand, evaluates a condition at the beginning of the loop. Since the test is being done at the top, we will usually need to make sure that the condition makes sense before the loop is processed, if this is not true the compiler may decide to remove the loop from the code.

```
var j: integer;
begin
```

```

j:=0;
while j < 5 do
begin
  j:=j+1;
  ShowMessage('Box:'+IntToStr(j)) ;
end;
end;

```

## Break and Continue

The Break and Continue procedures can be used to control the flow of repetitive statements: The Break procedure causes the flow of control to exit a for, while, or repeat statement and continue at the next statement following the loop statement. Continue allows the flow of control to proceed to the next iteration of repeating operation.

## Understanding Typed Constants in Delphi

### How to implement persistent values between function calls.

When Delphi invokes an event handler, the old values of local variables are wiped out. What if we want to keep track of how many times a button has been clicked? We could have the values persist by using a unit-level variable, but it is generally a good idea to reserve unit-level variables only for sharing information. What we need are usually called static variables or typed constants in Delphi.

### Variable or constant?

Typed constants can be compared to initialized variables-variables whose values are defined on entry to their block (usually event handler). Such a variable is initialized only when the program starts running. After that, the value of a typed constant persists between successive calls to their procedures.

Using typed constants is a very clean way of implementing automatically initialized variables.

To implement these variables without typed constants, we'll need to create an initialization section that sets the value of each initialized variable.

### Variable typed constants

Although we declare typed constants in the const section of a procedure, it is important to remember that they are not constants. At any point in your application, if you have access to the identifier for a typed constant you'll be able to modify its value.

To see typed constants at work, put a button on a blank form, and assign the following code to the OnClick event handler:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
const  
    clicks : Integer = 1; //not a true constant  
begin  
    Form1.Caption := IntToStr(clicks) ;  
    clicks := clicks + 1;  
end;
```

Notice that every time you click on the button, forms caption increments steadily.

Now try the following code:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
var  
    clicks : Integer;  
begin  
    Form1.Caption := IntToStr(clicks) ;  
    clicks := clicks + 1;  
end;
```

We are now using uninitialized variable for the clicks counter. Notice that weird value in the forms caption after you click on the button.



## Constant typed constants

You have to agree that idea of modifiable constants sound a bit strange. In 32 bit versions of Delphi Borland decided to discourage their use, but support them for Delphi 1 legacy code.

We can enable or disable Assignable typed constants on the Compiler page of the Project Options dialog box.

If you've disabled Assignable typed constants for a given project, when you attempt to compile previous code Delphi will give you 'Left side cannot be assigned to' error upon compilation. You can, however, create assignable typed constant by declaring:

```
{SJ+}
```

```
const clicks : Integer = 1;
```

```
{SJ-}
```

Therefore, the first example code looks like:

```
procedure TForm1.Button1Click(Sender: TObject) ;
```

```
const
```

```
{SJ+}
```

```
    clicks : Integer = 1; //not a true constant
```

```
{SJ-}
```

```
begin
```

```
    Form1.Caption := IntToStr(clicks) ;
```

```
    clicks := clicks + 1;
```

```
end;
```

## **Running Delphi Applications With Parameters**

**How to pass command-line parameters to your application and how to handle them.**

In the days of DOS it was a common practice run applications (command line programs) with some kind of parameters that will specify what we want to do. Even now, in the world of Windows, we can go to MS-Dos prompt and run MS-DOS based program like DIR /?. That '/' after program name (DIR) will give us some help regarding the usage of the DIR command.

In this article, we will find out how to respond to command line parameters passed to a Delphi application.

### **Parameters**

We can pass the parameter from the command line in Windows or from the development environment in Delphi under Run-Parameters menu option.

We will use Parameters dialog box to pass command-line parameters to an application when we run it (for testing purposes - from within Delphi), just as if we were running the application from the Windows Explorer.

### **ParamCount, ParamStr()**

Simply put, the ParamCount function returns the number of parameters passed to the program on the command line, and ParamStr returns a specified parameter from the command-line.

While application is running, the parameters are available to us so we can retrieve them within a specific section of the application (usually from the OnActivate event handler of the main form).

Note: In a program, the CmdLine variable contains a string with command-line arguments specified when the application was started. We can use CmdLine to access the entire paramstring passed to an application.

We'll start with a simple application. Start up a new project and place a Button component on Form. In the button's OnClick event handler, write the following code:

#### **Procedure**

```
TForm1.Button1Click(Sender: TObject) ;  
begin  
  ShowMessage(ParamStr(0)) ;  
end;
```

When you run the program and click the button, a message box appears with the path and file name of the executing program.

We can see, that even if we haven't passed any parameters to our application ParamStr function "works", the reason is that the array value 0 stores the file name of the executable application including path information.

Now, choose Parameters from the Run Menu and add 'Delphi Programming' to the drop down list (without apostrophes).

Note: when you pass parameters to your application separate them with spaces or tabs. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

We will be looping through the amount of parameters using ParamCount() to get the value of parameters passed, with ParamStr(i).

Change the button's OnClick event handler to:

```
procedure TForm1.Button1Click(Sender: TObject) ;  
var  
  j:integer;  
begin  
  for j := 1 to ParamCount do  
    ShowMessage(ParamStr(j)) ;  
end;
```



When you run the program and click the button, a message box appears displaying 'Delphi' (first parameter) and 'Programming' (second parameter).

Note: Working with parameters passed to the console mode application is the same.

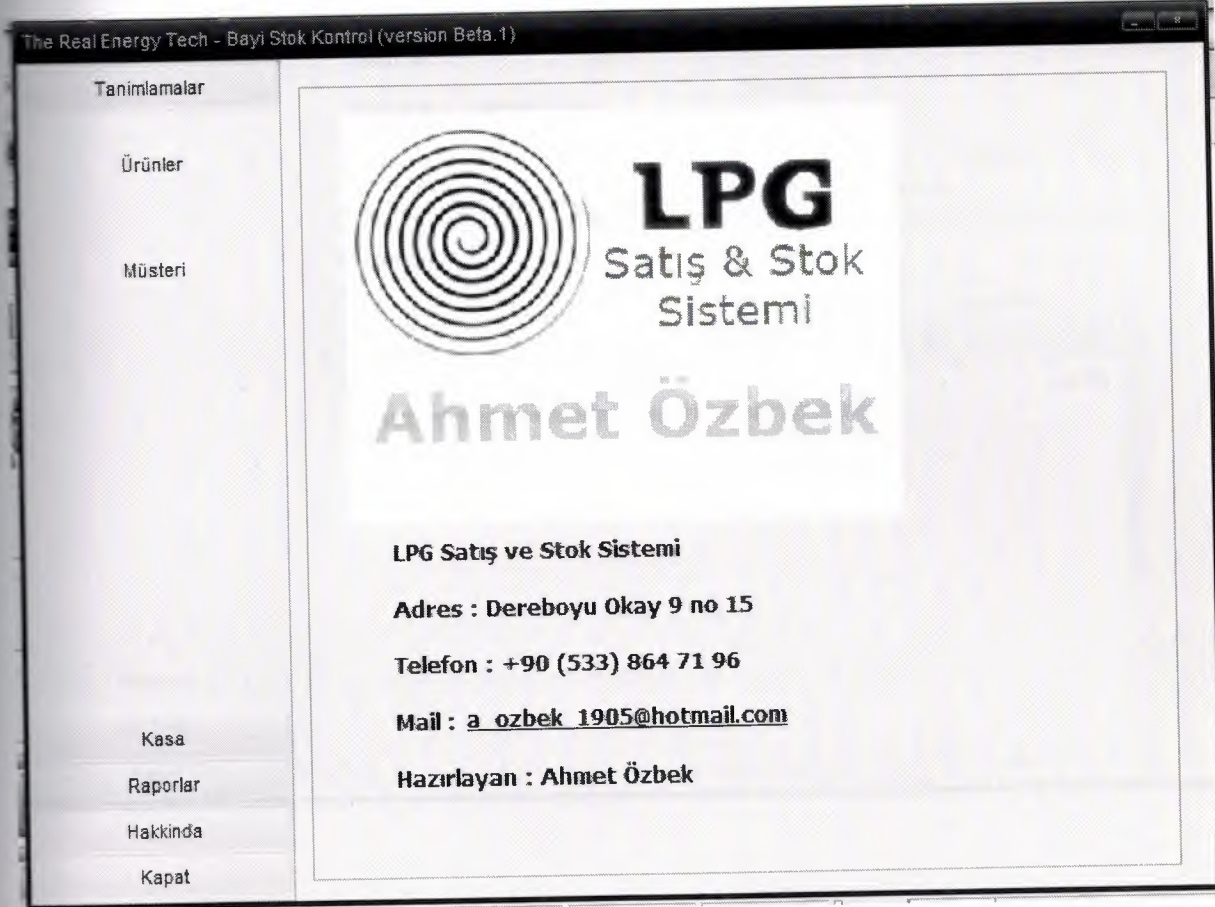
That's it, simple as only Delphi can be!

