



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

PHARMACY AUTOMATION

**Graduation Project
COM 400**

**Ahmad Watad
20002139**

Supervisor: Mr. Ümit İlhan

Nicosia - 2005



ACKNOWLEDGEMENT

First of all, I feel proud to pay my special regards to my project adviser “Mr. Ümit İlhan”. He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. He has Devine place in my heart and I am less than the half without his help. I am really thankful to my teacher.

More over I want to pay special regards to my parents who are enduring these all expenses and supporting me in all events. I am nothing without their prayers. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting. They may get peaceful life in Heaven. At the end I am again thankful to those all persons who helped me or even encouraged me to complete my project. My all efforts to complete this project might be fruitful.

To the best of my knowledge, I want to honor those all persons who have supported me or helped me in my project. I also pay my special thanks to my all friends who have helped me in my project and gave me their precious time to complete my project.

ABSTRACT

Nowadays everywhere computer automation programs are important, because every job needs computer automation to perform multiple tasks, which make it easier to the users to react with the jobs even they don't know all the information needed in that job.

On the other hand a good GUI (graphical user interface) must be implemented in such way that it will help the user to use the program and understand the idea, also to make a good connection between the user and the machine.

Visual Basic serves this purpose, it's an object oriented programming language depends on GUI and events.

In all environments like hospital, hotels, airports, pharmacies automation process is very important for arrangements and the speed of work, so I used Visual Basic to create pharmacy automation program for controlling products in any pharmacy.

TABLE OF CONTENTS

ACKNOWLEDGMENT	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	1
CHAPTER ONE: VISUAL BASIC PROGRAMMING	2
1.1 Introduction to Visual Basic	2
1.2 Brief History	4
1.3 The Basics of a Programming Language	4
1.4 Visual Basic is Windows Development Language	4
1.5 Developing an Application in VB	7
1.5.1 Building Applications with Visual Basic	7
1.5.2 Design VB Applications	8
1.5.3 Running Application	8
1.6 New Tools in data access	8
1.6.1 ADO (ActiveX Data Objects)	8
1.6.3 DataGrid Control	9
CHAPTER TWO: MICROSOFT ACCESS	10
2.1 Introduction to Microsoft Access	10
2.2 Data Definition of Access Databases	11
2.3 Defining Relationships and Referential Integrity Constraints	12
2.4 Data Manipulation in Access	12
2.5 Designing of the Databases	14
2.6 Naming Fields	16
2.7 Assigning Field Data Types and Defining Properties	17
2.8 Specifying a Primary Key	18
2.9 Adding Records	18
2.10 Retrieving and Reporting Information	19

CHAPTER THREE: THE TOOLS USED IN PROGRAM	20
3.1 The ADO Data Control	20
3.1.1 Possible Uses	20
3.2 The DataGrid Control	21
3.3 The ProgressBar	23
3.4 Using the ListView Control	23
3.4.1 Possible Uses	24
3.4.2 Set Column Text with the ListSubItems Collection	24
3.4.3 SubItems Depend on ColumnHeaders Presence	24
3.5 The Tag Property (ActiveX Controls)	25
3.6 The Combo Box Control	26
3.6.1 When to Use a Combo Box Instead of a List Box	26
3.6.2 Drop-down Combo Box	26
3.6.3 Accessing List Items with the List Property	26
3.6.4 Determining Position with the ListIndex Property	27
3.7 The Masked Edit	27
3.8 The Timer Control	28
3.8.1 Placing a Timer Control on a Form	29
3.8.2 Initializing a Timer Control	30
3.9 Multiple-Document Interface (MDI) Applications	31
CHAPTAR FOUR: PROGRAM DESIGN PROCESS	32
4.1 Block Diagram of System	32
4.2 Main Menu	33
4.2.1 Main Menu Screen	33
4.3 Sell Menu	34
4.3.1 Sell Menu Screen	35
4.3.2 Sell Flowchart	36
4.4 Customers Menu	37
4.4.1 Customer Menu Screen	38
4.4.2 Customer Flowchart	39
4.5 Products Menu	40
4.5.1 Purchase Menu Screen	41

4.5.2 Products Flowchart	42
4.6 Purchase Menu	43
4.6.1 Purchase Menu Screen	44
4.6.2 Purchase Flowchart	45
CONCLUSION	46
REFERENCES	47
APPENDIX A	48
APPENDIX B	55
APPENDIX C	57

INTRODUCTION

Now a day's, the computer science both hardware and software is being developed over the past years, programming is always providing the scientists by a systematic development, in my project I did construct special program related to Pharmacy Automation, the pharmacy industry not be regarded as standing separate and unrelated to other industries, it is within this framework that the history of pharmacy development should be examined, new concepts in pharmacy design have been developed more recently in an effort to meet the changing preferences and new characteristics.

The pharmacy consist of many departments like, sell, customers, products and purchase, my project program resume that the briefly in a quick time in order to have quick and economic services, on the other hand, the pharmacy development is suitable for researchers and students in computer science, the development of pharmacy automation programs is designed to help compute professionals who want to learn about this exciting field and to serve as a basic reference.

The aim of my project how to create and to develop a project in a scientific method to introduce the gab between scientific theoretical life and work normal life.

In my project, I did construct pharmacy automation program because the availability of information is incrementally important in all over the world, how to make a cays process in order to have a quick research, data process, analysis process.

Finally, full file enclosed full details about the project.

1. VISUAL BASIC PROGRAMMING

1.1 Introduction to Visual Basic

The "Visual" part refers to the method used to create the graphical user interface (GUI). Rather than writing numerous lines of code to describe the appearance and location of interface elements, you simply add rebuilt objects into place on screen. If you've ever used a drawing program such as Paint, you already have most of the skills necessary to create an effective user interface.

The "Basic" part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI. Beginners can create useful applications by learning just a few of the keywords, yet the power of the language allows professionals to accomplish anything that can be accomplished using any other Windows programming language.

Visual Basic is an Object-Oriented Programming (OOP) language and a Rapid Application Development (RAD) environment from Microsoft. Visual Basic provides tools for Internet programming, and helps developers quickly create and deploy enterprise client/server applications, most often to access both local and remote databases.

It's evolved from the earlier DOS version called BASIC (Beginners' All-Purpose Symbolic Instruction Code) in which programming is done in a text-only environment and the program is executed sequentially. BASIC has advanced through many versions since it was first created in 1964 at Dartmouth College. This initial version of BASIC allowed students to write programs to run the Time-Sharing System, one of the first time-share computer systems in the United States.

Visual Basic has diverged from BASIC into an Object-oriented Programming Language, and even further into a visual and action, or events, driven language. It offers a GUI (Graphical User Interface) to allow developers to choose and modify pre-selected sections of code written in BASIC syntax. Utilizing a graphical environment, Visual Basic developers can select and edit program objects independently. Consequently, a fully functional VB Program is made up of many subprograms that can be executed independently or grouped together.

The Visual Basic programming language is not unique to Visual Basic. The Visual Basic system Edition included in Microsoft Excel, Microsoft Access, and many other Windows applications uses the same language. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language. The investment you make in learning Visual Basic will carry over to these other areas.

Whether your goal is to create a small utility for yourself or your work group, a large enterprise-wide system, or even distributed applications spanning the globe via the Internet, Visual Basic has the tools you need.

Data access features allow you to create databases, front-end applications, and scalable server-side components for most popular database formats, including Microsoft SQL Server and other enterprise-level databases.

ActiveX™ technologies allow you to use the functionality provided by other applications, such as Microsoft Word processor, Microsoft Excel spreadsheet, and other Windows applications. You can even automate applications and objects created using the Professional or Enterprise editions of Visual Basic.

Internet capabilities make it easy to provide access to documents and applications across the Internet or intranet from within your application, or to create Internet server applications.

Visual Basic is designed for simple, rapid application development, and can be used to prototype an application that will later be written in a more difficult but efficient language. Other object-oriented programming languages such as C++, Java, and Smalltalk, operate in text-only environments, and do not employ a GUI to build programs.

1.2 Brief History

In 1988, Alan Cooper, the 'father' of Visual Basic, produced a drag-and-drop shell prototype for the BASIC programming language. The shell prototype, named Tripod, included a widget control box and a small language engine. After showing it to Bill Gates, Microsoft negotiated to buy the concept and code-named it Ruby. Microsoft joined Ruby with their current BASIC programming environment, QuickBasic, resulting in the first tool that allowed developers to create Windows applications quickly, easily, and visually (code named Thunder).

In 1991, Microsoft released Visual Basic 1.0. It was the first visual development tool from Microsoft, and was designed to compete with C, C++, Pascal, and any other well-known programming language at the time. However "when it came out, Visual Basic wasn't a success. It wasn't until Microsoft released VB 2.0 in 1993 that people really started to discover the power of the language, and when Microsoft released VB 3.0 it had become the fastest growing programming language on the market.

1.3 The Basics of a Programming Language

Traditional program languages are composed of commands (often called statements), operators, variables and data. Variables represent data and the statements and operators operate on the data to produce the required output.

1.4 Visual Basic is Windows Development Language

The VB is Windows development language, that's why you must be familiar with the Windows environment. Windows involves three key concepts:

1. Window

A window is a simply rectangular region with its own boundaries.

Examples of windows are:

An Explorer window in Windows operating system.

A document window in word processor.

Dialog box that pop up window and reminds you of an appointment.

A command button.

Icons.

Text boxes.

Option boxes.

Menu bars.

The Microsoft Windows Operating system manages all of these many windows by assigning each one a unique id number. The system continually monitors each of these windows for signs of activity or events.

2. Events

An event is an action recognized by a form or control. Events can occur through user action (response) such as a mouse click or a key press using objects of window (through programmatic control), or even as a result of another window's action.

Event-driven applications execute Basic code in response to an event. Each form and control in VB has a predefined set of events. If one of these events occurs and there is a user code in the associated event procedure, VB invokes that code.

For example most objects recognize a Click event. If a user clicks a form (object), code in the form's Click event procedure is executed. If a user clicks a command button, code in the button's click event procedure is executed.

Each time an event occurs, it causes a message to be sent to the O.S. The system processes the message and broadcasts it to the other windows. Each window can take the appropriate action based on its own instructions from dealing with that particular message.

Fortunately, VB insulates you from having to deal with all of the low-level message handling. Many of the messages are handled automatically by VB.

This allows you to quickly create powerful applications without having to deal with unnecessary details.

- **Understanding the Event-Driven Model**

Programs in conventional (traditional or procedural) programming languages run from the top down. For older programming languages, execution starts from the first line and moves with the flow of the program to different parts as needed.

A VB program usually works completely different. The code doesn't follow a predefined path. It executes different code section in response to events.

The core of a VB program is a set of independent pieces of code that are activated by, and so respond to, only the events they have been told to recognize.

The programming code in VB that tells your program how to respond to events (event procedure). An event procedure is a body of code that is only executed in response to an external event.

Your code can also trigger events during execution. It is for this reason that it is important to understand the event-driven model and keep it in mind when designing applications in windows environment.

1.5 Developing an Application in VB

As you develop an application, you work with a project to manage all the different files that make up the application. A project consists of:

One project file that keeps track of all the components (.vbp).

One file for each form (.frm).

One binary data file for each form containing data for properties of controls on the form (.frx). These files are not editable and are automatically generated for any .frm file that contains binary properties, such as Picture or Icon.

Optionally, one file for each class module (.cls).

Optionally, one file for each standard module (.bas).

Optionally, one or more files containing ActiveX controls (.ocx).

Optionally, a single resource files (.res).

The project file is simply a list of all the files and objects associated with the project, as well as information on the environment options you set. This information is updated every time you save the project. All of the files and objects can be shared by other projects as well.

1.5.1 Building Applications with Visual Basic

There are essentially 3 basic phases of building a computer application:

1. The Design phase, which is analogous to an architect designing a building before it is built.
2. The programming phase, where sets of instructions in the form of functions and subroutines are written to carry out the events of the application.
3. The final step, which actually never ends, is the de- bugging phase.

The last two phases are an iterative procedure, where the programmer should be continuously evaluating potential errors that might arise, and writing code to handle obvious errors. All programs have bugs, but good programs have fewer bugs.

1.5.2 Design VB Applications

Here is a summary of the steps you take to design a VB application:

- Customize the windows that the user sees.
- Decide what events the controls on the window should recognize.
- Write the event procedures for those events.

1.5.3 Running Application

Here is what happens when the application is running:

- The application starts and a form is loaded and displayed
- The form (or a control on the form) receives an event. The event might be caused by the user (for example, a keystroke), by the system (for example, a timer event), or indirectly by your code (for example, a Load event when your code loads a form)
- If you have written an event procedure, VB executes the code.
- The application waits for the next event.

1.6 New Tools in data access

1.6.1 ADO (ActiveX Data Objects)

in All Editions A new OLEDB-aware data source control that functions much like the intrinsic Data and Remote Data controls, in that it allows you to create a database application with minimum code.

Visual Database Tools Integration (Query Designer and Database Designer)

Enterprise Edition Visually create and modify database schemas and queries: Create SQL Server and Oracle database tables drag and drop to create views, and automatically change column data types.

Many data access applications created with earlier versions of Visual Basic store and manage data using the Microsoft Jet database engine, the engine used by Microsoft Access. These applications use Microsoft Data Access Objects (DAO) to access and manipulate data.

When you have completed all the files for a project, you can convert the project into an executable file (.exe): From the File menu, choose the Make project.exe command.

Interacting with Data in a Microsoft Jet/Microsoft Access Database

Now we can use Microsoft ActiveX Data Objects (ADO) to easily manipulate data in a variety of database formats, including Microsoft Jet format. We may still be able to use DAO to work with your local Microsoft Jet databases, but for new applications you'll probably want to use ADO and the new data access features of Visual Basic.

1.6.2 DataGrid Control

All Editions An OLEDB-aware version of DBGrid, the control allows you to quickly build an application to view and edit recordsets. It also supports the new ADO Data contro.

