NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

FARMER INFORMATION SYSTEM IN DELPHI PROGRAMMING

Graduation Project
COM-400

Student: Bülent Durukan (970302)

Supervisor: Ms. Besime Erin

Nicosia- 2002

ACKNOWLEDGEMENTS

Alot of effort and time was put in the preparation of this program, which deals with farmer real estate management. Credit is due to the Instructor Ms. Besime Erin for her timely help and persistence throughout the preparation phase of the project.

I would also like to take this opportunity to thank my parents, without whom, I wouldn't have gone this far, and whose encouragement was as always of great help. Special credit is due to my friends for their friendly advice and constructive ideas, which forwarded the completion of this project. And last but not least I would like to thank almighty God, without whom nothing is possible.

ABSTRACT

The program basically targets those customers who belong to agricultural field. The main purpose of this program is the registration of critical data to a database, which is maintained and manipulated within this program. It includes space for vital information for Critical personal data, which has to be precise and as accurate as possible, as not to let any margin of error occur.

This program was made in the visual language called Delphi 5 licensed to Borland Inc.Delphi 5 is a very versatile and flexible language allowing the programmer maneuvering space in delicate programming issues and not hindering the course of his creative thinking. It's flexibility arises due to its derivative traits which are taken from many other famous languages allowing the programmer to manipulate freely his ideas on a canvas which is not limited in its ability to perform the tasks required.

Since it's a visual programming language, tedious base language programming is not required as in the case of other non-visual languages, because the base is already in place in the form of visual components ;only the logic and the its respective application is required.

Searching and updating capabilities are also provided, to perform vital search and locate procedures as to delete or update specific data sets. This can be done according to a wide variety of options, which accompany these functions. Interaction with other windows based programs is also provided such as the listing of specific data sets to Excel, which may be required if the user requires some manipulation to be done when required, Strong listing capabilities are also mingled in the program, as to clearly show the data when and if required.

A user-friendly interface is provided along with hints where possible of the usage of various functions. Accomplishment of tasks was not sacrificed at the cost of user friendliness, as the efficient achievement of goals remains the prime priority of the programmer.

In this program relational database, which provides sophisticated search and listing options is used. By that way the user is protected from entering data to the wrong persons information and confusion risk of records is reduced. The program checks whether the land number and plot of land number are same. If they are same the user is given a message that no two people can own the same land at the same time. This is done to prevent the goverment from being cheated.

TABLE OF CONTENTS

INTRODUCTION

Delphi 5 is a visual language consisting of VCL components. It is a derivative of the Pascal Programming language, but the language has been constructed as such that there are many features and commands inherited by Delphi, which are the corner stones of other basic or complex programming languages. It is often called Object based Pascal language since it incorporates features of the famous programming languages in a modular and an object oriented language.

Object Pascal is a high-level, compiled, strongly typed language that supports structured and object-oriented design. Its benefits include easy-to-read code, quick compilation, and the use of multiple unit files for modular programming.

Object Pascal has special features that support Delphi's component framework and RAD environment. For the most part, descriptions and examples in this language reference assume that you are using Object Pascal to develop Delphi applications. Delphi handles many details of setting up projects and source files, such as maintenance of dependency information among units. The main concept behind the project is the registration of a certain land mass to an authority as to prevent any misuse of the specific land mass in question by imposters or any intended or non-intentional misuse of data. Keeping this in mind, specific fields pertaining to the farmers personal as well as his professional Information are given.

Since, the goal of this project was the creation of a program which would intake a large amount of data, both personal as well as documented in official records, some databases were made which would accommodate a large amount of data and then were sorted into place. Their construction was imperative to the project as the large amount of data can, would be manipulated by the end-user, and has to be in a format both easy to use and efficient. Efficiency is pivotal in the construction of any program as the main goal of all programmer s is to attain their goal with minimum risk of data loss and maximum efficiency.

The databases were made in a program accompanying the whole Delphi package, and these types of databases are called paradox databases.

Specific field for the recognition of land mass were included, as to store enough information about the land mass as to prevent forgery. Specific fields for the valued assets of the farmer were also included including livestock as well as stationary assets. Field relating to official documents such as ID no, Card registration number etc.