## CHAPTER THREE

### DEVELOPMENT OF LPG DISTRIBUTOR STOCK & SALE SYSTEM

The software development of LPG distributor stock & sale system is considered. The system is designed using delphi programme.

#### Main Menu



After we run the program main window welcomes us. In this window there are some menu items on the left.

This buttons helps us to navigate the system easily.

## Products Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Ürünler Müsteriler

Ürünler

Müşteri

Ürün No : 1

Ürün Adı : Gapgaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Birim Fiyat : 35,00 YTL

Stok Miktarı : 990

Yeni

Düzenle

Sil

Kaydet

iptal

Arama

Ürün No

Ürün Adı	Açıklama	Birim Fiyat	Stok Miktar
Gapgaz 25 Kg	Ev Tüpü 25 kg	35	990
LPG 1Lt	Litre ile Satış	1	100000

Kasa

Raporlar

Hakkında

Kapat

After we clicked the “products” button an inline window appears on the content panel. In this window we can record the stocks with all the details and we can search products.



## Customers Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Ürünler Müsteriler

Ürünler

Müşteri

Kasa

Raporlar

Hakkında

Kapat

Müşteri No : 1

Firma Adı : Melek A.Ş.

Müşteri Adı Soyadı : Burak Melek

Telefon : 5338662213

Faks :

Adres : fhkjshfhkjshfsjhfsjfhdsjft

Açıklama :

Yeni

Düzenle

Sil

Kaydet

İptal

Arama Firma Adı

Yetkili Adı	Firma Adı	Telefon	Faks
► Burak Melek	Melek A.Ş.	5338662213	

İşlemler

To collect the orders properly we need to record the customers. So we can record whom we sold the products.

## Orders Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Kasa

Alis

Satis

Raporlar

Hakkında

Kapat

Alis Satis

Arama

Ürün No : 1

Ürün No

Ürün Adı : Gaggaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Müşteri Adı (Firma)

Melek A.Ş

Birim Fiyat : 35

Stok Miktarı : 990

Onayla

Miktar

100

Ödeme Türü

Kredi Kartı

Onayla

Birim Fiyat

48,00 YTL

Açıklama

iptal

4.800,00 YTL

Önceki işlem

Ödeme Tarihi

14.08.2007

Durum

Henüz Ödenmedi

Alacak Kaydet

iptal

Yazdır

In this window we record the orders. First we choose the product by searching with any information and select customer from the combo box. And then we click the "confirm" button. Now we can enter the quantity, unit price and payment type. Then we click the confirm button twice. After all we select the payment date and payment situation.



## Purchase Window

The Real Energy Tech - Bayi Stok Kontrol (version Beta.1)