2. MICROSOFT ACCESS

2.1 Introduction to Microsoft Access

Access is one of the well-known implementations of the relational data model on the PC platform. It is considered as part of an integrated set of tools for creating and managing databases on the PC Windows platform. The database applications for Access may range from personal applications, such as maintaining an inventory of your personal audio and video collection, to small business applications, such as maintaining business-specific customer information. With compliance to the Microsoft Open Database Connectivity (ODBC) standard and the prevalence of today's client-server architectures, PC relational databases may be used as a front-end to databases stored on non-PC platforms. For example, an end user can specify ad hoc queries graphically in Access over an Oracle database stored on a UNIX server.

Access provides a database engine and a graphical user interface (GUI) for data definition and manipulation, with the power of SQL. It also provides a programming language called Access Basic. Users can quickly develop forms and reports for input/output operations against the database through the use of Wizards, which are interactive programs that guide the user through a series of questions in a dialog mode. The definition of the forms and reports is interactively accomplished when the user designs the layout and links the different fields on the form or report to items in the database. Access 97 (the latest release of Access at the time of this writing) also provides the database developer with hyperlinks as a native data type, extending the functionality of the database with the ability to share information on the Internet.

Access is an RDBMS that has several components. One component is the underlying database engine, called the Microsoft Jet engine which is responsible for managing the data. Another component is the user interface, which calls the engine to provide data services, such as storage and retrieval of data. The engine stores all the application data (tables, indexes, forms, reports, macros, and modules) in a single Microsoft database file (.mdb file). The engine also provides advanced capabilities, such as heterogeneous data

access through ODBC, data validation, concurrency control using locks, and query optimization.

Access works like a complete application development environment, with the internal engine serving to provide the user with RDBMS capabilities. The Access user interface provides Wizards and Builders to aid the user in designing a database application. Builders are interactive programs that help the user build syntactically correct expressions. The programming model used by Access is event-driven. The user builds a sequence of simple operations, called macros, to be performed in response to actions that occur during the use of the database application. While some applications can be written in their entirety using macros, others may require the extended capabilities of Access Basic, the programming language provided by Access.

There are different ways in which an application with multiple components that includes Access can be integrated. A component (in Microsoft terminology) is an application or development tool that makes its objects available to other applications. Using automation in Visual Basic, it is possible to work with objects from other components to construct a seamless integrated application. Using the Object Linking and Embedding (OLE) technology, a user can include documents created in another component on a report or form within Access. Automation and OLE are distinct technologies, which are a part of the Component Object Model (COM), a standard proposed by Microsoft.

2.2 Data Definition of Access Databases

Although Access provides a programmatic approach to data definition through Access SQL, its dialect of SQL, the Access GUI provides a graphical approach to defining tables and relationships among them. A table can be created directly in a design view or it can be created interactively under the guidance of a table wizard. Table definition contains not only the structure of the table but also the formatting of the field layout and masks for field inputs, validation rules, captions, default values, indexing, and so on. The data types for fields include text, number, date/time, currency, Yes/no (boolean), hyperlink, and AutoNumber, which automatically generates sequential

numbers for new records. Access also provides the capability to import data from external tables and to link to external tables.

Field Properties window for displaying the properties of the Fields. The format property provides for a default display format. The input mask provides automatic formatting characters for display during data input in order to validate the input data. For example, the input mask for SSN displays the hyphen positions and indicates that the other characters are digits. The caption property specifies the name to be used on forms and reports for this field. A blank caption specifies the default, which is the field name itself. A default value can be specified if appropriate for a particular field. Field validation includes the specification of validation rules and validation text—the latter displayed when a validation rule is violated. Other field properties include specifying whether the field is required—that is, NULL is not allowed—and whether textual fields allow zero length strings. Another field property includes the index specification, which allows for three possibilities: (1) no index, (2) an index with duplicates, or (3) an index without duplicates. In the case of primary key, the field is indexed with no duplicates allowed.

In addition to the Field Properties window, Access also provides a Table Properties window. This is used to specify table validation rules, which are integrity constraints across multiple columns of a table or across tables.

2.3 Defining Relationships and Referential Integrity Constraints

Access allows interactive definition of relationships between tables—which can specify referential integrity constraints—via the Relationships window. To define a relationship, the user first adds the two tables involved to the window display and then selects the primary key of one table and drags it to where it appears as a foreign key in the other table. This action pops up another window that prompts the user for further information regarding the establishment of the relationship, the user checks the "Enforce Referential Integrity" box if Access is to automatically enforce the referential integrity specified by the relationship. The user may also specify the automatic cascading of

updates to related fields and deletions of related records by selecting the appropriate boxes. The "Relationship Type" is automatically determined by Access based on the definition of the related fields. If only one of the related fields is a primary key or has a unique index, then Access creates a one-to-many relationship, indicating that an instance (value) of the primary key can appear many times as an instance of the foreign key in the related table. If both fields are either keys or have unique indexes, then Access creates a one-to-one relationship.

Although specifying a relationship is the mechanism used to specify referential integrity between tables, the user need not choose the option to enforce referential integrity because relationships are also used to specify implicit join conditions for queries. For example, if no relationship is pre-specified during the graphical design of a query, then a default join of the related fields is performed if related tables are selected for that query, regardless of whether referential integrity is enforced or not. Access chooses an inner join as the default join type but the user may choose a right or left outer join by clicking on the "Join Type" box and selecting the appropriate join type.

2.4 Data Manipulation in Access

The data manipulation operations of the relational model are categorized into retrieval queries and updates (insert, delete, and modify operations). Access provides for query definition either graphically through a QBE interface or programmatically through Access SQL. The user has the ability to design a graphical query and then switch to the SQL view to examine the SQL query generated by Access. Access provides for update operations through forms that are built by the application programmer, by direct manipulation of the table data in Datasheet view, or through the Access Basic programming language.

Retrieval operations are easily specified graphically in the Access QBE interface. In QBE and SQL. To establish a join that had not been prespecified the user selects the join attribute from one table and drags it over to the join attribute in the other table. To include an attribute in the query, the user drags it from the top window to the bottom

window. For attributes to be displayed in the query result, the user checks the "Show" box. To specify a selection condition on an attribute, the user can type an expression directly in the "Criteria" grid or use the aid of an Expression Builder. To see the equivalent query in Access SQL, the user switches from the QBE Design View to the SQL View.

Update operations on the database are typically guided by the use of forms that incorporate the business rules of the application. There is also a Datasheet view of a table that the sophisticated end user can use to insert, delete, or modify data directly by choosing "open table" from a database window. These updates are subject to the constraints specified through the data definition process, including data types, input masks, field and table validation rules, and relationships.

2.5 Designing of the Databases

A database is only useful if it is designed to meet the specific needs of its users. Good database design requires careful planning to determine the fields, tables, and relationships needed to satisfy the data input and output requirements. The following guidelines help to insure that the database will be able to produce the needed results:

- Identify all of the fields needed to produce the required information. Consider the type of information to be stored in the database and the type of reports that must be generated from the data. Plan fields that will produce this information.
- Group related fields into tables. Look for logical grouping of field information. For example, all information pertaining to students might be placed in one table. All information pertaining to counselors might be placed in a second table.
- Determine each table's primary key field. Look for a field that uniquely identifies each record. Such fields include social security numbers, identification codes, part numbers, or

product serial numbers. It might be necessary to assign a unique number to each record or to allow Access to assign one automatically.

- Include a common field in related tables. The common field is used to connect one table logically with another table. For example, each student record might include a counselor code that matches the counselor code listed for each counselor in the counselor table.
- Avoid redundancy. Data redundancy occurs when the data is stored in more than one place in the database. With the exception of the common field(s) to connect tables, redundancy wastes storage space and can increase the likelihood that data will be entered inconsistently.
- Determine the properties of each field. Field properties include field name, field type, maximum number of characters, field description, and validity

Microsoft Access database mainly consists of: Database File, Table, Record, Field, Field value, Data-type. Here is the Hierarchy that Microsoft Access uses in breaking down a database

Database File: This is your main file that encompasses the entire database and that is saved to your hard-drive or floppy disk, it's A collection of related tables. (Access is a relational database).

Table: A table is a collection of data about a specific topic (A collection of records.). There can be multiple tables in a database.

Field: Fields are the different categories within a Table. Tables usually contain multiple fields, it's a single characteristic or attribute of a person, place, object, event or idea (a column).

Field Value: The specific value, or content, of a field.

Primary Key - A field, or a collection of fields, whose values uniquely identify each record (unique identifier).

Common Field: A field that appears in both tables. The common field is used to connect tables.

Record: A set of field values that describe a person, place, object, event, or idea (a row).

Datatypes: Datatypes are the properties of each field. A field only has 1 datatype.

2.6 Naming Fields

Each field on a database must have a name (this is also true for anything in the computer). The name held by a field allows you, the database developer, and the operating system, to refer to that particular field.

It is best to choose a field name that describes the purpose of the field so that it is easy to remember. In addition, the following rules apply to naming fields:

- Must not exceed 255 characters. You should limit the name of a variable to 30 characters
- A name can contain letters, numbers, spaces, and special characters except for a period, exclamation mark, accent mark, and square brackets
- A name cannot begin with a space, Must begin with a letter (a-z or A-Z)
- A name must be unique within a table, but it can be used again in another table.

Experienced users of databases capitalize the first letter of each word in a field name, avoid using long field names, use standard abbreviations, and avoid using spaces in field names.

2.7 Assigning Field Data Types and Defining Properties

After specifying a name of the field, you can decide what type of data can be entered into that field. The data type determines the field values that can be entered in the field. Access provides the following data types:

Text: Allows field values containing letters, digits, spaces and special characters. Field size: 0 - 255 characters.

Memo: Allows field values containing letters, digits, spaces and special characters that make up long comments. Field size: 1-64,000 characters.

Number: Allows positive and negative numbers as field values. Field size: 1-15 digits.

Date/Time: Allows field values containing dates and times to December 31, 9999. Field size: 8 bytes.

Currency: Allows field values similar to number data type using the currency format. Field size: 15 digits on the left side of decimal and 4 digits on the right side.

AutoNumber: Integers controlled by Access. Access automatically inserts a value field and numbers records as they are entered. Field size: 9 digits.

Yes/No: Limits values to yes and no, on and off, or true and false. Field size: 1 character.

Hyperlink: Consists of a hyperlink address. Field size: 1 gigabyte maximum.

Lookup Wizard: Creates a field that lets you look up a field value in another table or in a predefined list of values. Field size: Same as the primary key field used to perform the lookup.

Each data type allows for a set of properties that help to insure that the data is entered accurately. Such properties include making fields required, selecting default values, entering captions, and specifying data validation rules and text.

2.8 Specifying a Primary Key

A primary key uniquely identifies each record in the table. Access does not allow for duplicate values in the primary key field. Once a primary key field is selected, every record must have a value in the primary key field. Access stores the records on the disk in the order they are entered but displays them in order by the field values of the primary key. In addition, Access responds faster to requests for specific records based on the primary key.

2.9 Adding Records

Records are added to tables by using the table datasheet or by creating a form. A table datasheet provides a simple way to add records. A table datasheet displays records in rows and columns. Each row is a separate record in the table, and each field is a separate column. When a table contains many fields, it is useful to create a form to maintain the records. While forms can be customized, Access provides a wizard that automatically creates a form for data entry.

2.10 Retrieving and Reporting Information

The process of retrieving information from a database is known as querying. Access provides powerful query capabilities that allow the user to display selected fields and records from a table, sort records, perform calculations, find and display information from two or more tables, and generate professionally designed reports.

3. THE TOOLS USED IN PROGRAM

3.1 The ADO Data Control

The ADO Data control uses Microsoft ActiveX Data Objects (ADO) to quickly create connections between data-bound controls and data providers. Data-bound controls are any controls that feature a DataSource property. Data providers can be any source written to the OLE DB specification. You can also easily create your own data provider using Visual Basic's class module.

Although you can use the ActiveX Data Objects directly in your applications, the ADO Data control has the advantage of being a graphic control (with Back and Forward buttons) and an easy-to-use interface that allows you to create database applications with a minimum of code.

Several of the controls found in Visual Basic's Toolbox can be data-bound, including the CheckBox, ComboBox, Image, Label, ListBox, PictureBox, and TextBox controls. Additionally, Visual Basic includes several data-bound ActiveX controls such as the DataGrid, DataCombo, Chart, and DataList controls. You can also create your own data-bound ActiveX controls, or purchase controls from other vendors.

Previous versions of Visual Basic featured the intrinsic Data control and the Remote Data control (RDC) for data access. Both controls are still included with Visual Basic for backward compatibility. However, because of the flexibility of ADO, it's recommended that new database applications be created using the ADO Data Control.

3.1.1 Possible Uses

- Connect to a local or remote database.

- Open a specified database table or define a set of records based on a Structured Query Language (SQL) query or stored procedure or view of the tables in that database.
- Pass data field values to data-bound controls, where you can display or change the values.
- Add new records or update a database based on any changes you make to data displayed in the bound controls.

To create a client, or front-end database application, add the ADO Data control to your forms just as you would any other Visual Basic control. You can have as many ADO Data controls on your form as you need. Be aware, however, that the control is a comparatively "expensive" method of creating connections, using at least two connections for the first control, and one more for each subsequent control.

3.2 The DataGrid Control

Displays and enables data manipulation of a series of rows and columns representing records and fields from a Recordset object.

The data-aware DataGrid control appears similar to the Grid control; however, you can set the DataGrid control's DataSource property to a Data control so that the control is automatically filled and its column headers set automatically from a Data control's Recordset object. The DataGrid control is really a fixed collection of columns, each with an indeterminate number of rows.

Each cell of a DataGrid control can hold text values, but not linked or embedded objects. You can specify the current cell in code, or the user can change it at run time using the mouse or the arrow keys. Cells can be edited interactively, by typing into the cell, or programmatically. Cells can be selected individually or by row.

If a cell's text is too long to be displayed in the cell, the text wraps to the next line within the same cell. To display the wrapped text, you must increase the cell's Column object's

Width property and/or the DataGrid control's RowHeight property. At design time, you can change the column width interactively by resizing the column or by changing the column's width in the Column object's property page.

Use the DataGrid control's Columns collection's Count Property and the Recordset object's RecordCount property to determine the number of columns and rows in the control. A DataGrid control can have as many rows as the system resources can support and up to 32767 columns.