Description of the type of ownership or rental status is also available for filling by the end user. This is vital as it describes the exact type of ownership status. Also, specific filed for the specific location its dimensions is provided, since it is the basic thing in question. Other very useful features are also provided.

The main concept that shouldn't be lost is that to maximize or ensure the integrity of the end users land mass, correct and authentic data should be registered with the proper authorities. This program serves as the missing link as it provides itself as tool for this very purpose.

WHAT IS DELPHI?

Delphi is Borland's best-selling rapid application development (RAD) product for writing Windows applications. With Delphi, you can write Windows programs more quickly and more easily than was ever possible before. You can create Win32 console applications or Win32 graphical user interface (GUI) programs. When creating Win32 GUI applications with Delphi, you have all the power of a true compiled programming language (Object Pascal) wrapped up in a RAD environment. What this means is that you can create the user interface to a program (the *user interface* means the menus, dialog boxes, main window, and so on) using drag-and-drop techniques for true rapid application development. You can also drop ActiveX controls on forms to create specialized programs such as Web browsers in a matter of minutes. Delphi gives you all this, and at virtually no cost: You don't sacrifice program execution speed because Delphi generates fast compiled code.

Object Pascal and the VCL

Back in 1994 or so, Borland began working on a RAD tool that it code-named Delphi. When it was decided that the component model architecture was the best way to implement RAD, it was then necessary to settle on the programming language that would be the heart of the system.

Object Pascal, a set of object-oriented extensions to standard Pascal, is the language of Delphi. The Visual Component Library (VCL) is a hierarchy of classes—written in Object Pascal and tied to the Delphi IDE—that allows you to develop applications quickly. Using Delphi's Component palette and Object Inspector, you can place VCL components on forms and manipulate their properties without writing code. All VCL objects descend from *TObject*, an abstract class whose methods encapsulate fundamental behavior like construction, destruction, and message handling. *Tobject* is the immediate ancestor of many simple classes. *Components* in the VCL descend from the abstract class *TComponent*. Components are objects that you can manipulate on forms at design time. Visual components—that is, components like *TForm* and *TSpeedButton* that appear on the screen at runtime—are called *controls*, and they descend from *TControl*. Despite its name, the VCL consists mostly of nonvisual objects. The Delphi IDE allows you to add many nonvisual components to your programs by dropping them onto forms. For example, if you were writing an application that connects to a database, you might place a *TDataSource* component on a form. Although

TDataSource is nonvisual, it is represented on the form by an icon (which doesn't appear at runtime). You can manipulate the properties and events of *TDataSource* in the Object Inspector just as you would those of a visual control. When you write classes of your own in Object Pascal, they should descend from *TObject*. By deriving new classes from the VCL's base class (or one of its descendants), you provide your classes with essential functionality and ensure that they work with the VCL.

The Delphi Workspace

The main part of the Delphi IDE is the workspace. The workspace initially displays the Form Designer. It should come as no surprise that the Form Designer enables you to create forms. In Delphi, a *form* represents a window in your program. The form might be the program's main window, a dialog box, or any other type of window. You use the Form Designer to place, move, and size components as part of the form creation process.

Hiding behind the Form Designer is the Code Editor. The Code Editor is where you type code when writing your programs. The Object Inspector, Form Designer, Code Editor, and Component palette work interactively as you build applications.

What is an Object?

An object, or *class*, is a data type that encapsulates *data* and *operations on data* in a single unit. Before object-oriented programming, data and operations (functions) were treated as separate elements. You can begin to understand objects if you understand Object Pascal *records*. Records (analogous to *structures* in C) are made of up fields that contain data, where each field has its own type. Records make it easy to refer to a collection of varied data elements. Objects are also collections of data elements. But objects—unlike records—contain procedures and functions that operate on their data. These procedures and functions are called *methods*. An object's data elements are accessed through *properties*. The properties of Delphi objects have values that you can change at design time without writing code. If you want a property value to change at runtime, you need to write only a small amount of code. The combination of data and functionality in a single unit is called *encapsulation*. In addition to encapsulation, object-oriented programming is characterized by *inheritance* and *polymorphism*. Inheritance means that objects derive functionality from other objects (called *ancestors*); objects can modify their inherited behavior. Polymorphism means

that different objects derived from the same ancestor support the same method and property interfaces, which often can be called interchangeably.