Tanımlamalar

Kasa

Alis

Satis

Raporlar

Hakkında

Kapat

Alis Satis

Arama

Ürün No

Ürün No : 1

Ürün Adı : Gaggaz 25 Kg

Açıklama : Ev Tüpü 25 kg

Birim Fiyat : 35

Stok Miktarı : 990

Onayla

Miktar

1

Birim Fiyat

42,00 YTL

42,00 YTL

Ödeme Türü

Çek

Açıklama

Onayla

iptal

Önceki işlem

Ödeme Tarihi

13.08.2007

Durum

Henüz Ödenmedi

Borc Kaydet

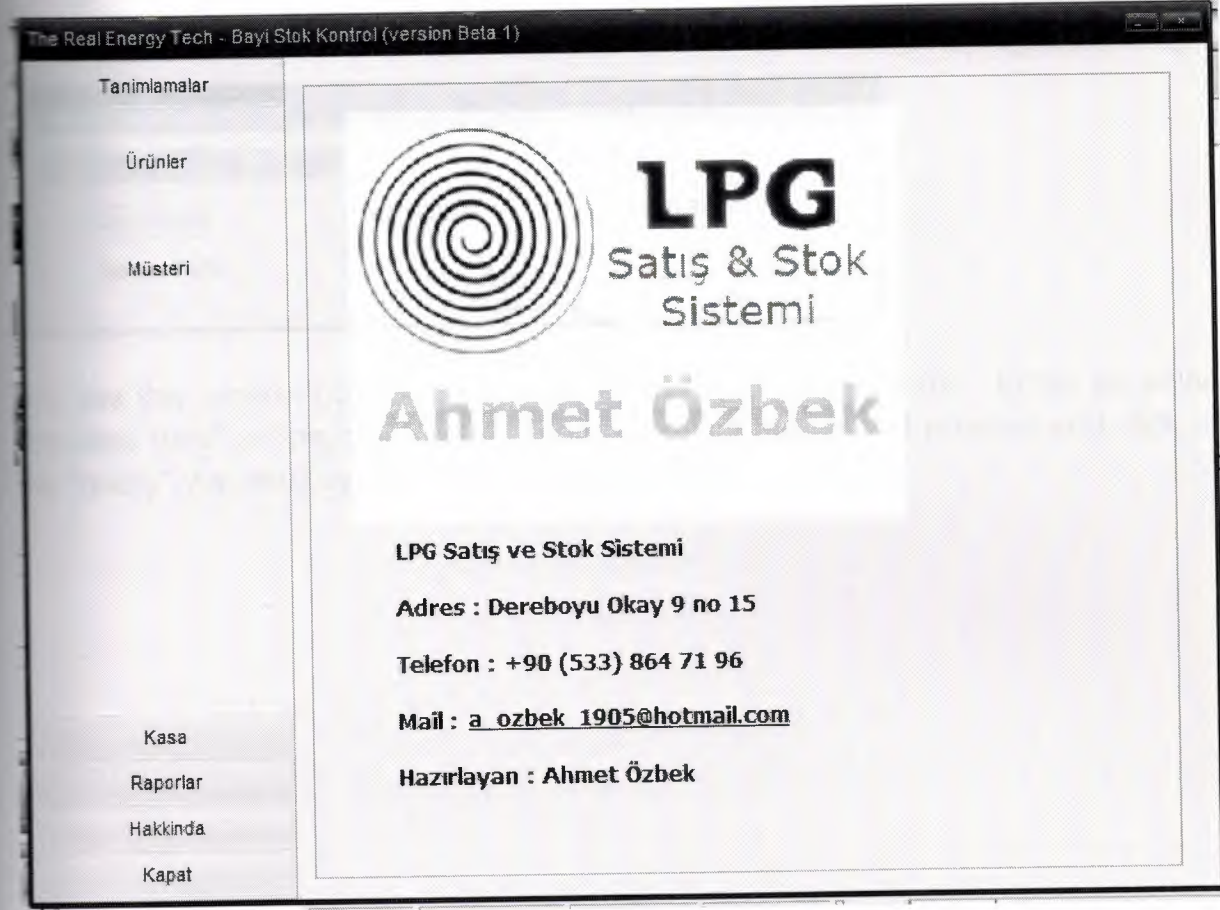
iptal

Yazdır

In this window we record the purchases. First we choose the product by searching with any information. And then we click the "confirm" button. Now we can enter the quantity, unit price and payment type. Then we click the confirm button twice. After all we select the payment date and payment situation.

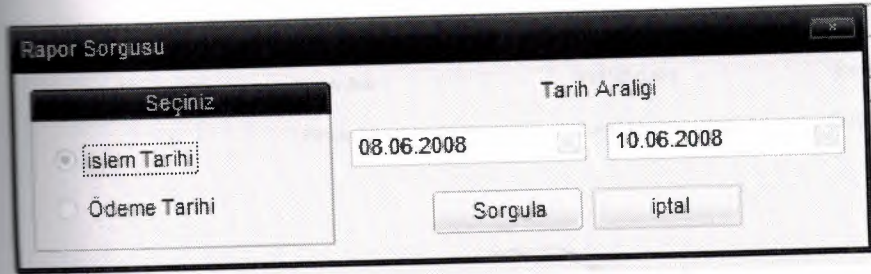


## About window



About window has the same view of the main window. Here we can see some texts about the program and programmer.

## Query window



Rapor Sorgusu

Seçiniz

☒ İşlem Tarihi

☐ Ödeme Tarihi

Tarih Aralığı

08.06.2008 10.06.2008

Sorgula İptal

We use this window both on “purchase reports” and “sale reports”. Either we select “process date” or “payment date”. Then we select the date of process and click on the “query”. As result system generates the report.

## Customers List

Müsteri Listesi				
Musteri No :	Firma Adi :	Yetkili Adi :	Telefon :	Faks :
1	Melek A.Ö	Burak Melek	5338662213	

The list of customers.



## Stocks List

Stok Listesi					
Stok No :	Stok Adi :	Aciklama :	Birim Fiyat (YTL) : Stok Miktari :		
1	Gapgaz 25 Kg	Ev Tüpü 25 kg	35	990	
2	LPG 1Lt	Litre ile Satıyb	1	100000	

## The list of Stocks

## CONCLUSION

TECHNOLOGY HAS AFFECTED THE REFERENCE and information culture in libraries. With the increasing scope of information transfer, users have higher service expectations of library and information science professionals. The emergence of a digital information environment has changed the century-old role of the reference professional. After the rise of the Internet, many skeptics foresaw the end of a need for librarians, particularly those working in traditional positions such as reference. Nevertheless, data from the Bureau of Labor Statistics indicates an increase in the number of information professionals by the year 2008. Reference professionals are becoming more--not less--essential. Graduate programs must examine the curriculum for reference and information access professionals. Greater access to information sources by users has highlighted the need for reference and information professionals to develop new skills including more technological knowledge, a better understanding of user information-seeking, new instructional techniques, and better communication skills. In addition to live classroom instruction, most schools offer reference and information access courses to a more diverse student body by employing distance-learning technologies.

## REFERENCES

- [1] Mastering Borland Delphi 2005 (Mastering) by Marco Cantu' (Paperback - Aug 19, 2005)
- [2] Inside Delphi 2006 (Wordware Delphi Developer's Library) by Ivan Hladni (Paperback - Nov 25, 2005)
- [3] Introducing Delphi Programming: Theory through Practice by John Barrow, Linda Miller, Katherine Malan, and Helene Gelderblom
- [4] [www.delphiturk.com](http://www.delphiturk.com)
- [5] <http://www.torry.net>
- [6] [www.about.com](http://www.about.com)



## APPENDIX

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, bsSkinData, BusinessSkinForm, bsSkinCtrls, bsSkinGrids,  
bsDBGrids, DB, ADODB, ComCtrls, bsSkinTabs, StdCtrls, Mask,  
bsSkinBoxCtrls, bsdbctrls, bsMessages, OleCtrls,  
ShockwaveFlashObjects\_TLB, ExtCtrls, jpeg;

type

TForm1 = class(TForm)

bsBusinessSkinForm1: TbsBusinessSkinForm;

bsSkinData1: TbsSkinData;

bsCompressedStoredSkin1: TbsCompressedStoredSkin;

bsSkinButtonsBar1: TbsSkinButtonsBar;  
  
bsSkinPageControl1: TbsSkinPageControl;  
  
bsSkinTabSheet1: TbsSkinTabSheet;  
  
bsSkinTabSheet2: TbsSkinTabSheet;  
  
ADOConnection1: TADOConnection;  
  
ADOTable1: TADOTable;  
  
DataSource1: TDataSource;  
  
bsSkinDBGrid1: TbsSkinDBGrid;  
  
bsSkinScrollBar1: TbsSkinScrollBar;  
  
bsSkinPanel1: TbsSkinPanel;  
  
bsSkinTextLabel1: TbsSkinTextLabel;  
  
bsSkinDBText1: TbsSkinDBText;  
  
bsSkinDBEdit1: TbsSkinDBEdit;  
  
bsSkinDBEdit2: TbsSkinDBEdit;  
  
bsSkinDBCurrencyEdit1: TbsSkinDBCurrencyEdit;  
  
bsSkinDBSpinEdit1: TbsSkinDBSpinEdit;  
  
bsSkinButton1: TbsSkinButton;  
  
bsSkinButton2: TbsSkinButton;  
  
bsSkinButton3: TbsSkinButton;  
  
bsSkinButton4: TbsSkinButton;

bsSkinButton5: TbsSkinButton;  
  
bsSkinStdLabel1: TbsSkinStdLabel;  
  
bsSkinComboBox1: TbsSkinComboBox;  
  
bsSkinEdit1: TbsSkinEdit;  
  
bsSkinPanel2: TbsSkinPanel;  
  
bsSkinTextLabel2: TbsSkinTextLabel;  
  
bsSkinDBText2: TbsSkinDBText;  
  
bsSkinDBEdit3: TbsSkinDBEdit;  
  
bsSkinDBEdit4: TbsSkinDBEdit;  
  
bsSkinButton6: TbsSkinButton;  
  
bsSkinButton8: TbsSkinButton;  
  
bsSkinButton9: TbsSkinButton;  
  
bsSkinButton10: TbsSkinButton;  
  
bsSkinComboBox2: TbsSkinComboBox;  
  
bsSkinStdLabel2: TbsSkinStdLabel;  
  
bsSkinEdit2: TbsSkinEdit;  
  
bsSkinDBGrid2: TbsSkinDBGrid;  
  
bsSkinMessage1: TbsSkinMessage;  
  
bsSkinButton7: TbsSkinButton;  
  
ADOTable2: TADOTable;