When you select a cell, the ColIndex property is set, thus selecting one of the Column objects in the DataGrid object's Columns collection. The Text and Value properties of the Column object reference the contents of the current cell. The data in the current row can be accessed using the Bookmark property, which provides access to the underlying Recordset object's record. Each column of the DataGrid control has its own font, border, word wrap, and other attributes that can be set without regard to other columns. At design time, you can set the column width and row height and establish columns that are not visible to the user. You can also prevent users from changing the formatting at run time.

Note If you set any of the DataGrid column properties at design time, you will need to set all of them in order to maintain the current settings.

Note If you use the Move method to position the DataGrid control, you may need to use the Refresh method to force it to repaint.

The DataGrid control functions similarly to the DBGrid control except that it doesn't support an unbound mode.

3.3 The ProgressBar

The ProgressBar control shows the progress of a lengthy operation by filling a rectangle with chunks from left to right; it monitors an operation's progress toward completion.

It has a range and a current position. The range represents the entire duration of the operation. The current position represents the progress the application has made toward completing the operation. The Max and Min properties set the limits of the range. The Value property specifies the current position within that range. Because chunks are used to fill in the control, the amount filled in only approximates the Value property's current setting. Based on the control's size, the Value property determines when to display the next chunk.

The ProgressBar control's Height and Width properties determine the number and size of the chunks that fill the control. The more chunks, the more accurately the control portrays an operation's progress. To increase the number of chunks displayed, decrease the control's Height or increase its Width. The BorderStyle property setting also affects the number and size of the chunks. To accommodate a border, the chunk size becomes smaller.

In my project the progressBar is used to make sure that the user enters his password within an interval of time, so if the progressBas is full the program will be terminated even the user enters the correct password.

3.4 The Using the ListView Control

The ListView control displays data as ListItem objects. Each ListItem object can have an optional icon associated with the label of the object. The control excels at representing subsets of data (such as members of a database) or discrete objects (such as document templates).

3.4.1 Possible Uses

To display the results of a query on a database.

To display all the records in a database table.

In tandem with a Treeview control, to give users an expanded view of a TreeView control node.

3.4.2 Set Column Text with the ListSubItems Collection

Notice that in any of the views except Report view, the ListItem object displays only one label the Text property. But in Report view, every ListItem object can have several other text items. For example, the "Hitchhiker's Guide to Visual Basic..." also has an author ("Vaughn, William R."), year (1996), and ISBN number associated with it. Each of these text items are members of the ListSubItems collection. To create a ListSubItem object, use the Add method for the ListSubItems collection. Thus, to set the author, year and ISBN number of a ListItem object.

3.4.3 SubItems Depend on ColumnHeaders Presence

Both the presence and number of ListSubItem objects depends on the presence and number of ColumnHeader objects. That is, you cannot create any ListSubItem objects if there are no ColumnHeader objects present. Further, the number of ColumnHeader objects determines the number of ListSubItem objects you can set for the ListItem object. And the number of ListSubItems is always one less than the number of ColumnHeader objects. This is because the first ColumnHeader object is always associated with the Text property of the ListItem object

3.5 The Tag Property (ActiveX Controls)

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the Tag property isn't used by Visual Basic; you can use this property to identify objects.

Syntax

`object.Tag [= expression]`

The Tag property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
expression	A string expression identifying the object. The default is a zero-length string ("").

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The Tag property is useful when you need to check the identity of a control or MDIForm object that is passed as a variable to a procedure.

Tip When you create a new instance of a form, assign a unique value to the Tag property.

Note The Tag property is of type Variant for ActiveX control collections such as Toolbar Button objects, TreeView Node objects, ListView ListItem and ColumnHeader objects, ImageList ListImage objects, TabStrip Tab objects, and StatusBar Panel objects. You can use the Tag property to pass values, but it does not allow you to pass objects

3.6 The Combo Box Control

A combo box control combines the features of a text box and a list box. This control allows the user to select an item either by typing text into the combo box, or by selecting it from the list.

3.6.1 When to Use a Combo Box Instead of a List Box

Generally, a combo box is appropriate when there is a list of suggested choices, and a list box is appropriate when you want to limit input to what is on the list. A combo box contains an edit field, so choices not on the list can be typed in this field.

In addition, combo boxes save space on a form. Because the full list is not displayed until the user clicks the down arrow (except for Style 1, which is always dropped down), a combo box can easily fit in a small space where a list box would not fit.

3.6.2 Drop-down Combo Box

The user can either enter text directly (as in a text box) or click the detached arrow at the right of the combo box to open a list of choices. Selecting one of the choices inserts it into the text portion at the top of the combo box. The user also can open the list by pressing ALT+ DOWN ARROW when the control has the focus.

3.6.3 Accessing List Items with the List Property

The List property provides access to all items in the list. This property contains an array in which each item in the list is an element of the array. Each item is represented in string form. To refer to an item in the list, use this syntax:

box.List(index)

The box argument is a reference to a combo box, and index is the position of the item.

3.6.4 Determining Position with the ListIndex Property

If you want to know the position of the selected item in a list in a combo box, use the ListIndex property. This property sets or returns the index of the currently selected item in the control and is available only at run time. Setting the ListIndex property for a combo box also generates a Click event for the control.

The value of this property is 0 if the first (top) item is selected, 1 if the next item down is selected, and so on. ListIndex is - 1 if no item is selected or if a user enters a choice in a combo box (Style 0 or 1) instead of selecting an existing item in the list.

Note The NewIndex property allows you to keep track of the index of the last item added to the list. This can be useful when inserting an item into a sorted list.

In the sell process I used the combobox to dropdown the names of customers included

In my database so I can choose the customer that I want to sell the product to.

3.7 The Masked Edit

The Masked Edit control provides restricted data input as well as formatted data output. This control supplies visual cues about the type of data being entered or displayed. This is what the control looks like as an icon in the Toolbox:

The Masked Edit control generally behaves as a standard text box control with enhancements for optional masked input and formatted output. If you don't use an input mask, the Masked Edit control behaves much like a standard text box, except for its dynamic data exchange (DDE) capability.

If you define an input mask using the Mask property, each character position in the Masked Edit control maps to either a placeholder of a specified type or a literal character. Literal characters, or literals, can give visual cues about the type of data being used. For example, the parentheses surrounding the area code of a telephone number are literals: (206).

If you attempt to enter a character that conflicts with the input mask, the control generates a ValidationError event. The input mask prevents you from entering invalid characters into the control.

The Masked Edit control has three bound properties: DataChanged, DataField, and DataSource. This means that it can be linked to a data control and display field values for the current record in the recordset. The Masked Edit control can also write out values to the recordset.

When the value of the field referenced by the DataField property is read, it is converted to a Text property string, if possible. If the recordset is updatable, the string is converted to the data type of the field.

In my project I used the maskedit as a user control with a label. Its behaves as a simple Textbox I used it just for simplicity.

3.8 The Timer Control

Timer controls respond to the passage of time. They are independent of the user, and you can program them to take actions at regular intervals. A typical response is checking the system clock to see if it is time to perform some task. Timers also are useful for other kinds of background processing.

Each timer control has an Interval property that specifies the number of milliseconds that pass between one timer events to the next. Unless it is disabled, a timer continues to receive an event (appropriately named the Timer event) at roughly equal intervals of time.

The Interval property has a few limitations to consider when you're programming a timer control:

If your application or another application is making heavy demands on the system — such as long loops, intensive calculations, or drive, network, or port access your application may not get timer events as often as the Interval property specifies.

The interval can be between 0 and 64,767, inclusive, which means that even the longest interval can't be much longer than one minute (about 64.8 seconds).

The interval is not guaranteed to elapse exactly on time. To ensure accuracy, the timer should check the system clock when it needs to, rather than try to keep track of accumulated time internally.

The system generates 18 clock ticks per second so even though the Interval property is measured in milliseconds, the true precision of an interval is no more than one-eighteenth of a second.

Every timer control must be associated with a form. Therefore, to create a timer application, you must create at least one form (though you don't have to make the form visible if you don't need it for any other purpose).

Note The word "timer" is used in several ways in Visual Basic, each closely related to the workings of the timer control. In addition to the control name and control type, "timer" is used in the Timer event and the Timer function.

3.8.1 Placing a Timer Control on a Form

Placing a timer control on a form is like drawing any other control: Click the timer button in the toolbox and drag it onto a form.

The timer appears on the form at design time only so you can select it, view its properties, and write an event procedure for it. At run time, a timer is invisible and its position and size are irrelevant.

3.8.2 Initializing a Timer Control

A timer control has two key properties.

Property	Setting
Enabled	If you want the timer to start working as soon as the form loads, set it to True. Otherwise, leave this property set to False. You might choose to have an outside event (such as a click of a command button) start operation of the timer.
Interval	Number of milliseconds between timer events.

Note that the Enabled property for the timer is different from the Enabled property for other objects. With most objects, the Enabled property determines whether the object can respond to an event caused by the user. With the Timer control, setting Enabled to False suspends timer operation.

Remember that the Timer event is periodic. The Interval property doesn't determine "how long" as much as it determines "how often." The length of the interval should depend on how much precision you want. Because there is some built-in potential for error, make the interval one-half the desired amount of precision.

Note The more often a timer event is generated, the more processor time is used in responding to the event. This can slow down overall performance. Don't set a particularly small interval unless you need it.

3.9 Multiple-Document Interface (MDI) Applications

The multiple-document interface (MDI) allows you to create an application that maintains multiple forms within a single container form. Applications such as Microsoft Excel and Microsoft Word for Windows have multiple-document interfaces.

An MDI application allows the user to display multiple documents at the same time, with each document displayed in its own window. Documents or child windows are contained in a parent window, which provides a workspace for all the child windows in the application. For example, Microsoft Excel allows you to create and display multiple-document windows of different types. Each individual window is confined to the area of the Excel parent window. When you minimize Excel, all of the document windows are minimized as well; only the parent window's icon appears in the task bar.

A child form is an ordinary form that has its MDIChild property set to true. Your application can include many MDI child forms of similar or different types.

At run time, child forms are displayed within the workspace of the MDI parent form (the area inside the form's borders and below the title and menu bars). When a child form is minimized, its icon appears within the workspace of the MDI form instead of on the taskbar.

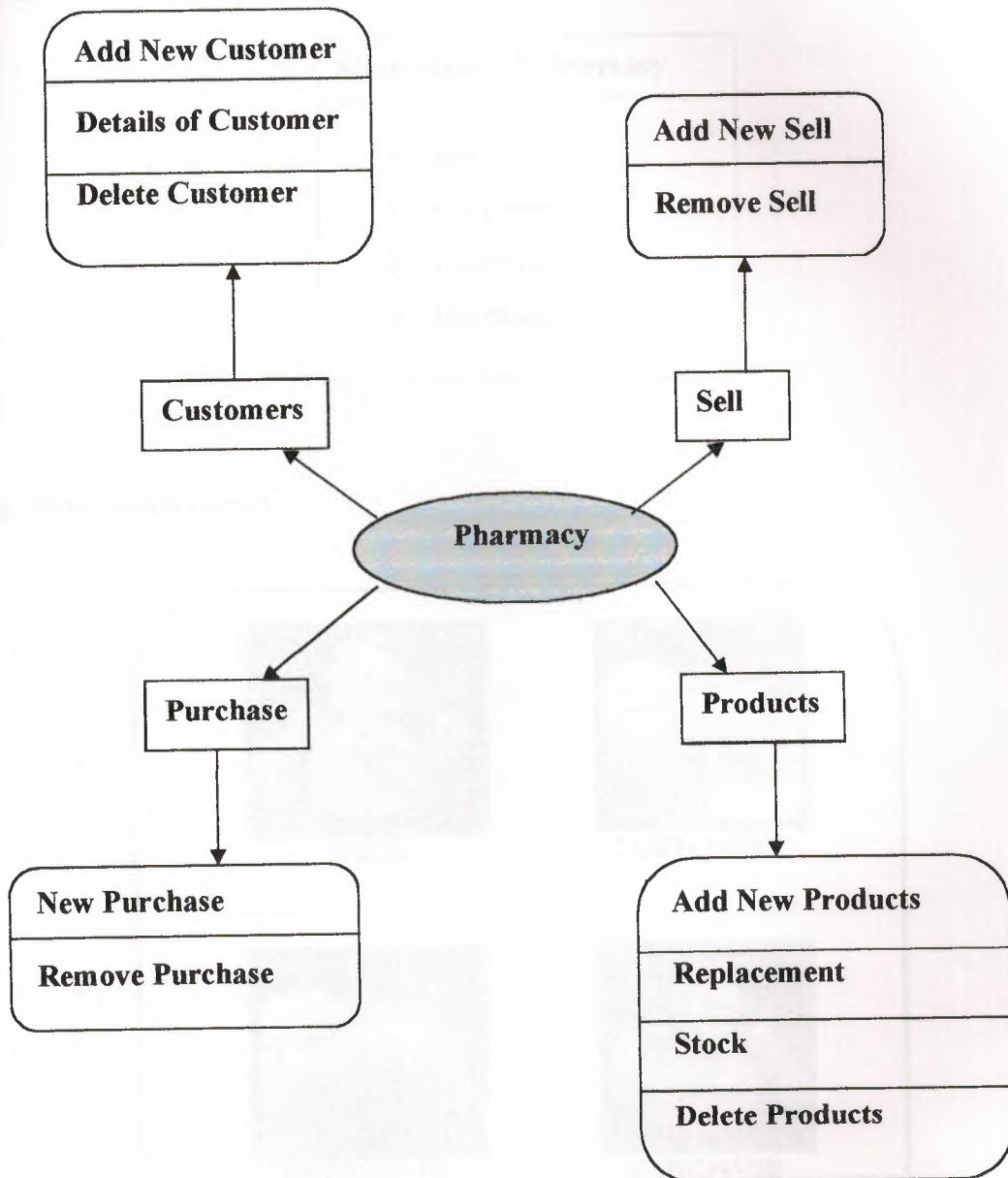
The application can also include standard, non-MDI forms that are not contained in the MDI form. A typical use of a standard form in an MDI application is to display a modal dialog box.