Private, Protected, Public and Published Declarations

When you declare a field, property, or method, the new member has a *visibility* indicated by one of the keywords private, protected, public, or published. The visibility of a member determines its accessibility to other objects and units.

- A private member is accessible only within the unit where it is declared. Private members are often used within a class to implement other (public or published) methods and properties.
- A protected member is accessible within the unit where its class is declared and within any descendant class, regardless of the descendant class's unit.
- A public member is accessible from wherever the object it belongs to is accessible—that is, from the unit where the class is declared and from any unit that uses that unit.
- A published member has the same visibility as a public member, but the compiler generates runtime type information for published members. Published properties appear in the Object Inspector at design time.

USING COMPONENTS

All components share features inherited from *TComponent*. By placing components on forms, you build the interface and functionality of your application. The standard components included with Delphi are sufficient for most application development, but you can extend the VCL by creating components of your own.

Delphi's standard components

The Component palette contains a selection of components that handle a wide variety of programming tasks. You can add, remove, and rearrange components on the palette, and you can create component *templates* and *frames* that group several components. The components on the palette are arranged in pages according to their purpose and functionality. Which pages appear in the default configuration depends on the version of Delphi you are running.

Page name	Contents
Standard	Standard Windows controls, menus
Additional	Additional controls
Win32	Windows 9x/NT 4.0 common controls
System	Components and controls for system-level access, including timers, multimedia, and DDE
Internet	Components for internet communication protocols and Web applications
Data Access	Nonvisual components for accessing databases tables, queries, and reports
Data Controls	Visual, data-aware controls
Decision Cube	Controls that let you summarize information from databases and view it from a variety of perspectives
QReport Quick	Report components for creating embedded reports
Dialogs	Windows common dialog boxes
Win 3.1	Components for compatibility with Delphi 1.0 projects
Samples	Sample custom components
ActiveX	Sample ActiveX controls
Midas	Components used for creating multi-tiered database applications

Properties common to visual components

All visual components (descendants of *TControl*) share certain properties including

- Position and size properties
- Display properties
- Parent properties
- Navigation properties
- Drag-and-drop properties
- Drag-and-dock properties

While these properties are inherited from *TControl*, they are published—and hence appear in the Object Inspector—only for components to which they are applicable. For example, *TImage* does not publish the *Color* property, since its color is determined by the graphic it displays.

Position and Size Properties

Four properties define the position and size of a control on a form:

- *Height* sets the vertical size
- *Width* sets the horizontal size
- *Top* positions the top edge
- *Left* positions the left edge

These properties aren't accessible in nonvisual components, but Delphi does keep track of where you place the component icons on your forms. Most of the time you'll set and alter these properties by manipulating the control's image on the form or using the Alignment palette. You can, however, alter them at runtime.

Display Properties

Four properties govern the general appearance of a control:

- *BorderStyle* specifies whether a control has a border.
- *Color* changes the background color of a control.
- *Ctrl3D* specifies whether a control will have a 3-D look or a flat border.
- *Font* changes the color, type family, style, or size of text.

Parent Properties

To maintain a consistent appearance across your application, you can make any control look like its container—called its *parent*—by setting the *parent...* properties to *True*. For example, if you place a button on a form and set the button's *ParentFont* property to *True*, changes to the form's *Font* property will automatically propagate to the button (and to the form's other children). Later, if you change the button's *Font* property, your font choice will take effect and the *ParentFont* property will revert to *False*.

Navigation Properties

Several properties determine how users navigate among the controls in a form:

- *Caption* contains the text string that labels a component. To underline a character in a string, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the control or menu item by pressing Alt while typing the underlined character.

• *TabOrder* indicates the position of the control in its parent's tab order, the order in which controls receive focus when the user presses the Tab key. Initially, tab order is the order in which the components are added to the form, but you can change this by changing *TabOrder*. *TabOrder* is meaningful only if *TabStop* is *True*.

• *TabStop* determines whether the user can tab to a control. If *TabStop* is *True*, the control is in the tab order.

Drag-and-Drop Properties

Two component properties affect drag-and-drop behavior:

• *DragMode* determines how dragging starts. By default, *DragMode* is *dmManual*, and the application must call the *BeginDrag* method to start dragging. When *DragMode* is *dmAutomatic*, dragging starts as soon as the mouse button goes down.