DataSource2: TDataSource;

bsSkinStdLabel3: TbsSkinStdLabel;

bsSkinDBMemo1: TbsSkinDBMemo;

bsSkinStdLabel4: TbsSkinStdLabel;

bsSkinDBMemo2: TbsSkinDBMemo;

bsSkinDBEdit5: TbsSkinDBEdit;

bsSkinDBEdit6: TbsSkinDBEdit;

bsSkinPageControl2: TbsSkinPageControl;

bsSkinTabSheet3: TbsSkinTabSheet;

bsSkinTabSheet4: TbsSkinTabSheet;

bsSkinPanel3: TbsSkinPanel;

bsSkinPanel4: TbsSkinPanel;

bsSkinPanel5: TbsSkinPanel;

bsSkinTextLabel3: TbsSkinTextLabel;

bsSkinStdLabel5: TbsSkinStdLabel;

bsSkinComboBox3: TbsSkinComboBox;

bsSkinEdit3: TbsSkinEdit;

bsSkinDBText3: TbsSkinDBText;

bsSkinDBText4: TbsSkinDBText;

bsSkinDBText5: TbsSkinDBText;

bsSkinDBText6: TbsSkinDBText;  
  
bsSkinDBText7: TbsSkinDBText;  
  
bsSkinButton11: TbsSkinButton;  
  
ADOTable3: TADOTable;  
  
DataSource3: TDataSource;  
  
bsSkinDBSpinEdit2: TbsSkinDBSpinEdit;  
  
bsSkinStdLabel6: TbsSkinStdLabel;  
  
bsSkinDBCurrencyEdit2: TbsSkinDBCurrencyEdit;  
  
bsSkinStdLabel7: TbsSkinStdLabel;  
  
bsSkinDBCurrencyEdit3: TbsSkinDBCurrencyEdit;  
  
bsSkinDBComboBox1: TbsSkinDBComboBox;  
  
bsSkinStdLabel8: TbsSkinStdLabel;  
  
bsSkinStdLabel9: TbsSkinStdLabel;  
  
bsSkinDBMemo3: TbsSkinDBMemo;  
  
bsSkinButton12: TbsSkinButton;  
  
bsSkinButton13: TbsSkinButton;  
  
bsSkinButton14: TbsSkinButton;  
  
bsSkinButton15: TbsSkinButton;  
  
ADOTable4: TADOTable;  
  
DataSource4: TDataSource;

bsSkinDBDateEdit1: TbsSkinDBDateEdit;  
  
bsSkinStdLabel10: TbsSkinStdLabel;  
  
bsSkinDBComboBox2: TbsSkinDBComboBox;  
  
bsSkinStdLabel11: TbsSkinStdLabel;  
  
bsSkinButton16: TbsSkinButton;  
  
bsSkinButton17: TbsSkinButton;  
  
bsSkinPanel8: TbsSkinPanel;  
  
bsSkinStdLabel12: TbsSkinStdLabel;  
  
bsSkinStdLabel13: TbsSkinStdLabel;  
  
bsSkinButton18: TbsSkinButton;  
  
bsSkinDBDateEdit2: TbsSkinDBDateEdit;  
  
bsSkinDBComboBox3: TbsSkinDBComboBox;  
  
bsSkinButton19: TbsSkinButton;  
  
bsSkinButton20: TbsSkinButton;  
  
bsSkinPanel7: TbsSkinPanel;  
  
bsSkinStdLabel14: TbsSkinStdLabel;  
  
bsSkinStdLabel15: TbsSkinStdLabel;  
  
bsSkinStdLabel16: TbsSkinStdLabel;  
  
bsSkinStdLabel17: TbsSkinStdLabel;  
  
bsSkinDBSpinEdit3: TbsSkinDBSpinEdit;



bsSkinDBCurrencyEdit4: TbsSkinDBCurrencyEdit;  
bsSkinDBCurrencyEdit5: TbsSkinDBCurrencyEdit;  
bsSkinDBComboBox4: TbsSkinDBComboBox;  
bsSkinDBMemo4: TbsSkinDBMemo;  
bsSkinButton21: TbsSkinButton;  
bsSkinButton22: TbsSkinButton;  
bsSkinButton23: TbsSkinButton;  
bsSkinPanel6: TbsSkinPanel;  
bsSkinTextLabel4: TbsSkinTextLabel;  
bsSkinStdLabel18: TbsSkinStdLabel;  
bsSkinDBText8: TbsSkinDBText;  
bsSkinDBText9: TbsSkinDBText;  
bsSkinDBText10: TbsSkinDBText;  
bsSkinDBText11: TbsSkinDBText;  
bsSkinDBText12: TbsSkinDBText;  
bsSkinComboBox4: TbsSkinComboBox;  
bsSkinEdit4: TbsSkinEdit;  
bsSkinButton24: TbsSkinButton;  
ADOTable5: TADOTable;  
DataSource5: TDataSource;

DataSource6: TDataSource;

ADOTable6: TADOTable;

ADOTable7: TADOTable;

DataSource7: TDataSource;

bsSkinDBLookupComboBox1: TbsSkinDBLookupComboBox;

bsSkinStdLabel19: TbsSkinStdLabel;

bsSkinPanel9: TbsSkinPanel;

bsSkinTextLabel5: TbsSkinTextLabel;

bsSkinLinkLabel2: TbsSkinLinkLabel;

bsSkinButton25: TbsSkinButton;

Image1: TImage;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure bsSkinButton1Click(Sender: TObject);

procedure bsSkinButton2Click(Sender: TObject);

procedure bsSkinButton4Click(Sender: TObject);

procedure bsSkinButton5Click(Sender: TObject);

procedure bsSkinButton3Click(Sender: TObject);

procedure bsSkinComboBox1Change(Sender: TObject);

procedure bsSkinEdit1Change(Sender: TObject);

```
procedure bsSkinButton6Click(Sender: TObject);

procedure bsSkinButton7Click(Sender: TObject);

procedure bsSkinButton8Click(Sender: TObject);

procedure bsSkinButton9Click(Sender: TObject);

procedure bsSkinButton10Click(Sender: TObject);

procedure bsSkinComboBox2Change(Sender: TObject);

procedure bsSkinEdit2Change(Sender: TObject);

procedure bsSkinButtonsBar1Sections0Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections0Items1Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections1Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections1Items1Click(Sender: TObject);

procedure bsSkinComboBox3Change(Sender: TObject);

procedure bsSkinEdit3Change(Sender: TObject);

procedure bsSkinButton11Click(Sender: TObject);

procedure bsSkinDBSpinEdit2Change(Sender: TObject);

procedure bsSkinDBCurrencyEdit2Change(Sender: TObject);

procedure bsSkinButton12Click(Sender: TObject);

procedure bsSkinButton13Click(Sender: TObject);

procedure bsSkinButton14Click(Sender: TObject);

procedure bsSkinButton16Click(Sender: TObject);
```



```

procedure bsSkinButton17Click(Sender: TObject);

procedure bsSkinComboBox4Change(Sender: TObject);

procedure bsSkinEdit4Change(Sender: TObject);

procedure bsSkinButton24Click(Sender: TObject);

procedure bsSkinDBSpinEdit3Change(Sender: TObject);

procedure bsSkinButton21Click(Sender: TObject);

procedure bsSkinButton22Click(Sender: TObject);

procedure bsSkinButton23Click(Sender: TObject);

procedure bsSkinButton19Click(Sender: TObject);

procedure bsSkinDBCurrencyEdit4Change(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items3Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items2Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items0Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections2Items1Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections3Click(Sender: TObject);

procedure bsSkinButtonsBar1Sections4Click(Sender: TObject);

procedure bsSkinButton25Click(Sender: TObject);

private
{ Private declarations }

public