An MDI form is similar to an ordinary form with one restriction. Its not allowed to place a control directly on a MDI form unless that control has an Align property (such as a picture box control) or has no visible interface (such as a timer control).

In my program the customers and products forms are MDI Childs for the Main form.

4. PROGRAM DESIGN PROCESS

4.1 Block Diagram of System

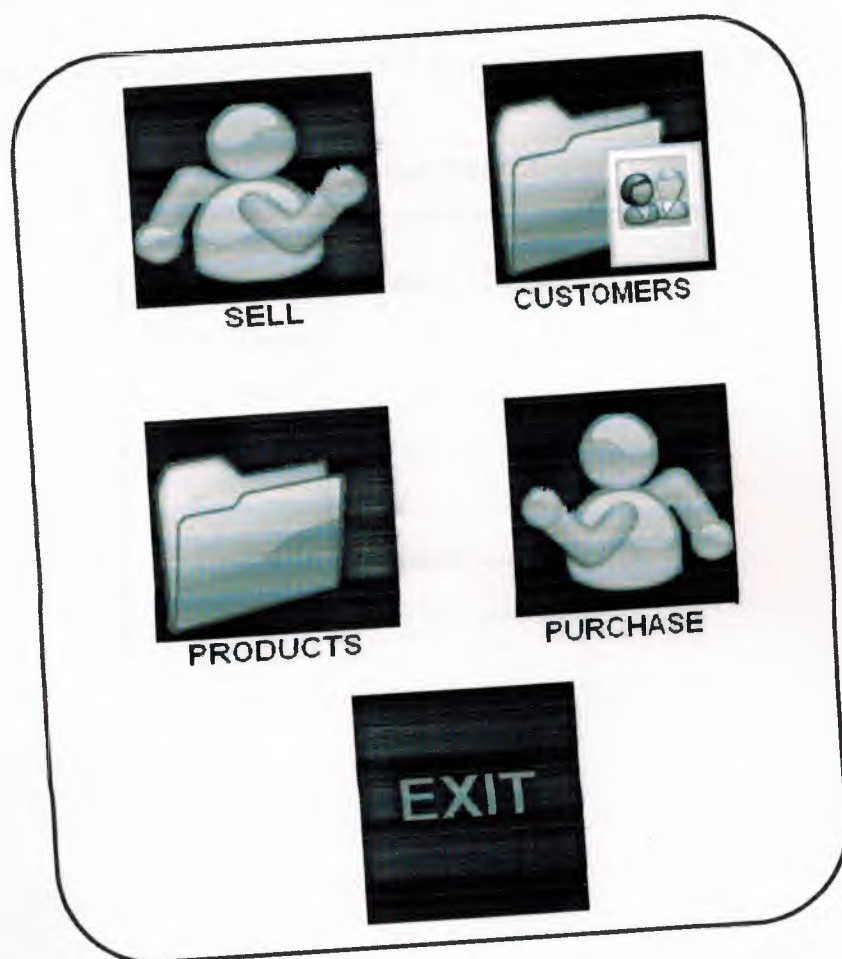


4.2 Main Menu

The aim of the main menu is to use the program **easily**, faster and use all the process screens or necessary program at the same time. In the main menu consists of four departments, under each department there is some information about it, as the following:

Main Menu of Pharmacy
1. Sell
2. Customers
3. Products
4. Purchase

4.2.1 Main Menu Screen



4.3 Sell Menu

In sell process the program allow the user to sell a product from the products In the database ,so the user is going to select this product and add it to the List view (which is described in the tools),the list view will show the quantity That will be sold and the price for each piece and the total price for all pieces and we can select a current customer to sell this product to him so if the customer Didn't pay all the price his dept will be increased and the screen and the flowchart Shows the whole process :

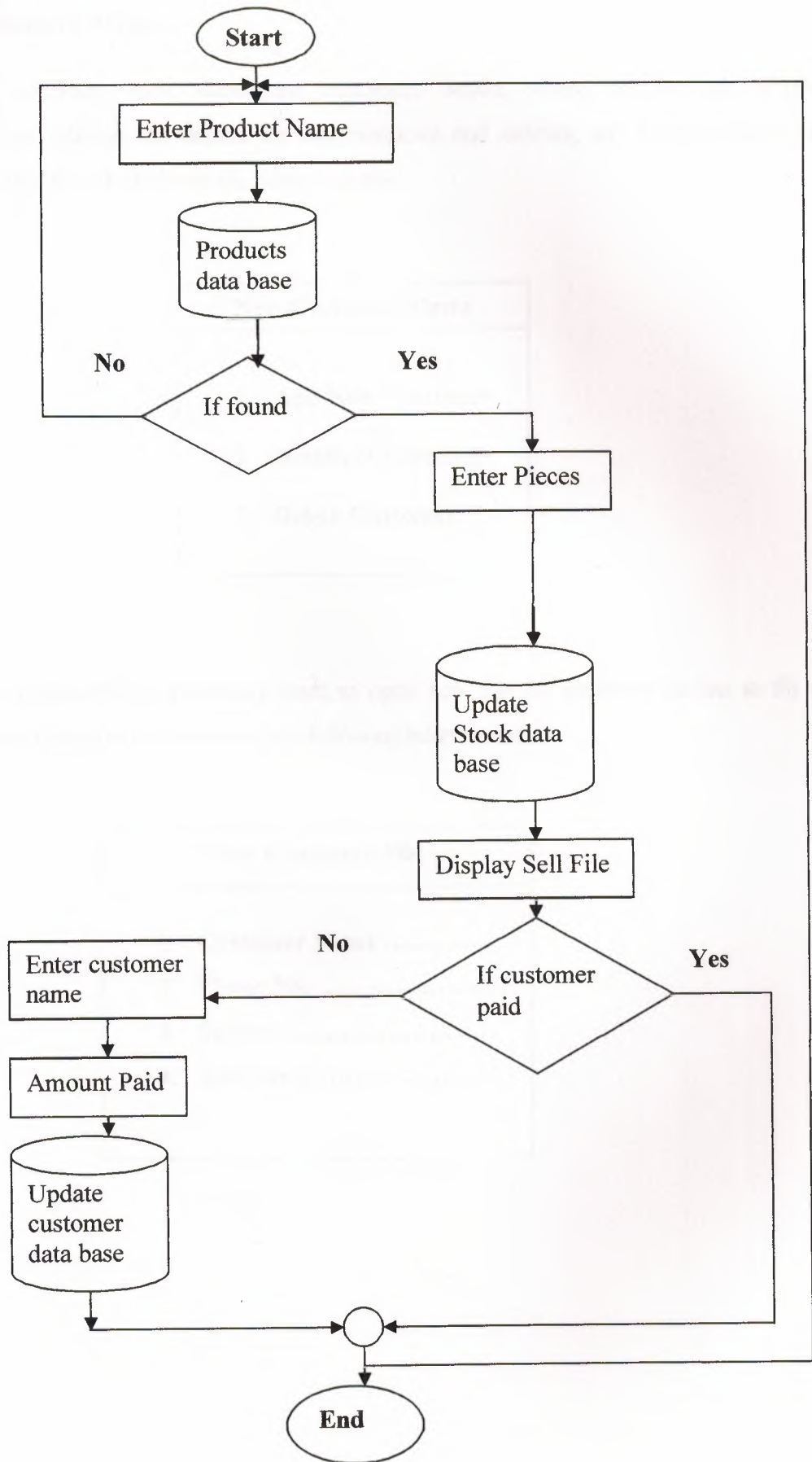
Sell Menu
<ol style="list-style-type: none">1. Add New Sell2. Remove Sell

When the add new sell option is selected then the following file has to fill.

New Sell File
<ol style="list-style-type: none">1. Product Name2. Quantity3. Price4. Total5. Grand Total6. Customer Name7. Paid

[illegible]

4.3.2 Sell Flowchart



4.4 Customers Menu

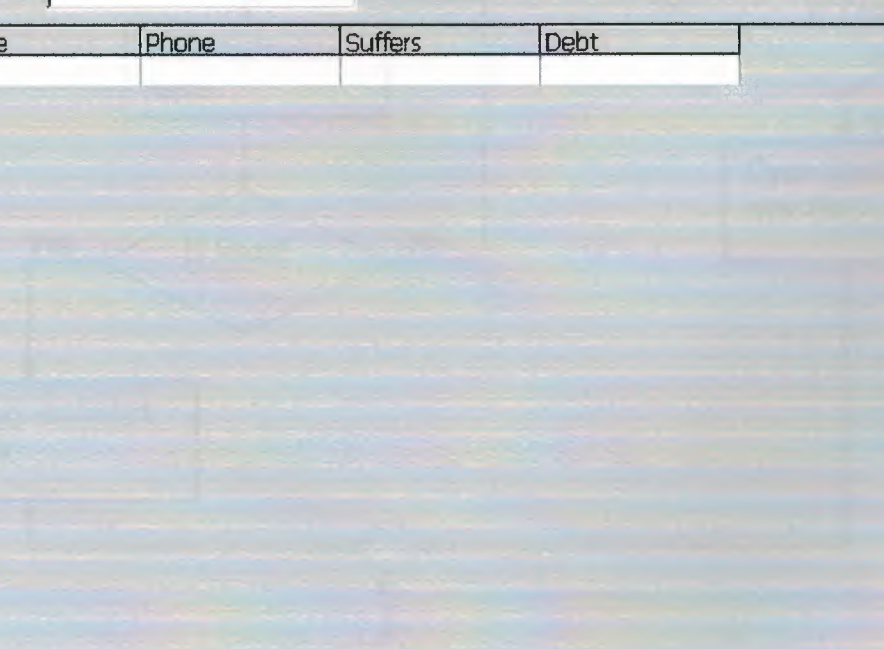
The customers form shows the customers details, where we can add a new customer and change the details for any customer and deleting any customer also the screen and the flowchart shows the whole process:

New Customer Menu
<ol style="list-style-type: none">1. Add New Customer2. Details of Customer3. Delete Customer

When the responsible of pharmacy want to open new file for customer he has to fill in the database related to that customer the following information:

New Customer File
<ol style="list-style-type: none">1. Customer Name2. Phone No.3. Suffers4. Address

Details of Customer	
1. Customer Name	
2. Phone No.	
3. Suffers	
4. Address	
5. Dept	



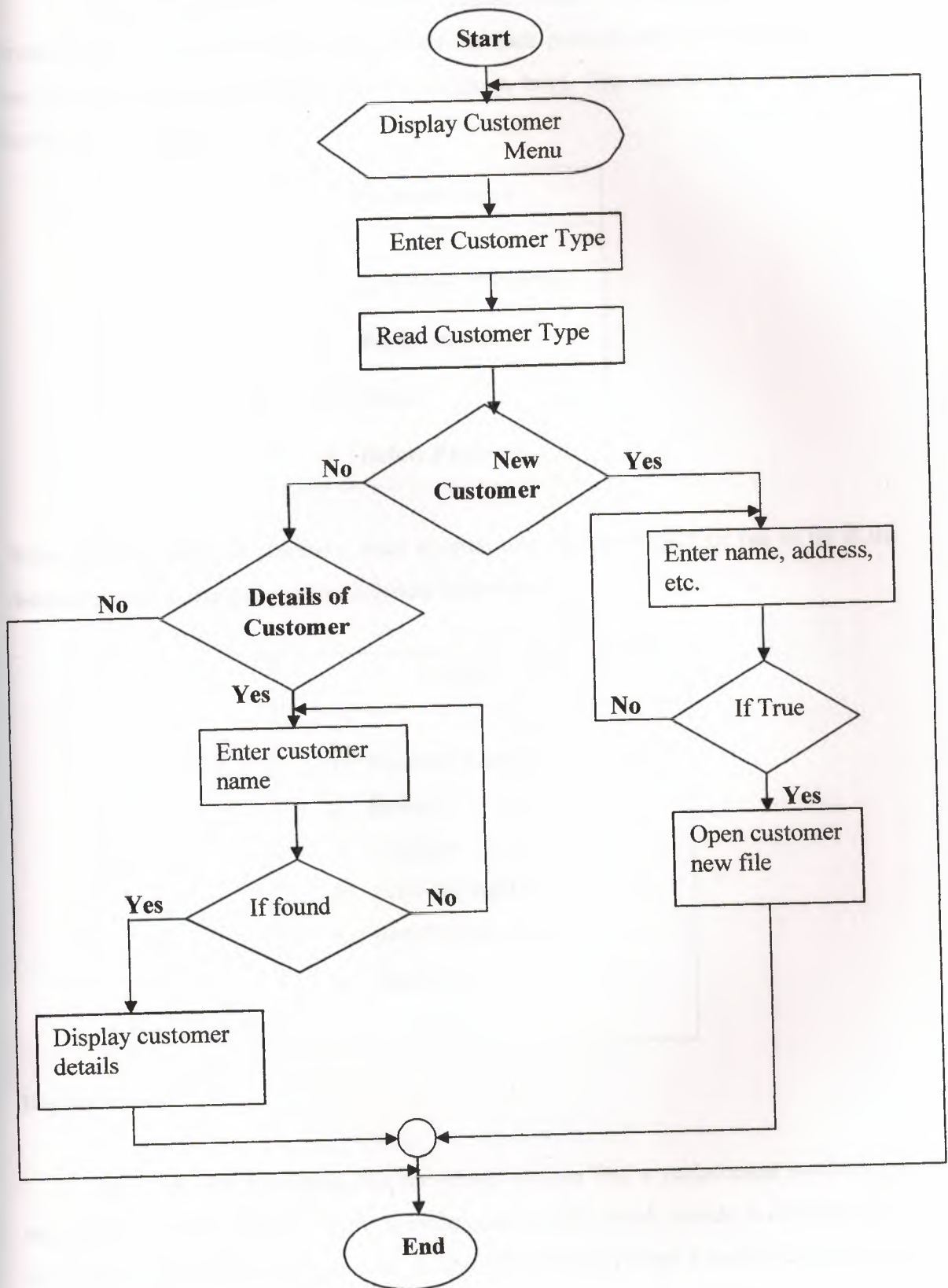
Customers

Search:

Name	Phone	Suffers	Debt	
------	-------	---------	------	--

Add [F1] Details [F2] Delete [F3] Close

4.4.2 Customer Flowchart



4.5 Products Menu

The products form shows products details, where we can add, delete a new product and it's also shows the replacements for each product (the product that the user can use instead of the medicine that he wants to buy). The screen and the flowchart Shows the whole process:

Products Menu
<ol style="list-style-type: none">1. Add New Products2. Replacement3. Stock4. Delete Products

When the responsible of pharmacy want to open new file for product he has to fill in the database related to that product the following information:

Product File
<ol style="list-style-type: none">1. Product Name2. Brand3. Category4. Price (bought)5. Retail Price6. Barcode

Replacements:

this is a very interesting process where we can find a replacement medicine for each product .in the database there is replacements table which include a common id for each product and the replacement for it, the SQL statement that I used makes an inner join between the table of replacements and the product, produt_1 tables to select the replacement for the product that we choose from the products form.