• *DragCursor* determines the shape of the mouse pointer when it is over a draggable component.

• *DockSite*

• *DragKind*

• *DragMode*

• *FloatingDockSiteClass*

• *AutoSize*

Text Controls

Many applications present text to the user or allow the user to enter text. The type of control used for this purpose depends on the size and format of the information.

Used components:

Edit	Edit a single line of text
Memo	Edit multiple lines of text
MaskEdit	Adhere to a particular format, such as a postal code or phone number
RichEdit	Edit multiple lines of text using rich text format

Properties Common to All Text Controls

All of the text controls have these properties in common:

• *Text* determines the text that appears in the edit box or memo control.

• *CharCase* forces the case of the text being entered to lowercase or uppercase.

• *ReadOnly* specifies whether the user is allowed to change the text.

• *MaxLength* limits the number of characters in the control.

• *PasswordChar* hides the text by displaying a single character (usually an asterisk).

• *HideSelection* specifies whether selected text remains highlighted when the control does not have focus.

Properties shared by memo and rich text controls Memo and rich text controls, which handle multiple lines of text, have several properties in common:

• *Alignment* specifies how text is aligned (left, right, or center) in the component.

•The *Text* property contains the text in the control. Your application can tell if the text changes by checking the *Modified* property.

• *Lines* contains the text as a list of strings.

• *OEMConvert* determines whether the text is temporarily converted from ANSI to OEM as it is entered. This is useful for validating file names.

• *WordWrap* determines whether the text will wrap at the right margin.

• *WantReturns* determines whether the user can insert hard returns in the text.

• *WantTabs* determines whether the user can insert tabs in the text.

• *AutoSelect* determines whether the text is automatically selected (highlighted) when the control becomes active.

• *SelText* contains the currently selected (highlighted) part of the text.

• *SelStart* and *SelLength* indicate the position and length of the selected part of the text.

At runtime, you can select all the text in the memo with the *SelectAll* method.

Rich Text Controls

The rich edit component is a memo control that supports rich text formatting, printing, searching, and drag-and-drop of text. It allows you to specify font properties, alignment, tabs, indentation, and numbering.

The following components provide additional ways of capturing input.

ScrollBar	Select values on a continuous range
TrackBar	Select values on a continuous range (more visually effective than scroll bar)
UpDown	Select a value from a spinner attached to an edit component
HotKey	Enter Ctrl/ Shift/ Alt keyboard sequences

Buttons and Similar Controls

A side from menus, buttons provide the most common way to invoke a command in an application. Delphi offers several button-like controls:

Button Present command choices on buttons with text

BitBtn Present command choices on buttons with text and glyphs

SpeedButton Create grouped toolbar buttons

CheckBox Present on/off options

RadioButton Present a set of mutually exclusive choices

ToolBar Arrange tool buttons and other controls in rows and automatically adjust their sizes and positions.

CoolBar Display a collection of windowed controls within movable, resizable bands

Button controls

Users click button controls to initiate actions. Double-clicking a button at design time takes you to the button's *OnClick* event handler in the Code editor.

•Set *Cancel* to *True* if you want the button to trigger its *OnClick* event when the user presses Esc.

•Set *Default* to *True* if you want the Enter key to trigger the button's *OnClick* event.

Bitmap buttons

A bitmap button *(BitBtn)* is a button control that presents a bitmap image on its face.

• To choose a bitmap for your button, set the *Glyph* property.

•Use *Kind* to automatically configure a button with a glyph and default behavior.

• By default, the glyph is to the left of any text. To move it, use the *Layout* property.

• The glyph and text are automatically centered in the button. To move their position, use the *Margin* property. *Margin* determines the number of pixels between the edge of the image and the edge of the button.

• By default, the image and the text are separated by 4 pixels. Use *Spacing* to increase or decrease the distance.

• Bitmap buttons can have 3 states: up, down, and held down. Set the *NumGlyphs* property to 3 to show a different bitmap for each state.

Speed Buttons

Speed buttons, which usually have images on their faces, can function in groups. They are commonly used with panels to create toolbars.

• To make speed buttons act as a group, give the *GroupIndex* property of all the buttons the same nonzero value.

• By default, speed buttons appear in an up (unselected) state. To initially display a speed button as selected, set the *Down* property to *True*.