```

```

{ Public declarations }

end;

var
    Form1: TForm1;

implementation

uses Unit3, Unit4, Unit5, Unit2;

{$R *.dfm}

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if bsskinmessage1.MessageDlg('Programi kapatmak istediginize emin misiniz
    ?',mtconfirmation,[mbyes,mbno],0)=mryes then

        application.Terminate

    else

        application.Run;

        form2.hide;

end;

```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
tmpstr:string;
```

```
begin
```

```
  TmpStr:=ExtractFileDir(ParamStr(0));
```

```
  if TmpStr[Length(TmpStr)]<>'\' Then TmpStr:=TmpStr+'\';
```

```
  adoconnection1.Connected:=false;
```

```
  adoconnection1.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data  
  Source='+tmpstr+'db.mdb;Persist Security Info=False';
```

```
  adoconnection1.Connected:=true;
```

```
  adotable1.Active:=true;
```

```
  adotable2.Active:=true;
```

```
  adotable3.Active:=true;
```

```
  adotable4.Active:=true;
```

```
  adotable5.Active:=true;
```

```
  adotable6.Active:=true;
```

```
  adotable7.Active:=true;
```

```
  // stok grid duzenlemeler //
```

```
  bsskindbgrid1.Columns[0].Visible:=false;
```



```
bsskindbgrid1.Columns[1].Title.caption:='Ürün Adı';  
  
bsskindbgrid1.Columns[1].Width:=100;  
  
bsskindbgrid1.Columns[2].Title.caption:='Açıklama';  
  
bsskindbgrid1.Columns[2].Width:=150;  
  
bsskindbgrid1.Columns[3].Title.caption:='Birim Fiyat';  
  
bsskindbgrid1.Columns[3].Width:=90;  
  
bsskindbgrid1.Columns[4].Title.caption:='Stok Miktar';  
  
bsskindbgrid1.Columns[4].Width:=90;  
  
// muster grid duzenlemeler //  
  
bsskindbgrid2.Columns[0].Visible:=false;  
  
bsskindbgrid2.Columns[1].Title.caption:='Yetkili Adı';  
  
bsskindbgrid2.Columns[1].Width:=150;  
  
bsskindbgrid2.Columns[2].Title.caption:='Firma Adı';  
  
bsskindbgrid2.Columns[2].Width:=100;  
  
bsskindbgrid2.Columns[3].Title.caption:='Telefon';  
  
bsskindbgrid2.Columns[3].Width:=90;  
  
bsskindbgrid2.Columns[4].Title.caption:='Faks';  
  
bsskindbgrid2.Columns[4].Width:=90;  
  
bsskindbgrid2.Columns[5].Visible:=false;  
  
bsskindbgrid2.Columns[6].Visible:=false;
```

end;

procedure TForm1.bsSkinButton1Click(Sender: TObject);

begin

adotable1.Insert;

bsskinbutton1.Enabled:=false;

bsskinbutton2.Enabled:=false;

bsskinbutton4.Enabled:=false;

bsskinbutton3.Enabled:=true;

bsskinbutton5.Enabled:=true;

bsskinpanel1.Enabled:=true;

end;

procedure TForm1.bsSkinButton2Click(Sender: TObject);

begin

adotable1.Edit;

bsskinbutton1.Enabled:=false;

bsskinbutton2.Enabled:=false;

bsskinbutton4.Enabled:=false;

```
bsskinbutton3.Enabled:=true;
```

```
bsskinbutton5.Enabled:=true;
```

```
bsskinpanel1.Enabled:=true;
```

```
end;
```

```
procedure TForm1.bsSkinButton4Click(Sender: TObject);
```

```
begin
```

```
if bsskinmessage1.MessageDlg('Kaydi silmek istediginize emin misiniz  
'?,mtwarning,[mbytes,mbno],0) = mryes then
```

```
adotable1.Delete;
```

```
end;
```

```
procedure TForm1.bsSkinButton5Click(Sender: TObject);
```

```
begin
```

```
if bsskinmessage1.MessageDlg('Kaydi iptal etmek istediginize emin misiniz  
'?,mtwarning,[mbytes,mbno],0) = mryes then begin
```

```
adotable1.cancel;
```

```
bsskinbutton1.Enabled:=true;
```

```
bsskinbutton2.Enabled:=true;
```

```
bsskinbutton4.Enabled:=true;
```

```
bsskinbutton3.Enabled:=false;
```



```
bsskinbutton5.Enabled:=false;
```

```
bsskinpanel1.Enabled:=false;
```

```
end;
```

```
end;
```

```
procedure TForm1.bsSkinButton3Click(Sender: TObject);
```

```
begin
```

```
if (bsskindbedit1.Text='') or (bsskindbedit2.Text='') or  
(bsskindbcurrencyedit1.Value<=0) or (bsskindbspinedit1.Value<=0) then
```

```
bsskinmessage1.MessageDlg('Tüm verilerinizin tutarlılığını kontrol  
ediniz',mtwarning,[mbok],0)
```

```
else begin
```

```
adotable1.post;
```

```
bsskinbutton1.Enabled:=true;
```

```
bsskinbutton2.Enabled:=true;
```

```
bsskinbutton4.Enabled:=true;
```

```
bsskinbutton3.Enabled:=false;
```

```
bsskinbutton5.Enabled:=false;
```

```
bsskinpanel1.Enabled:=false;
```

```
end;
```

```
end;
```

```
procedure TForm1.bsSkinComboBox1Change(Sender: TObject);
```

```
begin
```

```
if bsskinedit1.Text<>'' then begin
```

```
if bssskincombobox1.ItemIndex=0 then
```

```
begin
```

```
adotable1.Filtered:=false;
```

```
adotable1.Filter:='s_no='+bsskinedit1.Text;
```

```
adotable1.Filtered:=true;
```

```
end;
```

```
if bssskincombobox1.ItemIndex=1 then
```

```
begin
```

```
adotable1.Filtered:=false;
```

```
adotable1.Filter:='Adi like '+#39+bsskinedit1.Text+'%'+#39;
```

```
adotable1.Filtered:=true;
```

```
end;
```

```
end;
```

```
end;
```

```
procedure TForm1.bsSkinEdit1Change(Sender: TObject);
```

```
begin
```

```

if bsskinedit1.Text<>" then begin

if bssskincombobox1.ItemIndex=0 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='s_no='+bsskinedit1.Text;

adotable1.Filtered:=true;

end;

if bssskincombobox1.ItemIndex=1 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='Adi like '+#39+bsskinedit1.Text+'%'+#39;

adotable1.Filtered:=true;

end;

end;

end;

procedure TForm1.bsSkinButton6Click(Sender: TObject);

begin

adotable2.Insert;

bssskinbutton6.Enabled:=false;

bssskinbutton7.Enabled:=false;

```



```
bsskinbutton8.Enabled:=false;
```

```
bsskinbutton9.Enabled:=true;
```

```
bsskinbutton10.Enabled:=true;
```

```
bsskinpanel2.Enabled:=true;
```

```
end;
```

```
procedure TForm1.bsSkinButton7Click(Sender: TObject);
```

```
begin
```

```
adotable2.Edit;
```

```
bsskinbutton6.Enabled:=false;
```

```
bsskinbutton7.Enabled:=false;
```

```
bsskinbutton8.Enabled:=false;
```

```
bsskinbutton9.Enabled:=true;
```

```
bsskinbutton10.Enabled:=true;
```

```
bsskinpanel2.Enabled:=true;
```

```
end;
```

```
procedure TForm1.bsSkinButton8Click(Sender: TObject);
```

```
begin
```

```
if bsskinmessage1.MessageDlg('Kaydi silmek istedinize emin misiniz  
' ,mtwarning,[mbyes,mbno],0) = mryes then
```