Selecting the Stocks:

In this process we select stocks from the stock table after making inner join between the products and stocks tables, in here we can search for the stocks using product name or product barcode or product category.

Stock File
1. Product Name
2. Amount

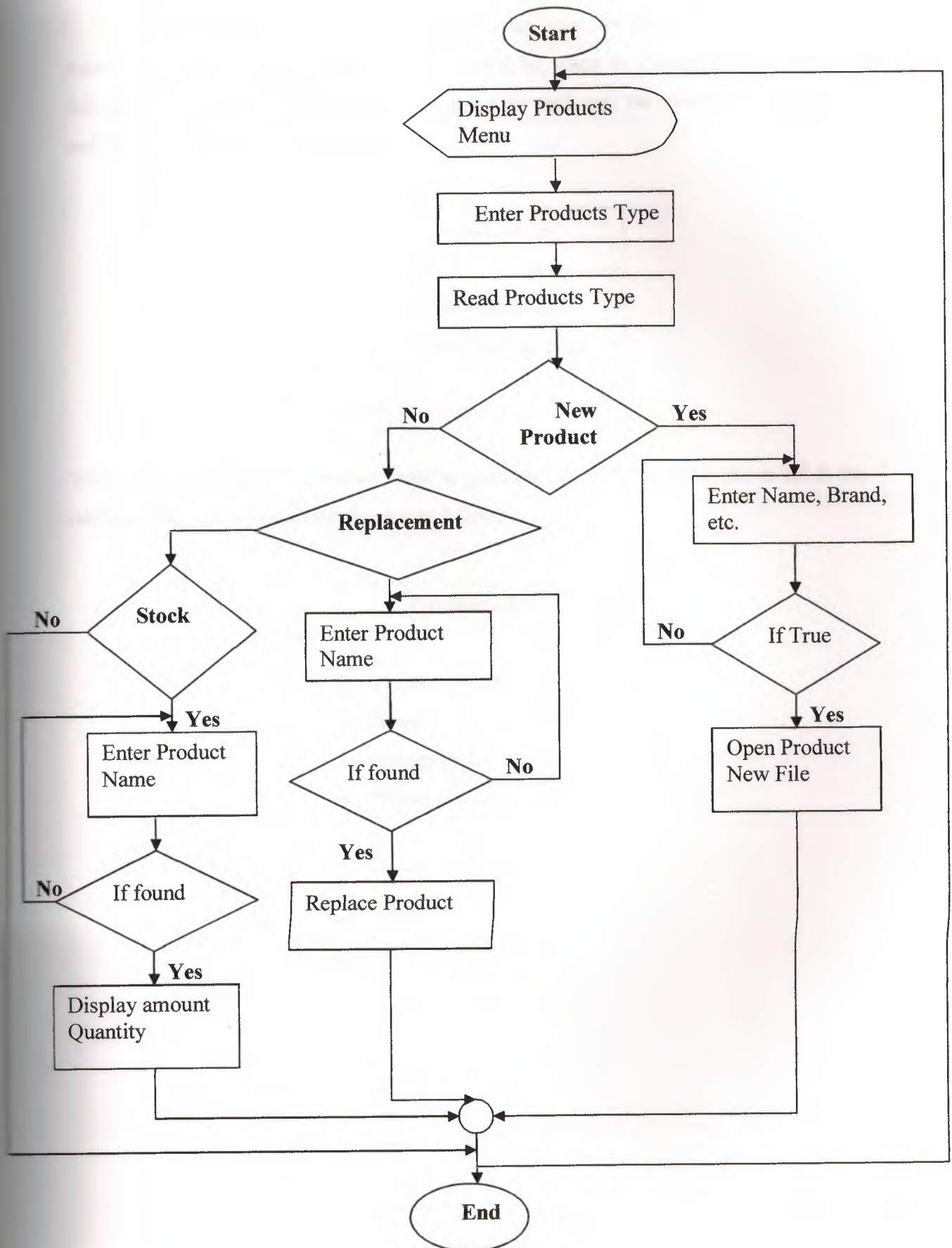
4.5.1 products Menu Screen

The screenshot shows a software window titled 'Products'. At the top, there is a search bar labeled 'Search:' and a button labeled 'Stock [F4]'. Below this is a table with the following data:

	Name	Brand	Category	Price	Barcode
▶	profeen	jap.ITD	mosaken alam	\$ 17.50	111
	aspreen	popopdd	headick	\$ 16.50	3655555
	Allerfin	sult-jordan	antiheamine	\$ 25.00	113
	panadol	momowawa	headeack	\$ 7.00	114
	morofeen	tokyo-jaban	mosaken alam	\$ 17.00	115

Below the table, there is a large empty rectangular area. At the bottom of the window, there are four buttons: 'Add [F1]', 'Replacements [F2]', 'Delete [F3]', and 'Close'.

4.5.1 Products Flowchart



4.6 Purchase Menu

In Purchase process the program allow the user to purchase a product after selecting it from the products list and how many he wants to purchase so the list view will show the quantity purchased and the price for each and the total price. The screen and the flowchart Shows the whole process:

Purchase Menu
1. New Purchase 2. Remove Purchase

When the responsible of pharmacy want to purchase a new product he has to fill in the database related the product information as following:

Purchase File
1. Product Name
2. Brand
3. Category
4. Price

4.6.1 Purchase Menu Screen

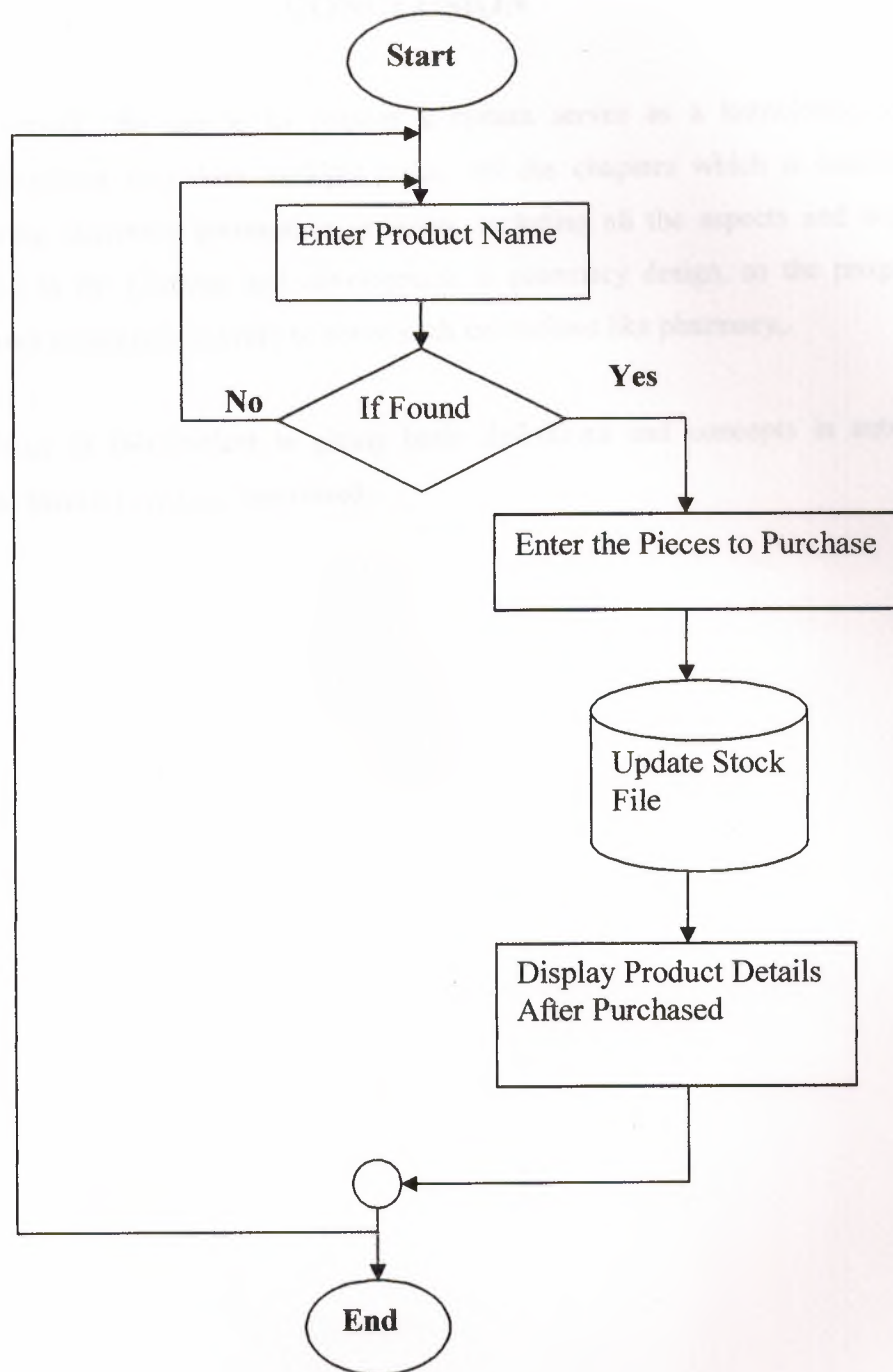
The screenshot shows a window titled "Pruchase" with a close button (X) in the top right corner. The window contains a table with four columns: "Product", "Qty", "Price", and "Total". The table has 10 rows, all of which are empty. Below the table, there are two buttons: "Add" and "Remove". To the right of these buttons is a label "Grand Total" followed by a text box containing "\$ 0.00". At the bottom of the window, there are two more buttons: "OK" and "Cancel".

Product	Qty	Price	Total

Add Remove Grand Total \$ 0.00

OK Cancel

4.6.2 Purchase Flowchart



CONCLUSION

As a result, the aim is to present a system serves as a knowledge base for automation programs that does multiple tasks. All the chapters which is construct my project covering pharmacy automation program, including all the aspects and steps that must be taken in the planning and development in pharmacy design, so the programmer he/she construct a program in order to serve such institutions like pharmacy,.

The main focus of this project is giving basic definitions and concepts in automation programs that almost every one interested.

REFERENCES

- [1] Visual Basic 6.0 How to Program deitel & deitel
- [2] Fundamental of Database by Addison wisely
- [3] Microsoft MSDN Library
- [4] www.planet-source-code.com
- [5] www.vbdiamond.com
- [6] www.freevbcode.com

APPENDIX A

Main Menu Screen



Sell Screens

The screenshot shows the 'Sell' application window. In the background, a table lists products with columns: Product, Qty, Price, and Total. The 'betaval' product is highlighted. In the foreground, two dialog boxes are open. The 'Select a Product' dialog shows a search for 'bet' and a table with one result: 'betaval' (Brand: betamethasone, Category: skin diseases). The 'Purchase' dialog asks 'How many pieces you want to purchase?' with '3' entered in the input field. Both dialogs have OK and Cancel buttons.

Product	Qty	Price	Total
betaval			

Select a Product

Search:

	Name	Brand	Category
▶	betaval	betamethasone	skin diseases

Purchase

How many pieces you want to purchase?

OK Cancel

OK Cancel

[illegible]

Customer Screen

The screenshot shows a window titled "Customers" with a search bar and a table of customer records. An "Add Customer" dialog box is open in the foreground, allowing the user to enter details for a new customer.

Search:

	Name	Phone	Suffers	Debt
▶	mofdi	05338774532	skin deseas	\$ 0.00
	annas	05338746621	enfelawanza	\$ 0.00

Add Customer

Name:

Phone:

Suffers:

Address:

Customer Details Screen

The screenshot shows a window titled "Customer Details" with fields for Name, Phone, Suffers, Address, and Debt. Below these fields are buttons for "Add [F1]" and "Delete [F3]". At the bottom, there is a table showing details for a specific customer, with buttons for "Change [F2]" and "Close".

Name:

Phone:

Suffers:

Address:

Debt:

	Name	Brand	Category
▶	betaval	betamethasone	skin diseases

Product Screen



Products

Search: Stock [F4]

	Name	Brand	Category	Price	Barcode
▶	betaval	betamethasone	skin diseases	\$ 6.00	111111
	paracetamol	cetamolthises	painful, feverish	\$ 4.00	111112

Add new Product

Name

Brand

Category

Price (bought)

Retail Price

Barcode

Replacements Screen

Products ✕

Search: Stock [F4]

	Name	Brand	Category	Price	Barcode
	betaval	betamethasone	skin diseases	\$ 6.00	111111
▶	paracetamol	cetamolthises	painful, feverish	\$ 4.00	111112
	cetamol	cetamolthises	painful, feverish	\$ 8.00	111113

Replacements ✕

	Name	Brand
▶	cetamol	cetamolthises

Add [F1]

Delete [F3]

Close

Add [F1]

Replacements [F5]

Delete [F3]

Close

Stock Screen

Stock

Search:

be

	Product	Amount
▶	betaval	50

Close

Purchase Screen

The screenshot shows a Windows-style application titled "Purchase". It features two overlapping dialog boxes. The top dialog, "Select a Product", has a search bar containing "par" and a table with the following data:

Name	Brand	Category
paracetamol	cetamolthises	painful, feverish

The bottom dialog, "Purchase", contains the text "How many pieces you want to purchase?" and a text input field with the value "30". Both dialogs have "OK" and "Cancel" buttons at the bottom.