•If *AllowAllUp* is *True*, all of the speed buttons in a group can be unselected. Set *AllowAllUp* to *False* if you want a group of buttons to act like a radio group.

Check Boxes

A check box is a toggle that presents the user with two, or sometimes three, choices.

•Set *Checked* to *True* to make the box appear checked by default.

•Set *AllowGrayed* to *True* to give the check box three possible states: checked, unchecked, and grayed.

•The *State* property indicates whether the check box is checked *(cbChecked)*, unchecked *(cbUnchecked)*, or grayed *(cbGrayed)*.

Radio Buttons

Radio buttons present a set of mutually exclusive choices. You can use individual radio buttons or the *radio group* component, which arranges groups of radio buttons automatically.

Toolbars

Toolbars provide an easy way to arrange and manage visual controls. You can create a toolbar out of a panel component and speed buttons, or you can use the *ToolBar* component, then right-click and choose New Button to add buttons to the toolbar. The *ToolBar* component has several advantages: Buttons on a toolbar automatically maintain uniform dimensions and spacing; other controls maintain their relative position and height; controls can automatically wrap around to start a new row when they do not fit horizontally; and the *ToolBar* offers display options like transparency, pop-up borders, and spaces and dividers to group controls.

Handling Lists

Lists present the user with a collection of items to select from. Several components display lists: Used the nonvisual *TStringList* and *TImageList* components to manage sets of strings and images.

Used components:

ListBox	A list of text strings
CheckListBox	A list with a check box in front of each item
ComboBox	An edit box with a scrollable drop-down list
TreeView	A hierarchical list
ListView	A list of (draggable) items with optional icons, columns, and headings
DateTimePicker	A list box for entering dates or times
MonthCalendar	A calendar for selecting dates

List Boxes and Check-list Boxes

List boxes and check-list boxes display lists from which users can select items.

• *Items* uses a *TStringList* object to fill the control with values.

• *ItemIndex* indicates which item in the list is selected.

• *MultiSelect* specifies whether a user can select more than one item at a time.

• *Sorted* determines whether the list is arranged alphabetically.

• *Columns* specifies the number of columns in the list control.

• *IntegralHeight* specifies whether the list box shows only entries that fit completely in the vertical space.

• *ItemHeight* specifies the height of each item in pixels. The *Style* property can cause *ItemHeight* to be ignored.

•The *Style* property determines how a list control displays its items. By default, items are displayed as strings. By changing the value of *Style*, you can create *owner-draw* list boxes that display items graphically or in varying heights.

Combo Boxes

A combo box combines an edit box with a scrollable list. When users enter data into the control—by typing or selecting from the list—the value of the *Text* property changes. Use the *Style* property to select the type of combo box you need:

•Use *csDropdown* if you want an edit box with a drop-down list. Use *csDropDownList* to make the edit box read-only (forcing users to choose from the list). Set the *DropDownCount* property to change the number of items displayed in the list.

•Use *csSimple* to create a combo box with a fixed list that does not close. Be sure to resize the combo box so that the list items are displayed.

•Use *csOwnerDrawFixed* or *csOwnerDrawVariable* to create *owner-draw* combo boxes that display items graphically or in varying heights.

Grouping Components

A graphical interface is easier to use when related controls and information are presented in groups. Delphi provides several components for grouping components:

Used components:

GroupBox A standard group box with a title

RadioGroup A simple group of radio buttons

Panel A more visually flexible group of controls

ScrollBox A scrollable region containing controls

TabControl A set of mutually exclusive notebook-style tabs

PageControl A set of mutually exclusive notebook-style tabs with corresponding pages, each of which may contain other controls HeaderControl Resizable column headers.

Group Boxes and Radio Groups

A group box is a standard Windows component that arranges related controls on a form. The most commonly grouped controls are radio buttons. After placing a group box on a form, select components from the Component palette and place them in the group box. The *Caption* property contains text that labels the group box at runtime. The radio group component simplifies the task of assembling radio buttons and making them work together. To add radio buttons to a radio group, edit the *Items* property in the Object Inspector; each string in *Items* makes a radio button appear in the group box with the string as its caption. The value of the *ItemIndex* property determines which radio button is currently selected. Display the radio buttons in a single column or in multiple columns by setting the value of the *Columns* property. To respace the buttons, resize the radio group component.