```
adotable2.Delete;
```

```
end;
```

```
procedure TForm1.bsSkinButton9Click(Sender: TObject);
```

```
begin
```

```
if (bsskindbedit3.Text='') or (bsskindbedit4.Text='') or (bsskindbedit5.text='') or  
(bsskindbmemo1.Text='') then
```

```
bsskinmessage1.MessageDlg('Tüm verilerinizin tutarlılığını kontrol  
ediniz',mtwarning,[mbok],0)
```

```
else begin
```

```
adotable2.post;
```

```
bsskinbutton6.Enabled:=true;
```

```
bsskinbutton7.Enabled:=true;
```

```
bsskinbutton8.Enabled:=true;
```

```
bsskinbutton9.Enabled:=false;
```

```
bsskinbutton10.Enabled:=false;
```

```
bsskinpanel2.Enabled:=false;
```

```
end;
```

```
end;
```

```
procedure TForm1.bsSkinButton10Click(Sender: TObject);
```

begin

if bsskinmessage1.MessageDlg('Kaydi iptal etmek istediginize emin misiniz  
' ,mtwarning,[mbyes,mbno],0) = mryes then begin

adotable2.cancel;

bsskinbutton6.Enabled:=true;

bsskinbutton7.Enabled:=true;

bsskinbutton8.Enabled:=true;

bsskinbutton9.Enabled:=false;

bsskinbutton10.Enabled:=false;

bsskinpanel2.Enabled:=false;

end;

end;

procedure TForm1.bsSkinComboBox2Change(Sender: TObject);

begin

if bsskinedit2.Text<>'' then begin

if bsskincombobox2.ItemIndex=0 then

begin

adotable2.Filtered:=false;

adotable2.Filter:='[Firma Adi] like '+#39+bsskinedit2.Text+'%'+#39;

adotable2.Filtered:=true;



end;

if bsskincombobox2.ItemIndex=1 then

begin

adotable2.Filtered:=false;

adotable2.Filter:='Adi\_Soyadi like '+#39+bsskinedit2.Text+'%'+#39;

adotable2.Filtered:=true;

end;

end;

end;

procedure TForm1.bsSkinEdit2Change(Sender: TObject);

begin

if bsskinedit2.Text<>'' then begin

if bsskincombobox2.ItemIndex=0 then

begin

adotable2.Filtered:=false;

adotable2.Filter:='[Firma Adi] like '+#39+bsskinedit2.Text+'%'+#39;

adotable2.Filtered:=true;

end;

if bsskincombobox2.ItemIndex=1 then

begin

adotable2.Filtered:=false;

adotable2.Filter:='Adi\_Soyadi like '+#39+bsskinedit2.Text+'%'+#39;

adotable2.Filtered:=true;

end;

end;

end;

begin

procedure TForm1.bsSkinButtonsBar1Sections0Items0Click(Sender: TObject);

begin

bsskinpagecontrol1.BringToFront;

bsskinpagecontrol1.TabIndex:=0;

end;

begin

procedure TForm1.bsSkinButtonsBar1Sections0Items1Click(Sender: TObject);

begin

bsskinpagecontrol1.BringToFront;

bsskinpagecontrol1.TabIndex:=1;

end;

```
procedure TForm1.bsSkinButtonsBar1Sections1Items0Click(Sender: TObject);  
  
begin  
  
bsskinpagecontrol2.BringToFront;  
  
bsskinpagecontrol2.TabIndex:=0;  
  
end;
```

```
procedure TForm1.bsSkinButtonsBar1Sections1Items1Click(Sender: TObject);  
  
begin  
  
bsskinpagecontrol2.BringToFront;  
  
bsskinpagecontrol2.TabIndex:=1;  
  
end;
```

```
procedure TForm1.bsSkinComboBox3Change(Sender: TObject);  
  
begin  
  
if bsskinedit3.Text<>" then begin  
  
if bsskincombobox3.ItemIndex=0 then  
  
begin  
  
adotable1.Filtered:=false;  
  
adotable1.Filter:='s_no='+bsskinedit3.Text;  
  
adotable1.Filtered:=true;
```



```

end;

if bsskincombobox3.ItemIndex=1 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='Adi like '+#39+bsskinedit3.Text+'%'+#39;

adotable1.Filtered:=true;

end;

end;

end;

```

```

procedure TForm1.bsSkinEdit3Change(Sender: TObject);

```

```

begin

if bsskinedit3.Text<>" then begin

if bsskincombobox3.ItemIndex=0 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='s_no='+bsskinedit3.Text;

adotable1.Filtered:=true;

end;

if bsskincombobox3.ItemIndex=1 then

```

begin

adotable1.Filtered:=false;

adotable1.Filter:='Adi like '+#39+bsskinedit3.Text+'%'+#39;

adotable1.Filtered:=true;

end;

end;

end;

procedure TForm1.bsSkinButton11Click(Sender: TObject);

begin

if bsskindbtext3.Caption="" then

bsskinmessage1.MessageDlg('Ürün Seçmediniz',mtwarning,[mbok],0)

else begin

bsskinpanel3.Enabled:=false;

bsskinpanel4.Enabled:=true;

adotable3.Insert;

end;

end;

procedure TForm1.bsSkinDBSpinEdit2Change(Sender: TObject);

```
begin
```

```
if (bsskindbspinedit2.Value>0) and (bsskindbcurrencyedit2.Value>0) then
```

```
bsskindbcurrencyedit3.Value:=bsskindbspinedit2.Value*bsskindbcurrencyedit2.Value
```

```
;
```

```
end;
```

```
procedure TForm1.bsSkinDBCurrencyEdit2Change(Sender: TObject);
```

```
begin
```

```
if (bsskindbspinedit2.Value>0) and (bsskindbcurrencyedit2.Value>0) then
```

```
bsskindbcurrencyedit3.Value:=bsskindbspinedit2.Value*bsskindbcurrencyedit2.Value
```

```
;
```

```
end;
```

```
procedure TForm1.bsSkinButton12Click(Sender: TObject);
```

```
var
```

```
a:integer;
```

```
begin
```

```
datasource3.DataSet.FieldByName('Tarih').AsDateTime:=now;
```

```
datasource3.DataSet.FieldByName('Saat').AsDateTime:=now;
```

```
datasource3.DataSet.FieldByName('Stok_adi').AsString:=bsskindbtext4.Caption;
```