[illegible]

APPENDIX B

Customer Table

Customer : Table						
		id	Name	Phone	Suffers	Address
	+	13	mofdi	05338774532	skin deseas	lefkosha
	+	14	annas	05338746621	enfelowanza	gocmenkoy
	+	16	qusai	0542887546	skin diseases	metropol
						Debt
						0.00 TL
						0.00 TL
						8.00 TL

Product Table

Product : Table								
		id	Name	Brand	Cat	Desc	Price	Bought
	+	16	betaval	betamethasone	skin diseases		6.00 TL	6.00 TL 111111
	+	17	paracetamol	cetamolthises	painful,feverish		4.00 TL	4.00 TL 111112
	+	19	cetamol	cetamolthises	painful,feverish		8.00 TL	8.00 TL 111113

Replacement Table

Replacements : Table			
	id	Product1	Product2
	2	9	7
	7	19	17

Stock Table

stock : Table			
	id	Product	Amount
	2	7	10
	3	9	84
	4	10	7
	5	14	9
	6	12	52
	7	13	10
	8	17	30
	9	19	15
	10	16	47

Uses Table

Uses : Table			
	id	Customer	Product
▶	15	4	7
	17	4	9
	18	3	9
	19	8	14
	20	8	12
	21	8	13
	22	8	14
	23	16	16

APPENDIX C

Main Form Codes

```
Private Sub cmdCustomers_Click()  
    frmCustomers.Show  
End Sub
```

```
Private Sub cmdExit_Click()  
End  
End Sub
```

```
Private Sub cmdProds_Click()  
    frmProds.Show  
End Sub
```

```
Private Sub cmdPurchase_Click()  
    frmPurchase.Show 1, Me  
End Sub
```

```
Private Sub cmdSell_Click()  
    frmSell.Show 1, Me  
End Sub
```


Sell Form Codes

```
Private Sub cmbCust_Click()
If cmbCust.ItemData(cmbCust.ListIndex) = -1 Then
    txtPaid.Enabled = False
Else
    txtPaid.Enabled = True
End If
End Sub

Private Sub cmdAdd_Click()
frmSelectProd.Show 1, frmMain
If frmSelectProd.Tag > -1 Then
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select name,bought from product where id=" &
frmSelectProd.Tag
    num = InputBox("How many pieces you want to purchase?", "Prurchase", 1)
    If num = "" Then Exit Sub
    'removing duplicates
    For i = 1 To lstProducts.ListItems.Count
        If lstProducts.ListItems(i).Tag = frmSelectProd.Tag Then
            lstProducts.ListItems.Remove i
        End If
    Next
    With lstProducts.ListItems.Add(, , adoProd.Recordset!Name)
        .SubItems(1) = num
        .SubItems(2) = Format(adoProd.Recordset!bought, "$ #,##0.00")
        .SubItems(3) = Format(adoProd.Recordset!bought * num, "$ #,##0.00")
        .Tag = frmSelectProd.Tag
    End With
End If
adoProd.Refresh
Sum = 0
```

```

For i = 1 To lstProducts.ListItems.Count
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select bought from product where id=" &
lstProducts.ListItems(i).Tag
    Sum = Sum + adoProd.Recordset!bought * lstProducts.ListItems(i).SubItems(1)
Next
txtPrice = Sum
End Sub

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOK_Click()
    On Error Resume Next
    adoProd.Refresh
    suff = True
    For i = 1 To lstProducts.ListItems.Count
        adoProd.Recordset.Close
        adoProd.Recordset.Open "select id from stock where product=" &
lstProducts.ListItems(i).Tag & " and amount>=" & lstProducts.ListItems(i).SubItems(1)
        If adoProd.Recordset.RecordCount = 0 Then
            suff = False
            Exit For
        End If
    Next

    If suff Then
        adoProd.Refresh
        For i = 1 To lstProducts.ListItems.Count
            adoProd.Recordset.Close
            adoProd.Recordset.Open "select id from stock where product=" &
lstProducts.ListItems(i).Tag
            id = adoProd.Recordset!id

```

```

    adoProd.Recordset.Close
    adoProd.Recordset.Open "update stock set amount=amount-" &
lstProducts.ListItems(i).SubItems(1) & " where id=" & id
Next
If txtPaid <> txtPrice Then
    adoProd.Recordset.Close
    adoProd.Recordset.Open "update customer set debt=debt+" & txtPrice.Text - txtPaid
    & " where id=" & cmbCust.ItemData(cmbCust.ListIndex)
End If
MsgBox "Your purchase is made!", vbInformation, "Sale"
Unload Me
Else
    MsgBox "Not enough products! Sale cannot be made.", vbCritical, "Sale"
End If
End Sub

```

```

Private Sub cmdRemove_Click()
On Error Resume Next
lstProducts.ListItems.Remove lstProducts.SelectedItem.Index

```

```

adoProd.Refresh
Sum = 0
For i = 1 To lstProducts.ListItems.Count
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select bought from product where id=" &
lstProducts.ListItems(i).Tag
    Sum = Sum + adoProd.Recordset!bought * lstProducts.ListItems(i).SubItems(1)
Next
txtPrice = Sum
End Sub

```

```

Private Sub Form_Load()
adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
App.Path & "\pharmacy.mdb;Persist Security Info=False"

```



```

adoProd.Refresh
lstProducts.ColumnHeaders.Add , , "Product"
lstProducts.ColumnHeaders.Add , , "Qty"
lstProducts.ColumnHeaders.Add , , "Price"
lstProducts.ColumnHeaders.Add , , "Total"
cmbCust.AddItem "[None]"
cmbCust.ItemData(0) = -1
adoProd.Recordset.Close
adoProd.Recordset.Open "select id,name from customer"
If adoProd.Recordset.RecordCount > 0 Then adoProd.Recordset.MoveFirst
For i = 0 To adoProd.Recordset.RecordCount - 1
    cmbCust.AddItem adoProd.Recordset!Name
    cmbCust.ItemData(i + 1) = adoProd.Recordset!id
    adoProd.Recordset.MoveNext
Next
cmbCust.ListIndex = 0
End Sub

```

```

Private Sub txtPrice_Change()
    txtPaid = txtPrice
End Sub

```

Customers Form Code

```
Private Sub cmdAdd_Click()  
frmAddCustomer.Show 1, frmMain  
adoCust.Refresh  
End Sub
```

```
Private Sub cmdClose_Click()  
Unload Me  
End Sub
```

```
Private Sub cmdDel_Click()  
id = adoCust.Recordset!id  
adoCust.Recordset.Close  
adoCust.Recordset.Open "delete from customer where id=" & id  
adoCust.Refresh  
End Sub
```

```
Private Sub cmdDetails_Click()  
frmCustomerDet.adoUses.RecordSource = "select uses.id,Name,Brand,cat as Category  
from uses inner join product on (uses.product=product.id) where uses.customer=" &  
adoCust.Recordset!id  
frmCustomerDet.adoUses.Refresh  
frmCustomerDet.adoCust.RecordSource = "select * from customer where id=" &  
adoCust.Recordset!id  
frmCustomerDet.adoCust.Refresh  
frmCustomerDet.Change  
frmCustomerDet.Show 1, frmMain  
adoCust.Refresh  
DataGrid1.Refresh  
End Sub
```

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
```

```
    Select Case KeyCode
```

```
    Case vbKeyF1
```

```
        cmdAdd_Click
```

```
    Case vbKeyF2
```

```
        cmdDetails_Click
```

```
    Case vbKeyF3
```

```
        cmdDel_Click
```

```
    End Select
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    adoCust.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
    App.Path & "\pharmacy.mdb;Persist Security Info=False"
```

```
    adoCust.Refresh
```

```
End Sub
```

```
Private Sub txtSrc_Change()
```

```
    adoCust.RecordSource = "select id,Name,Phone,Suffers,Debt from customer where  
    name like '" & txtSrc.Text & "%' or suffers like '%" & txtSrc.Text & "%"
```

```
    adoCust.Refresh
```

```
End Sub
```

Add Customer Form Code

```
Private Sub cmdAdd_Click()
```

```
    adoCust.Recordset.Close
```

```
    adoCust.Recordset.Open "insert into customer (name,phone,suffers,address) values('" &  
    txtName & "','" & txtPhone & "','" & txtSuffer & "','" & txtAddr & "')
```

```
    Unload Me
```

```
End Sub
```



```
Private Sub cmdClose_Click()
```

```
Unload Me
```

```
End Sub
```

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
```

```
Select Case KeyCode
```

```
Case vbKeyF1
```

```
cmdAdd_Click
```

```
End Select
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
adoCust.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
App.Path & "\pharmacy.mdb;Persist Security Info=False"
```

```
adoCust.Refresh
```

```
End Sub
```

Customer Details Form Code

```
Public Sub Change()
```

```
txtName = adoCust.Recordset!Name
```

```
txtPhone = adoCust.Recordset!phone
```

```
txtSuffer = adoCust.Recordset!suffers
```

```
txtAddr = adoCust.Recordset!address
```

```
txtDebt = adoCust.Recordset!debt
```

```
End Sub
```

```
Private Sub cmdAdd_Click()
```

```
frmSelectProd.Show 1, frmMain
```

```
If frmSelectProd.Tag > -1 Then
```

```
adoUses.Recordset.Close
```

```
adoUses.Recordset.Open "insert into uses (customer,product) values(" &  
adoCust.Recordset!id & "," & frmSelectProd.Tag & ")"
```

```
adoUses.Refresh
```

End If

End Sub

Private Sub cmdChange_Click()

id = adoCust.Recordset!id

adoCust.Recordset.Close

adoCust.Recordset.Open "update customer set name='" & txtName & "',phone='" &
txtPhone & "',suffers='" & txtSuffer & "',address='" & txtAddr & "',debt='" & txtDebt &
" where id=" & id

Unload Me

End Sub

Private Sub cmdClose_Click()

Unload Me

End Sub

Private Sub cmdDelete_Click()

On Error GoTo ErrH

id = adoUses.Recordset!id

adoUses.Recordset.Close

adoUses.Recordset.Open "delete from uses where id=" & id

adoUses.Refresh

ErrH:

End Sub

Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode

Case vbKeyF1

cmdAdd_Click

Case vbKeyF2

cmdChange_Click

Case vbKeyF3

cmdDelete_Click

End Select

End Sub

Private Sub Form_Load()

adoCust.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
App.Path & "\pharmacy.mdb;Persist Security Info=False"

adoCust.Refresh

adoUses.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
App.Path & "\pharmacy.mdb;Persist Security Info=False"

adoUses.Refresh

End Sub

Products Form Code

```
Private Sub cmdAdd_Click()  
frmAddProd.Show 1, frmMain  
adoProd.Refresh  
End Sub
```

```
Private Sub cmdClose_Click()  
Unload Me  
End Sub
```

```
Private Sub cmdDel_Click()  
id = adoProd.Recordset!id  
adoProd.Recordset.Close  
adoProd.Recordset.Open "delete from product where id=" & id  
adoProd.Refresh  
End Sub
```

```
Private Sub cmdCats_Click()  
frmStock.Show  
End Sub
```

```
Private Sub cmdSub_Click()  
frmRepl.adoProd.RecordSource = "(select replacements.id,Name,Brand from  
Replacements inner join Product on (replacements.product1=product.id) where  
product2=" & adoProd.Recordset!id & ") union (select replacements.id,Name,Brand  
from Replacements inner join Product on (replacements.product2=product.id) where  
product1=" & adoProd.Recordset!id & ")"  
frmRepl.adoProd.Refresh  
frmRepl.Tag = adoProd.Recordset!id  
frmRepl.Show 1, frmMain
```

```
adoProd.Refresh
```

```
End Sub
```

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
```

```
    Select Case KeyCode
```

```
    Case vbKeyF1
```

```
        cmdAdd_Click
```

```
    Case vbKeyF3
```

```
        cmdDel_Click
```

```
    Case vbKeyF4
```

```
        cmdCats_Click
```

```
    Case vbKeyF3
```

```
        cmdSub_Click
```

```
    End Select
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
```

```
    App.Path & "\pharmacy.mdb;Persist Security Info=False"
```

```
    adoProd.Refresh
```

```
End Sub
```

```
Private Sub txtSrc_Change()
```

```
    adoProd.RecordSource = "select Product.id as ID,Name,Brand,Cat as
```

```
    Category,Price,Barcode from Product where brand like '" & txtSrc.Text & "%' or name
```

```
    like '" & txtSrc.Text & "%' or cat like '" & txtSrc.Text & "%' or barcode='" &
```

```
    txtSrc.Text & """
```

```
    adoProd.Refresh
```

```
End Sub
```

Add Product Form Code

```
Private Sub cmdAdd_Click()  
    adoProd.Recordset.Close  
    adoProd.Recordset.Open "insert into product (name,brand,cat,price,bought,barcode)  
    values('" & txtName & "','" & txtBrand & "','" & txtCat & "','" & txtRetail & "','" &  
    txtRetail & "','" & txtBarcode & "')"  
    adoProd.Refresh  
    Unload Me  
End Sub
```

```
Private Sub cmdClose_Click()  
    Unload Me  
End Sub
```

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)  
    Select Case KeyCode  
        Case vbKeyF1  
            cmdAdd_Click  
    End Select  
End Sub  
Private Sub Form_Load()  
    adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
    App.Path & "\pharmacy.mdb;Persist Security Info=False"  
    adoProd.Refresh  
End Sub
```

Replacement Form Code

```
Private Sub cmdAdd_Click()  
    frmSelectProd.Show 1, frmMain  
    If frmSelectProd.Tag > -1 Then  
        adoProd.Recordset.Close
```



```
adoProd.Recordset.Open "insert into replacements (product1,product2) values(" &  
frmSelectProd.Tag & "," & Me.Tag & ")"
```

```
adoProd.Refresh
```

```
End If
```

```
End Sub
```

```
Private Sub cmdClose_Click()
```

```
Unload Me
```

```
End Sub
```

```
Private Sub cmdDel_Click()
```

```
id = adoProd.Recordset!id
```

```
adoProd.Recordset.Close
```

```
adoProd.Recordset.Open "delete from replacements where id=" & id
```

```
adoProd.Refresh
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
```

```
App.Path & "\pharmacy.mdb;Persist Security Info=False"
```

```
adoProd.Refresh
```

```
End Sub
```