Panels

The panel component provides a generic container for other controls. Panels can be aligned with the form to maintain the same relative position when the form is resized. The *BorderWidth* property determines the width, in pixels, of the border around a panel.

Visual Feedback

There are many ways to provide users with information about the state of an application. For example, some components—including *TForm*—have a *Caption* property that can be set at runtime. You can also create dialog boxes to display messages. In addition, the following components are especially useful for providing visual feedback at runtime.

Used component:

Label and StaticText	Display non-editable text
StatusBar	Display a status region (usually at the bottom of a window)
ProgressBar	Show the amount of work completed for a particular task
Hint and ShowHint	Activate fly-by or "tool-tip" help
HelpContext and HelpFile	Link context-sensitive online Help

Labels and Static-Text Components

Labels display text and are usually placed next to other controls. The standard label component, *TLabel*, is a non windowed control, so it cannot receive focus; when you need a label with a window handle, use *TStaticText* instead. Label properties include the following:

• *Caption* contains the text string for the label.

• *FocusControl* links the label to another control on the form. If *Caption* includes an accelerator key, the control specified by *FocusControl* receives focus when the user presses the accelerator key.

• *ShowAccelChar* determines whether the label can display an underlined accelerator character. If *ShowAccelChar* is *True*, any character preceded by an ampersand (&) appears underlined and enables an accelerator key.

• *Transparent* determines whether items under the label (such as graphics) are visible.

Help and hint properties

Most visual controls can display context-sensitive Help as well as fly-by hints at runtime. The *HelpContext* and *HelpFile* properties establish a Help context number and Help file for the control. The *Hint* property contains the text string that appears when the user moves the mouse pointer over a control or menu item. To enable hints, set *ShowHint* to *True*; setting *ParentShowHint* to *True* causes the control's *ShowHint* property to have the same value as its parent's.

Grids

Grids display information in rows and columns. If you're writing a database application, use the *TDBGrid* or *TDBCtrlGrid*. Otherwise, use a standard draw grid or string grid.

Graphic Display

The following components make it easy to incorporate graphics into an application.

Used components to display:

Image	Graphics files
Shape	Geometric shapes
Bevel	3D lines and frames
PaintBox	Graphics drawn by your program at runtime
Animate	AVI files

Images

The image component displays a graphical image, like a bitmap, icon, or metafile. The *Picture* property determines the graphic to be displayed. Use *Center*, *AutoSize*, *Stretch*, and *Transparent* to set display options.

Shapes

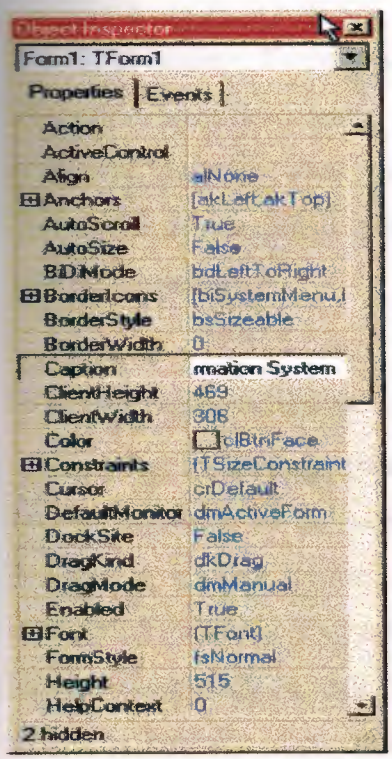
The shape component displays a geometric shape. It is a non windowed control and cannot receive user input. The *Shape* property determines which shape the control assumes. To change the shape's color or add a pattern, use the *Brush* property, which holds a *TBrush* object. How the shape is painted depends on the *Color* and *Style* properties of *TBrush*.