```

datasource3.DataSet.FieldByName('Toplam_tutar').AsFloat:=bsskindbcurrencyedit3.
value;

datasource3.DataSet.FieldByName('Alis_Miktari').AsInteger:=strtoint(bsskindbspinedi
t2.text);

adotable3.Post;

bsskinpanel4.Enabled:=false;

a:=datasource1.DataSet.FieldByName('Stok_Miktar').AsInteger;

adotable1.Edit;

a:=a+strtoint(bsskindbspinedit2.text);

datasource1.DataSet.FieldByName('Stok_Miktar').AsInteger:=a;

adotable1.Post;

bsskinpanel5.Enabled:=true;

adotable4.Insert;

end;

procedure TForm1.bsSkinButton13Click(Sender: TObject);

begin

adotable3.Cancel;

end;

procedure TForm1.bsSkinButton14Click(Sender: TObject);

```



begin

adotable3.Cancel;

bsskinpanel4.Enabled:=false;

bsskinpanel5.Enabled:=false;

bsskinpanel3.Enabled:=true;

end;

begin

procedure TForm1.bsSkinButton16Click(Sender: TObject);

begin

datasource4.DataSet.FieldName('Alis\_No').AsString:=datasource3.DataSet.FieldName('Alis\_No').AsString;

datasource4.DataSet.FieldName('Verecek\_Tutar').AsFloat:=datasource3.DataSet.FieldName('Toplam\_Tutar').AsFloat;

datasource4.DataSet.FieldName('Odeme\_Turu').AsString:=datasource3.DataSet.FieldName('Odeme\_turu').AsString;

datasource4.DataSet.FieldName('Odeme\_Tarihi').AsDateTime:=bsskindbdateedit1.Date;

datasource3.DataSet.FieldName('Alis\_No').AsString;

datasource4.DataSet.FieldName('Durum').AsString:=bsskindbcombobox2.Text;

adotable4.Post;

bsskinpanel5.Enabled:=false;

bsskinpanel4.Enabled:=false;

```
bsskinpanel3.Enabled:=true;
```

```
bsskinmessage1.MessageDlg('islem basariyla tamamlandi',mtinformation,[mbok],0 );
```

```
bsskincombobox3.SetFocus;
```

```
end;
```

```
procedure TForm1.bsSkinButton17Click(Sender: TObject);
```

```
begin
```

```
adotable4.Cancel;
```

```
bsskinpanel5.Enabled:=false;
```

```
bsskinpanel4.Enabled:=true;
```

```
end;
```

```
procedure TForm1.bsSkinComboBox4Change(Sender: TObject);
```

```
begin
```

```
if bsskinedit4.Text<>" then begin
```

```
if bsskincombobox4.ItemIndex=0 then
```

```
begin
```

```
adotable1.Filtered:=false;
```

```
adotable1.Filter:='s_no='+bsskinedit4.Text;
```

```
adotable1.Filtered:=true;
```

```

end;

if bsskincombobox4.ItemIndex=1 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='Adi like '+#39+bsskinedit4.Text+'%'+#39;

adotable1.Filtered:=true;

end;

end;

end;

```

```

procedure TForm1.bsSkinEdit4Change(Sender: TObject);

begin

if bsskinedit4.Text<>'' then begin

if bsskincombobox4.ItemIndex=0 then

begin

adotable1.Filtered:=false;

adotable1.Filter:='s_no='+bsskinedit4.Text;

adotable1.Filtered:=true;

end;

if bsskincombobox4.ItemIndex=1 then

```

```

begin

adotable1.Filtered:=false;

adotable1.Filter:='Adi like '+#39+bsskinedit4.Text+'%'+#39;

adotable1.Filtered:=true;

end;

end;

end;

end;

procedure TForm1.bsSkinButton24Click(Sender: TObject);
begin
if (bsskindbtext8.Caption='') or (bsskindblookupcombobox1.Text='') then
bsskinmessage1.MessageDlg('Ürün veya Müsteri Seçmediniz',mtwarning,[mbok],0)
else begin
bsskinpanel6.Enabled:=false;
bsskinpanel7.Enabled:=true;

adotable6.Insert;

end;

end;

procedure TForm1.bsSkinDBSpinEdit3Change(Sender: TObject);

```



```

begin
if (bsskindbspinedit3.Value>0) and (bsskindbcurrencyedit4.Value>0) then

bsskindbcurrencyedit5.Value:=bsskindbspinedit3.Value*bsskindbcurrencyedit4.Value
;
end;

procedure TForm1.bsSkinButton21Click(Sender: TObject);

var
a:integer;
begin
datasource6.DataSet.FieldByName('Tarih').AsDateTime:=now;

datasource6.DataSet.FieldByName('Saat').AsDateTime:=now;

datasource6.DataSet.FieldByName('Stok_adi').AsString:=bsskindbtext9.Caption;

datasource6.DataSet.FieldByName('Toplam_tutar').AsFloat:=bsskindbcurrencyedit5.
value;

datasource6.DataSet.FieldByName('Satis_Miktari').AsInteger:=strtoint(bsskindbspine
dit3.text);

adotable6.Post;

bssskinpanel7.Enabled:=false;

a:=datasource1.DataSet.FieldByName('Stok_Miktar').AsInteger;

```

```
adotable1.Edit;
```

```
a:=a-strtoint(bsskindbspinedit3.text);
```

```
datasource1.DataSet.FieldByName('Stok_Miktar').AsInteger:=a;
```

```
adotable1.Post;
```

```
bsskinpanel8.Enabled:=true;
```

```
adotable5.Insert;
```

```
end;
```

```
procedure TForm1.bsSkinButton22Click(Sender: TObject);
```

```
begin
```

```
adotable6.Cancel;
```

```
end;
```

```
procedure TForm1.bsSkinButton23Click(Sender: TObject);
```

```
begin
```

```
adotable6.Cancel;
```

```
bsskinpanel7.Enabled:=false;
```

```
bsskinpanel8.Enabled:=false;
```

```
bsskinpanel6.Enabled:=true;
```

```
end;
```

```

procedure TForm1.bsSkinButton19Click(Sender: TObject);

begin

datasource5.DataSet.FieldByName('Satis_No').AsString:=datasource6.DataSet.Field
ByName('Satis_No').AsString;

datasource5.DataSet.FieldByName('Alacak_Tutar').AsFloat:=datasource6.DataSet.Fi
eldByName('Toplam_Tutar').AsFloat;

datasource5.DataSet.FieldByName('Odeme_Turu').AsString:=datasource6.DataSet.
FieldByName('Odeme_turu').AsString;

datasource5.DataSet.FieldByName('Odeme_Tarihi').AsDateTime:=bsskindbdateedit2
.Date;

datasource5.DataSet.FieldByName('Durum').AsString:=bsskindbcombobox3.Text;

adotable5.Post;

bsskinpanel8.Enabled:=false;

bsskinpanel7.Enabled:=false;

bsskinpanel6.Enabled:=true;

bsskinmessage1.MessageDlg('islem basariyla tamamlandi',mtinformation,[mbok],0 );

bsskincombobox4.SetFocus;

end;

procedure TForm1.bsSkinDBCurrencyEdit4Change(Sender: TObject);

begin

```

```

if (bsskindbspinedit3.Value>0) and (bsskindbcurrencyedit4.Value>0) then

bsskindbcurrencyedit5.Value:=bsskindbspinedit3.Value*bsskindbcurrencyedit4.Value
;

end;

procedure TForm1.bsSkinButtonsBar1Sections2Items3Click(Sender: TObject);
begin

form4.showmodal;

end;

procedure TForm1.bsSkinButtonsBar1Sections2Items2Click(Sender: TObject);
begin

form5.ShowModal;

end;

procedure TForm1.bsSkinButtonsBar1Sections2Items0Click(Sender: TObject);
begin

form3.QuickRep4.ReportTitle:='Mevcut Stok Durumu';

form3.QuickRep4.Preview;

end;

```



```
procedure TForm1.bsSkinButtonsBar1Sections2Items1Click(Sender: TObject);
```

```
begin
```

```
form3.QuickRep3.ReportTitle:='Müşteri Listesi';
```

```
form3.QuickRep3.Preview;
```

```
end;
```

```
procedure TForm1.bsSkinButtonsBar1Sections3Click(Sender: TObject);
```

```
begin
```

```
bsskinpanel9.BringToFront;
```

```
end;
```

```
procedure TForm1.bsSkinButtonsBar1Sections4Click(Sender: TObject);
```

```
begin
```

```
if bsskinmessage1.MessageDlg('Programı kapatmak istedinize emin misiniz  
' ,mtconfirmation,[mbyes,mbno],0)=mryes then
```