Stock Form Code

```
Private Sub cmdClose_Click()
```

```
Unload Me
```

```
End Sub
```

```
Private Sub txtSrc_Change()
```

```
adoStock.RecordSource = "select stock.id as ID, product.name as Product,Amount from  
stock inner join product on (stock.product=product.id) where product.name like '" &  
txtSrc & "%' or product.barcode='" & txtSrc & "' or product.cat like '" & txtSrc & "%'"
```

```
adoStock.Refresh
```

```
End Sub
```

Select Product Form Code

```
Private Sub cmdCancel_Click()
```

```
Me.Tag = -1
```

```
Me.Hide
```

```
End Sub
```

```
Private Sub cmdOK_Click()
```

```
On Error Resume Next
```

```
Me.Tag = adoProd.Recordset!id
```

```
If Err Then Exit Sub
```

```
Me.Hide
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &  
App.Path & "\pharmacy.mdb;Persist Security Info=False"
```

```
adoProd.Refresh
```

```
Me.Tag = -1
```

```
End Sub
```

```
Private Sub txtSrc_Change()
```

```
adoProd.RecordSource = "select Product.id as ID,Name,Brand,Cat as Category from  
Product where brand like '" & txtSrc.Text & "%' or name like '" & txtSrc.Text & "%' or  
cat like '" & txtSrc.Text & "%' or barcode='" & txtSrc.Text & """
```

```
adoProd.Refresh
```

```
End Sub
```

Purchase Form Code

```
Private Sub cmdAdd_Click()
frmSelectProd.Show 1, frmMain
If frmSelectProd.Tag > -1 Then
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select name,bought from product where id=" &
frmSelectProd.Tag
    num = InputBox("How many pieces you want to purchase?", "Pruchase", 1)
    If num = "" Then Exit Sub
    With lstProducts.ListItems.Add(, , adoProd.Recordset!Name)
        .SubItems(1) = num
        .SubItems(2) = Format(adoProd.Recordset!bought, "$ #,##0.00")
        .SubItems(3) = Format(adoProd.Recordset!bought * num, "$ #,##0.00")
        .Tag = frmSelectProd.Tag
    End With
End If
adoProd.Refresh
Sum = 0
For i = 1 To lstProducts.ListItems.Count
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select bought from product where id=" &
lstProducts.ListItems(i).Tag
    Sum = Sum + adoProd.Recordset!bought * lstProducts.ListItems(i).SubItems(1)
Next
txtPrice = Format(Sum, "$ #,##0.00")
End Sub

Private Sub cmdCancel_Click()
Unload Me
End Sub

Private Sub cmdOK_Click()
```



```

On Error Resume Next
adoProd.Refresh
For i = 1 To lstProducts.ListItems.Count
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select id from stock where product=" &
lstProducts.ListItems(i).Tag
    If adoProd.Recordset.RecordCount = 0 Then
        adoProd.Recordset.Close
        adoProd.Recordset.Open "insert into stock (product,amount) values(" &
lstProducts.ListItems(i).Tag & "," & lstProducts.ListItems(i).SubItems(1) & ")"
    Else
        id = adoProd.Recordset!id
        adoProd.Recordset.Close
        adoProd.Recordset.Open "update stock set amount=amount+" &
lstProducts.ListItems(i).SubItems(1) & " where id=" & id
    End If
Next
MsgBox "Your purchase is made!", vbInformation, "Purchase"
Unload Me
End Sub

Private Sub cmdRemove_Click()
On Error Resume Next
lstProducts.ListItems.Remove lstProducts.SelectedItem.Index

adoProd.Refresh
Sum = 0
For i = 1 To lstProducts.ListItems.Count
    adoProd.Recordset.Close
    adoProd.Recordset.Open "select bought from product where id=" &
lstProducts.ListItems(i).Tag
    Sum = Sum + adoProd.Recordset!bought * lstProducts.ListItems(i).SubItems(1)
Next
txtPrice = Format(Sum, "$ #,##0.00")

```

End Sub

Private Sub Form_Load()

adoProd.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &

App.Path & "\pharmacy.mdb;Persist Security Info=False"

adoProd.Refresh

lstProducts.ColumnHeaders.Add , , "Product"

lstProducts.ColumnHeaders.Add , , "Qty"

lstProducts.ColumnHeaders.Add , , "Price"

lstProducts.ColumnHeaders.Add , , "Total"

End Sub

Login Form Code

```
Dim a
Private Sub Command1_Click()
If Text1.Text = "" And text2.Text = "" Then
MsgBox "Please Enter User or password"
Beep
Exit Sub
End If

If a >= 3 Then
MsgBox " Sorry.....Only Three Attempts"
End
End If

If Text1.Text = "watad" And text2.Text = "2000" Then
Unload Me
frmMain.Show
Else
a = a + 1
MsgBox "Sorry.. Who are you..?"
text2.SetFocus
text2.Text = ""
End If
End Sub

Private Sub text2_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then Command1_Click
End Sub

Private Sub Timer1_Timer()
If pic1.Visible = True Then
pic.Visible = True
pic1.Visible = False
```


User Control Code (maskedit and label)

'Default Property Values:

Const m_def_ColName = ""

Const m_def_VAlign = 1

Const m_def_HideSelection = True

Const m_def_LabelWidth = 1200

Const m_def_AutoSize = 0

'Const m_def_LabelWidth = 1500

Const m_def_Text = ""

'Property Variables:

Dim m_ColName As String

Dim m_VAlign As Integer

Dim m_HideSelection As Boolean

Dim m_BorderStyle As Integer

Dim m_LabelWidth As Integer

Dim m_AutoSize As Boolean

'Dim m_LabelWidth As Integer

Dim m_Text As String

'Event Declarations:

Event Click() 'MappingInfo=lbl,lbl,-1,Click

Event Change() 'MappingInfo=mask,mask,-1,Change

Event DblClick() 'MappingInfo=lbl,lbl,-1,DblClick

Event KeyDown(KeyCode As Integer, Shift As Integer) 'MappingInfo=mask,mask,-1,KeyDown

Event KeyPress(KeyAscii As Integer) 'MappingInfo=mask,mask,-1,KeyPress

Event KeyUp(KeyCode As Integer, Shift As Integer) 'MappingInfo=mask,mask,-1,KeyUp

Event MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single) 'MappingInfo=lbl,lbl,-1,MouseDown

Event MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single) 'MappingInfo=lbl,lbl,-1,MouseMove

Event MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single) 'MappingInfo=lbl,lbl,-1,MouseUp

Public Enum BStyle

[No Border] = 0

[Single] = 1

[Thin Raised] = 2

[Thick Raised] = 3

[Thin Inset] = 4

[Thick Inset] = 5

[Etched] = 6

[Bump] = 7

End Enum

Public Enum BordStyle

BS_None = 0

BS_Single = 1

End Enum

Public Enum VAlign

VA_Top = 0

VA_Center = 1

VA_Bottom = 2

End Enum

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=mask,mask,-1,AllowPrompt

Public Property Get AllowPrompt() As Boolean

AllowPrompt = mask.AllowPrompt

End Property

Public Property Let AllowPrompt(ByVal New-AllowPrompt As Boolean)

mask.AllowPrompt() = New-AllowPrompt

PropertyChanged "AllowPrompt"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

'MappingInfo=mask,mask,-1,AutoTab

Public Property Get AutoTab() As Boolean

AutoTab = mask.AutoTab

End Property

Public Property Let AutoTab(ByVal New_AutoTab As Boolean)

mask.AutoTab() = New_AutoTab

PropertyChanged "AutoTab"

End Property

Private Sub lbl_Click()

RaiseEvent Click

End Sub

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

'MappingInfo=lbl,lbl,-1,Caption

Public Property Get Caption() As String

Caption = lbl.Caption

End Property

Public Property Let Caption(ByVal New_Caption As String)

lbl.Caption() = New_Caption

If Me.AutoSize Then

Me.LabelWidth = lbl.Width

End If

PropertyChanged "Caption"

End Property

Private Sub mask_Change()

RaiseEvent Change

End Sub

Private Sub lbl_DblClick()

 RaiseEvent DblClick

End Sub

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=lbl,lbl,-1,Font

Public Property Get Font() As Font

 Set Font = lbl.Font

End Property

Public Property Set Font(ByVal New_Font As Font)

 Set lbl.Font = New_Font

 PropertyChanged "Font"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=lbl,lbl,-1,ForeColor

Public Property Get ForeColor() As OLE_COLOR

 ForeColor = lbl.ForeColor

End Property

Public Property Let ForeColor(ByVal New_ForeColor As OLE_COLOR)

 lbl.ForeColor() = New_ForeColor

 PropertyChanged "ForeColor"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=mask,mask,-1,Format

Public Property Get Format() As String

```
Format = mask.Format  
End Property
```

```
Public Property Let Format(ByVal New_Format As String)  
    mask.Format() = New_Format  
    mask.Text = mask.Text  
    PropertyChanged "Format"  
End Property
```

```
Private Sub mask_GotFocus()  
    mask.SelStart = 0  
    mask.SelLength = Len(mask.Text)  
End Sub
```

```
Private Sub mask_KeyDown(KeyCode As Integer, Shift As Integer)  
    RaiseEvent KeyDown(KeyCode, Shift)  
End Sub
```

```
Private Sub mask_KeyPress(KeyAscii As Integer)  
    RaiseEvent KeyPress(KeyAscii)  
End Sub
```

```
Private Sub mask_KeyUp(KeyCode As Integer, Shift As Integer)  
    RaiseEvent KeyUp(KeyCode, Shift)  
End Sub
```

```
'  
"WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED  
LINES!"
```

```
"MappingInfo=mask,mask,-1,Mask
```

```
'Public Property Get mask() As String
```

```
'    mask = mask.mask
```

```
'End Property
```

```
'
```

```
'Public Property Let mask(ByVal New_Mask As String)
```



```
' mask.mask() = New_Mask
' PropertyChanged "Mask"
'End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=mask,mask,-1,MaxLength
Public Property Get MaxLength() As Integer
    MaxLength = mask.MaxLength
End Property
```

```
Public Property Let MaxLength(ByVal New_MaxLength As Integer)
    mask.MaxLength() = New_MaxLength
    PropertyChanged "MaxLength"
End Property
```

```
Private Sub lbl_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    RaiseEvent MouseDown(Button, Shift, X, Y)
End Sub
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=lbl,lbl,-1,MouseIcon
Public Property Get MouseIcon() As Picture
    Set MouseIcon = lbl.MouseIcon
End Property
```

```
Public Property Set MouseIcon(ByVal New_MouseIcon As Picture)
    Set lbl.MouseIcon = New_MouseIcon
    PropertyChanged "MouseIcon"
End Property
```

```
Private Sub lbl_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
    RaiseEvent MouseMove(Button, Shift, X, Y)
```

```
End Sub
```

```
'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!
```

```
'MappingInfo=lbl,lbl,-1,MousePointer
```

```
Public Property Get MousePointer() As MousePointerConstants
```

```
    MousePointer = lbl.MousePointer
```

```
End Property
```

```
Public Property Let MousePointer(ByVal New_MousePointer As
```

```
MousePointerConstants)
```

```
    lbl.MousePointer() = New_MousePointer
```

```
    PropertyChanged "MousePointer"
```

```
End Property
```

```
Private Sub lbl_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
    RaiseEvent MouseUp(Button, Shift, X, Y)
```

```
End Sub
```

```
'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!
```

```
'MappingInfo=mask,mask,-1,PromptChar
```

```
Public Property Get PromptChar() As String
```

```
    PromptChar = mask.PromptChar
```

```
End Property
```

```
Public Property Let PromptChar(ByVal New_PromptChar As String)
```

```
    mask.PromptChar() = New_PromptChar
```

```
    PropertyChanged "PromptChar"
```

```
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=mask,mask,-1,PromptInclude
Public Property Get PromptInclude() As Boolean
    PromptInclude = mask.PromptInclude
End Property
```

```
Public Property Let PromptInclude(ByVal New_PromptInclude As Boolean)
    mask.PromptInclude() = New_PromptInclude
    PropertyChanged "PromptInclude"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=UserControl,UserControl,-1,TextHeight
Public Function TextHeight(ByVal Str As String) As Single
    TextHeight = UserControl.TextHeight(Str)
End Function
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=UserControl,UserControl,-1,TextWidth
Public Function TextWidth(ByVal Str As String) As Single
    TextWidth = UserControl.TextWidth(Str)
End Function
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```
'MappingInfo=mask,mask,-1,ToolTipText
Public Property Get ToolTipText() As String
    ToolTipText = mask.ToolTipText
End Property
```



```
Public Property Let ToolTipText(ByVal New_ToolTipText As String)
    mask.ToolTipText() = New_ToolTipText
    PropertyChanged "ToolTipText"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

```
'MappingInfo=lbl,lbl,-1,UseMnemonic
Public Property Get UseMnemonic() As Boolean
    UseMnemonic = lbl.UseMnemonic
End Property
```

```
Public Property Let UseMnemonic(ByVal New_UseMnemonic As Boolean)
    lbl.UseMnemonic() = New_UseMnemonic
    PropertyChanged "UseMnemonic"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

```
'MappingInfo=mask,mask,-1,ForeColor
Public Property Get ForeColorText() As OLE_COLOR
    ForeColorText = mask.ForeColor
End Property
```

```
Public Property Let ForeColorText(ByVal New_ForeColorText As OLE_COLOR)
    mask.ForeColor() = New_ForeColorText
    PropertyChanged "ForeColorText"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

```
'MappingInfo=mask,mask,-1,BackColor
Public Property Get BackColorText() As OLE_COLOR
    BackColorText = mask.BackColor
```

End Property

Public Property Let BackColorText(ByVal New_BackColorText As OLE_COLOR)

mask.BackColor() = New_BackColorText

PropertyChanged "BackColorText"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=mask,mask,-1,Font

Public Property Get FontText() As Font

Set FontText = mask.Font

End Property

Public Property Set FontText(ByVal New_FontText As Font)

Set mask.Font = New_FontText

PropertyChanged "FontText"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MemberInfo=13,0,0,

Public Property Get Text() As String

Text = mask.Text

End Property

Public Property Let Text(ByVal New_Text As String)

mask.Text = New_Text

PropertyChanged "Text"

End Property

'Initialize Properties for User Control

Private Sub UserControl_InitProperties()

m_Text = ""