```
application.Terminate;
```

```
end;
```

```
procedure TForm1.bsSkinButton25Click(Sender: TObject);
```

```

begin

with form3.adoquery1 do begin

close;

sql.Clear;

sql.Add('SELECT    Alacak.*,    Satis.*    FROM    Alacak,    Satis    Where
Alacak.Satis_No=Satis.Satis_No');

sql.Add(' and Musteri_Adi='+#39+bsskindbtext2.caption+#39);

open;

end;

form3.QuickRep1.ReportTitle:=bsskindbedit3.text+' adli firmaya yapilan satis raporu';

form3.QuickRep1.Preview;

end;

end.

```

```

unit Unit2;

```

```

interface

```

```

uses

```

```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

```

Dialogs, jpeg, ExtCtrls;

type

TForm2 = class(TForm)

Timer1: TTimer;

Image1: TImage;

procedure FormCreate(Sender: TObject);

procedure Timer1Timer(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form2: TForm2;

implementation

uses Unit1;

var

i:integer;

{ \$R \*.dfm }

end

procedure TForm2.FormCreate(Sender: TObject);

begin

timer1.Enabled:=true;

i:=0;

end;

View in the IDE: System -> Forms -> Graphics, Controls, Forms.

procedure TForm2.Timer1Timer(Sender: TObject);

begin

if i=150 then begin

form1.show;

form1.Enabled:=true;

form2.Hide;

timer1.Enabled:=false;

end

else

i:=i+1;



end;

end.

unit Unit4;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, bsSkinCtrls, StdCtrls, Mask, bsSkinBoxCtrls, BusinessSkinForm;

type

TForm4 = class(TForm)

bsBusinessSkinForm1: TbsBusinessSkinForm;

bsSkinGroupBox1: TbsSkinGroupBox;

bsSkinCheckRadioBox1: TbsSkinCheckRadioBox;

bsSkinCheckRadioBox2: TbsSkinCheckRadioBox;

bsSkinDateEdit1: TbsSkinDateEdit;

bsSkinDateEdit2: TbsSkinDateEdit;

```

bsSkinButton1: TbsSkinButton;

bsSkinButton2: TbsSkinButton;

bsSkinStdLabel2: TbsSkinStdLabel;

procedure bsSkinButton1Click(Sender: TObject);

procedure bsSkinButton2Click(Sender: TObject);

procedure FormShow(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form4: TForm4;

implementation

uses unit1, Unit3;

{$R *.dfm}

procedure TForm4.bsSkinButton1Click(Sender: TObject);

```

```

var

tar1,tar2:string;

begin

tar1:=formatdatetime('dd-mm-yyyy',bsskindateedit1.Date);

tar2:=formatdatetime('dd-mm-yyyy',bsskindateedit2.Date);

if bsskincheckradiobox1.Checked=true then begin

with form3.adoquery1 do begin

close;

sql.Clear;

sql.Add('SELECT    Alacak.*,    Satis.*    FROM    Alacak,    Satis    Where
Alacak.Satis_No=Satis.Satis_No');

sql.Add(' and Tarih between #'+tar1+'# and #'+tar2+'#');

open;

end;

end

else begin

with form3.adoquery1 do begin

close;

sql.Clear;

sql.Add('SELECT    Alacak.*,    Satis.*    FROM    Alacak,    Satis    Where
Alacak.Satis_No=Satis.Satis_No');

```

```
sql.Add(' and Odeme_Tarihi between #'+tar1+'# and #'+tar2+'#');
```

```
open;
```

```
end;
```

```
end;
```

```
form3.QuickRep1.ReportTitle:=tar1+' ve '+tar2+' tarihleri arasindaki satis raporu';
```

```
form3.QuickRep1.Preview;
```

```
end;
```

```
procedure TForm4.bsSkinButton2Click(Sender: TObject);
```

```
begin
```

```
form4.Close;
```

```
end;
```

```
procedure TForm4.FormShow(Sender: TObject);
```

```
begin
```

```
bsskindateedit1.Date:=now-1;
```

```
bsskindateedit2.Date:=now+1;
```

```
end;
```

```
end.
```



unit Unit5;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, BusinessSkinForm, bsSkinCtrls, StdCtrls, Mask, bsSkinBoxCtrls;

type

TForm5 = class(TForm)

bsSkinStdLabel2: TbsSkinStdLabel;

bsSkinGroupBox1: TbsSkinGroupBox;

bsSkinCheckRadioBox1: TbsSkinCheckRadioBox;

bsSkinCheckRadioBox2: TbsSkinCheckRadioBox;

bsSkinDateEdit1: TbsSkinDateEdit;

bsSkinDateEdit2: TbsSkinDateEdit;

bsSkinButton1: TbsSkinButton;

bsSkinButton2: TbsSkinButton;

bsBusinessSkinForm1: TbsBusinessSkinForm;

procedure bsSkinButton1Click(Sender: TObject);

```

procedure FormShow(Sender: TObject);

procedure bsSkinButton2Click(Sender: TObject);

private

    { Private declarations }

public

    { Public declarations }

end;

var

    Form5: TForm5;

implementation

    uses unit1, Unit3;

    {$R *.dfm}

    procedure TForm5.bsSkinButton1Click(Sender: TObject);

    var

        tar1,tar2:string;

    begin

        tar1:=formatdatetime('dd-mm-yyyy',bsskindateedit1.Date);

```

```
tar2:=formatdatetime('dd-mm-yyyy',bsskindateedit2.Date);
```

```
if bsskincheckradiobox1.Checked=true then begin
```

```
with form3.adoquery2 do begin
```

```
close;
```

```
sql.Clear;
```

```
sql.Add('SELECT Verecek.*, Alis.* FROM Verecek, Alis Where  
Verecek.Alis_No=Alis.Alis_No');
```

```
sql.Add(' and Tarih between #'+tar1+'# and #'+tar2+'#');
```

```
open;
```

```
end;
```

```
end
```

```
else begin
```

```
with form3.adoquery2 do begin
```

```
close;
```

```
sql.Clear;
```

```
sql.Add('SELECT Verecek.*, Alis.* FROM Verecek, Alis Where  
Verecek.Alis_No=Alis.Alis_No');
```

```
sql.Add(' and Odeme_Tarihi between #'+tar1+'# and #'+tar2+'#');
```

```
open;
```

```
end;
```

```
end;
```

```
form3.QuickRep2.ReportTitle:=tar1+' ve '+tar2+' tarihleri arasindaki alis raporu';
```

```
form3.QuickRep2.Preview;
```

```
end;
```

```
procedure TForm5.FormShow(Sender: TObject);
```

```
begin
```

```
bsskindateedit1.Date:=now-1;
```

```
bsskindateedit2.Date:=now+1;
```

```
end;
```

```
procedure TForm5.bsSkinButton2Click(Sender: TObject);
```

```
begin
```

```
close;
```

```
end;
```

```
end.
```