```

m_AutoSize = m_def_AutoSize
m_LabelWidth = m_def_LabelWidth
m_BorderStyle = m_def_BorderStyle
Me.Caption = Extender.Name
m_VAlign = m_def_VAlign
m_ColName = m_def_ColName
End Sub

```

```

Private Sub UserControl_Paint()
Dim di As Long
Dim rc As RECT
Dim xTwips As Integer, yTwips As Integer
xTwips = Screen.TwipsPerPixelX
yTwips = Screen.TwipsPerPixelY
di = GetClientRect(UserControl.hwnd, rc)
Select Case m_BorderStyle
    Case [No Border]
        di = DrawEdge(UserControl.hdc, rc, BDR_RAISEDOUTER, BF_MONO)
    Case [Single]
        di = DrawEdge(UserControl.hdc, rc, BDR_RAISEDOUTER, BF_RECT Or
BF_MONO)
    Case [Thin Raised]
        di = DrawEdge(UserControl.hdc, rc, BDR_RAISEDINNER, BF_TOPLEFT)
        di = DrawEdge(UserControl.hdc, rc, BDR_RAISEDOUTER,
BF_BOTTOMRIGHT)
    Case [Thick Raised]
        di = DrawEdge(UserControl.hdc, rc, EDGE_RAISED, BF_TOPLEFT)
        di = DrawEdge(UserControl.hdc, rc, EDGE_RAISED, BF_BOTTOMRIGHT)
    Case [Thin Inset]
        di = DrawEdge(UserControl.hdc, rc, BDR_SUNKENINNER, BF_TOPLEFT)
        di = DrawEdge(UserControl.hdc, rc, BDR_SUNKENOUTER,
BF_BOTTOMRIGHT)
    Case [Thick Inset]
        di = DrawEdge(UserControl.hdc, rc, EDGE_SUNKEN, BF_TOPLEFT)

```



```

        di = DrawEdge(UserControl.hdc, rc, EDGE_SUNKEN, BF_BOTTOMRIGHT)
    Case [Etched]
        di = DrawEdge(UserControl.hdc, rc, EDGE_ETCHED, BF_TOPLEFT)
        di = DrawEdge(UserControl.hdc, rc, EDGE_ETCHED, BF_BOTTOMRIGHT)
    Case [Bump]
        di = DrawEdge(UserControl.hdc, rc, EDGE_BUMP, BF_TOPLEFT)
        di = DrawEdge(UserControl.hdc, rc, EDGE_BUMP, BF_BOTTOMRIGHT)
End Select

End Sub

'Load property values from storage
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
On Error Resume Next
' lbl.Alignment = PropBag.ReadProperty("Alignment", 0)
mask.AllowPrompt = PropBag.ReadProperty("AllowPrompt", False)
' lbl.AutoSize = PropBag.ReadProperty("AutoSize", False)
mask.AutoTab = PropBag.ReadProperty("AutoTab", False)
UserControl.BackColor = PropBag.ReadProperty("BackColor", &H8000000F)
' UserControl.BorderStyle = PropBag.ReadProperty("BorderStyle", 0)
lbl.Caption = PropBag.ReadProperty("Caption", "")
Set lbl.Font = PropBag.ReadProperty("Font", Ambient.Font)
lbl.ForeColor = PropBag.ReadProperty("ForeColor", &H80000012)
mask.Format = PropBag.ReadProperty("Format", "")
' mask.mask = PropBag.ReadProperty("Mask", "")
mask.MaxLength = PropBag.ReadProperty("MaxLength", 64)
Set MouseIcon = PropBag.ReadProperty("MouseIcon", Nothing)
lbl.MousePointer = PropBag.ReadProperty("MousePointer", 0)
mask.PromptChar = PropBag.ReadProperty("PromptChar", "_")
mask.PromptInclude = PropBag.ReadProperty("PromptInclude", True)
mask.ToolTipText = PropBag.ReadProperty("ToolTipText", "")
lbl.UseMnemonic = PropBag.ReadProperty("UseMnemonic", True)
mask.ForeColor = PropBag.ReadProperty("ForeColorText", &H80000008)
mask.BackColor = PropBag.ReadProperty("BackColorText", &H80000005)
Set mask.Font = PropBag.ReadProperty("FontText", Ambient.Font)

```

```

mask.Text = PropBag.ReadProperty("Text", m_def_Text)
mask.mask = PropBag.ReadProperty("MaskStr", "")
lbl.Width = PropBag.ReadProperty("LabelWidth", m_def_LabelWidth)
m_AutoSize = PropBag.ReadProperty("AutoSize", m_def_AutoSize)
m_LabelWidth = PropBag.ReadProperty("LabelWidth", m_def_LabelWidth)
m_BorderStyle = PropBag.ReadProperty("BorderStyle", m_def_BorderStyle)
mask.HideSelection = PropBag.ReadProperty("HideSelection", m_def_HideSelection)
mask.BorderStyle = PropBag.ReadProperty("BorderStyleText", 1)
mask.Appearance = PropBag.ReadProperty("ApperanceText", 1)
m_VAlign = PropBag.ReadProperty("VAlign", m_def_VAlign)
Set DataSource = PropBag.ReadProperty("DataSource", Nothing)
lbl.Enabled = PropBag.ReadProperty("Enabled", True)
mask.Enabled = PropBag.ReadProperty("Enabled", True)
m_ColName = PropBag.ReadProperty("ColName", m_def_ColName)
End Sub

Private Sub UserControl_Resize()
On Error Resume Next
mask.Height = UserControl.ScaleHeight - 60
mask.Left = Me.LabelWidth + 150
mask.Width = UserControl.ScaleWidth - (Me.LabelWidth + 180)
Select Case VAlign
Case 0
    lbl.Top = 30
Case 1
    lbl.Top = (UserControl.ScaleHeight - lbl.Height) / 2
Case 2
    lbl.Top = (UserControl.ScaleHeight - lbl.Height) - 30
End Select

If UserControl.Height < 300 Then UserControl.Height = 300
If UserControl.Width < 300 Then UserControl.Width = 300
End Sub

```



```
Private Sub UserControl_Show()
```

```
Me.AutoSize = Me.AutoSize
```

```
Me.Text = Me.Text
```

```
End Sub
```

```
'Write property values to storage
```

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
```

```
' Call PropBag.WriteProperty("Alignment", lbl.Alignment, 0)
```

```
Call PropBag.WriteProperty("AllowPrompt", mask.AllowPrompt, False)
```

```
' Call PropBag.WriteProperty("AutoSize", lbl.AutoSize, False)
```

```
Call PropBag.WriteProperty("AutoTab", mask.AutoTab, False)
```

```
Call PropBag.WriteProperty("BackColor", UserControl.BackColor, &H8000000F)
```

```
' Call PropBag.WriteProperty("BorderStyle", UserControl.BorderStyle, 0)
```

```
Call PropBag.WriteProperty("Caption", lbl.Caption, "")
```

```
Call PropBag.WriteProperty("Font", lbl.Font, Ambient.Font)
```

```
Call PropBag.WriteProperty("ForeColor", lbl.ForeColor, &H80000012)
```

```
Call PropBag.WriteProperty("Format", mask.Format, "")
```

```
' Call PropBag.WriteProperty("Mask", mask.mask, "")
```

```
Call PropBag.WriteProperty("MaxLength", mask.MaxLength, 64)
```

```
Call PropBag.WriteProperty("MouseIcon", MouseIcon, Nothing)
```

```
Call PropBag.WriteProperty("MousePointer", lbl.MousePointer, 0)
```

```
Call PropBag.WriteProperty("PromptChar", mask.PromptChar, "_")
```

```
Call PropBag.WriteProperty("PromptInclude", mask.PromptInclude, True)
```

```
Call PropBag.WriteProperty("ToolTipText", mask.ToolTipText, "")
```

```
Call PropBag.WriteProperty("UseMnemonic", lbl.UseMnemonic, True)
```

```
Call PropBag.WriteProperty("ForeColorText", mask.ForeColor, &H80000008)
```

```
Call PropBag.WriteProperty("BackColorText", mask.BackColor, &H80000005)
```

```
Call PropBag.WriteProperty("FontText", mask.Font, Ambient.Font)
```

```
Call PropBag.WriteProperty("Text", mask.Text, m_def_Text)
```

```
Call PropBag.WriteProperty("MaskStr", mask.mask, "")
```

```
Call PropBag.WriteProperty("LabelWidth", lbl.Width, m_def_LabelWidth)
```

```
Call PropBag.WriteProperty("AutoSize", m_AutoSize, m_def_AutoSize)
```

```
Call PropBag.WriteProperty("LabelWidth", lbl.Width, m_def_LabelWidth)
```



```

Call PropBag.WriteProperty("BorderStyle", m_BorderStyle, m_def_BorderStyle)
Call PropBag.WriteProperty("HideSelection", mask.HideSelection,
m_def_HideSelection)
Call PropBag.WriteProperty("BorderStyleText", mask.BorderStyle, 1)
Call PropBag.WriteProperty("ApperanceText", mask.Appearance, 1)
Call PropBag.WriteProperty("VAlign", m_VAlign, m_def_VAlign)
Call PropBag.WriteProperty("DataSource", DataSource, Nothing)
Call PropBag.WriteProperty("Enabled", mask.Enabled, True)
Call PropBag.WriteProperty("Enabled", mask.Enabled, True)
Call PropBag.WriteProperty("ColName", m_ColName, m_def_ColName)
End Sub

```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```

'MappingInfo=mask,mask,-1,Mask
Public Property Get MaskStr() As String
    MaskStr = mask.mask
End Property

```

```

Public Property Let MaskStr(ByVal New_MaskStr As String)
    mask.mask() = New_MaskStr
    PropertyChanged "MaskStr"
End Property

```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

```

'MemberInfo=0,0,0,0
Public Property Get AutoSize() As Boolean
    AutoSize = m_AutoSize
End Property

```

```

Public Property Let AutoSize(ByVal New_AutoSize As Boolean)
    If New_AutoSize Then
        Me.LabelWidth = lbl.Width
    End If
End Property

```

```
End If
m_AutoSize = New_AutoSize
PropertyChanged "AutoSize"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

```
'MemberInfo=7,0,0,1000
Public Property Get LabelWidth() As Integer
    LabelWidth = lbl.Width
End Property
```

```
Public Property Let LabelWidth(ByVal New_LabelWidth As Integer)
    lbl.Width = New_LabelWidth
    mask.Left = New_LabelWidth + 150
    mask.Width = UserControl.ScaleWidth - (New_LabelWidth + 180)

    PropertyChanged "LabelWidth"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

```
'MappingInfo=UserControl,UserControl,-1,BackColor
Public Property Get BackColor() As OLE_COLOR
    BackColor = UserControl.BackColor
End Property
```

```
Public Property Let BackColor(newv As OLE_COLOR)
    UserControl.BackColor = newv
    PropertyChanged "BackColor"
End Property
```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

'MemberInfo=7,0,0,

Public Property Get BorderStyle() As BStyle

BorderStyle = m_BorderStyle

End Property

Public Property Let BorderStyle(ByVal New_BorderStyle As BStyle)

m_BorderStyle = New_BorderStyle

UserControl_Paint

PropertyChanged "BorderStyle"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

'MemberInfo=0,0,0,True

Public Property Get HideSelection() As Boolean

HideSelection = mask.HideSelection

End Property

Public Property Let HideSelection(ByVal New_HideSelection As Boolean)

mask.HideSelection = New_HideSelection

PropertyChanged "HideSelection"

End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED LINES!

'MappingInfo=mask,mask,-1,BorderStyle

Public Property Get BorderStyleText() As BordStyle

BorderStyleText = mask.BorderStyle

End Property

Public Property Let BorderStyleText(ByVal New_BorderStyleText As BordStyle)

mask.BorderStyle() = New_BorderStyleText


```
PropertyChanged "BorderStyleText"
End Property
```

```
'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!
```

```
'MappingInfo=mask,mask,-1,Appearance
Public Property Get ApperanceText() As AppearanceConstants
    ApperanceText = mask.Appearance
End Property
```

```
Public Property Let ApperanceText(ByVal New_ApperanceText As
AppearanceConstants)
    mask.Appearance() = New_ApperanceText
    PropertyChanged "ApperanceText"
End Property
```

```
'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!
```

```
'MemberInfo=7,0,0,1
Public Property Get VAlign() As VAlign
    VAlign = m_VAlign
End Property
```

```
Public Property Let VAlign(ByVal New_VAlign As VAlign)
    m_VAlign = New_VAlign
    UserControl_Resize
    PropertyChanged "VAlign"
End Property
```

```
"
```

```
""WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!
```

```
""MappingInfo=mask,mask,-1,DataSource
"Public Property Get DataSource() As DataSource
" Set DataSource = mask.DataSource
```

```

"End Property
"

"Public Property Set DataSource(ByVal New_DataSource As DataSource)
" Set mask.DataSource = New_DataSource
" PropertyChanged "DataSource"
"End Property
"

"WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

"MappingInfo=UserControl,UserControl,-1,Enabled
'Public Property Get Enabled() As Boolean
' Enabled = UserControl.Enabled
'End Property
'

'Public Property Let Enabled(ByVal New_Enabled As Boolean)
' UserControl.Enabled() = New_Enabled
' lbl.Enabled = New_Enabled
' mask.Enabled = New_Enabled
' PropertyChanged "Enabled"
'End Property
'

"WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MappingInfo=mask,mask,-1,Enabled
Public Property Get Enabled() As Boolean
    Enabled = mask.Enabled
End Property

Public Property Let Enabled(ByVal New_Enabled As Boolean)
    mask.Enabled() = New_Enabled
    lbl.Enabled = New_Enabled
    PropertyChanged "Enabled"
End Property

```

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTED
LINES!

'MemberInfo=13,0,0,

Public Property Get ColName() As String

ColName = m_ColName

End Property

Public Property Let ColName(ByVal New_ColName As String)

m_ColName = New_ColName

PropertyChanged "ColName"

End Property