SETTING COMPONENT PROPERTIES

Published properties can be set at design time in the Object Inspector and, in some cases, with special property editors. To set properties at runtime, assign them new values in your application source code

Using the Object Inspector

When you select a component on a form, the Object Inspector displays its published properties and (when appropriate) allows you to edit them. Use the Tab key to toggle between the Value column and the Property column. When the cursor is in the Property column, you can navigate to any property by typing the first letters of its name. For properties of Boolean or enumerated types, you can choose values from a drop-down list or toggle their settings by double-clicking in Value column. If a plus (+) symbol appears next to a property name, clicking the plus symbol displays a list of sub values for the property. By default, properties in the Legacy category are not shown; to change the display filters, right-click in the Object Inspector and choose View. When more than one component is selected, the Object Inspector displays all properties—except *Name*—that are shared by the selected components. If the value for a shared property differs among the selected components, the Object Inspector displays either the default value or the value from the first component selected. When you change a shared property, the change applies to all selected components.



Using Property Editors

Properties give the application developer the illusion of setting or reading the value of a variable, while allowing the component writer to hide the underlying data structure or to implement special processing when the value is accessed. There are several advantages to using properties:

• Properties are available at design time. The application developer can set or change initial values of properties without having to write code.

• Properties can check values or formats as the application developer assigns them. Validating input at design time prevents errors.

• The component can construct appropriate values on demand. Perhaps the most common type of error programmers make is to reference a variable that has not been initialized. By representing data with a property, you can ensure that a value is always available on demand.

• Properties allow you to hide data under a simple, consistent interface. You can alter the way information is structured in a property without making the change visible to application developers.

Working with Events

An event is a special property that invokes code in response to input or other activity at runtime. Events give the application developer a way to attach specific blocks of code to specific runtime occurrences, such as mouse actions and keystrokes. The code that executes when an event occurs is called an *event handler*. Events allow application developers to specify responses to different kinds of input without defining new components.

Working with Methods

Class methods are procedures and functions that operate on a class rather than on specific instances of the class. For example, every component's constructor method *(Create*) is a class method. Component methods are procedures and functions that operate on the component instances themselves. Application developers use methods to direct a component to perform a specific action or return a value not contained by any property. Because they require execution of code, methods can be called only at runtime. Methods are useful for several reasons:

• Methods encapsulate the functionality of a component in the same object where the data resides.

• Methods can hide complicated procedures under a simple, consistent interface. An application developer can call a component's *AlignControls* method without knowing how the method works or how it differs from the *AlignControls* method in another component.

• Methods allow updating of several properties with a single call.

Table 1.1. Data types used in Delphi 5 (32-bit programs).

Data Type	*Size in Bytes*	*Possible Range of Values*
ShortInt	1	-128 to 127
Byte	1	0 to 255
Word	2	0 to 65,535
LongInt	4	-2,147,483,648 to 2,147,483,647
Integer	4	Same as LongInt
Real	8	$5.0 ¥ 10^{-324}$ to $1.7 ¥ 10^{308}$ (same as Double)
Currency	8	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Boolean	1	True or False
Variant	16	Varies

Table 1.2. String manipulation functions and procedures.

Name	*Description*
Copy	Returns a sub-string within a string.
Delete	Deletes part of a string.
Format	Formats and returns a string based on the format string and arguments passed.
Insert	Inserts text into a string.
IntToStr	Converts an integer value to a string.
Length	Returns the length of a string.
LowerCase	Converts a string to lowercase.
StrToInt	Converts a string to an integer. If the string cannot be converted, an exception is thrown.
StrToXXX	Additional conversion functions that convert a string to a floating point, Currency, Date, or Time value.
UpperCase	Converts a string to uppercase.

TYPES OF DATABASES

You can connect to different types of databases, depending on what drivers you have installed with the BDE or ADO. These drivers may connect your application to local databases such as Paradox, Access, and dBASE or remote database servers like Microsoft SQL Server, Oracle, and Informix. Similarly, the InterBase Express components can access either a local or remote version of InterBase..

Local Databases

Local databases reside on your local drive or on a local area network. They have proprietary APIs for accessing the data. Often, they are dedicated to a single system. When they are shared by several users, they use file-based locking mechanisms. Because of this, they are sometimes called file-based databases. Local databases can be faster than remote database servers because they often reside on the same system as the database application. Because they are file-based, local databases are more limited than remote database servers in the amount of data they can store. Therefore, in deciding whether to use a local database, you must consider how much data the tables are expected to hold. Applications that use local databases are called single-tiered applications because the application and the database share a single file system. Examples of local databases include Paradox, dBASE, FoxPro, and Access.

Connecting to Databases

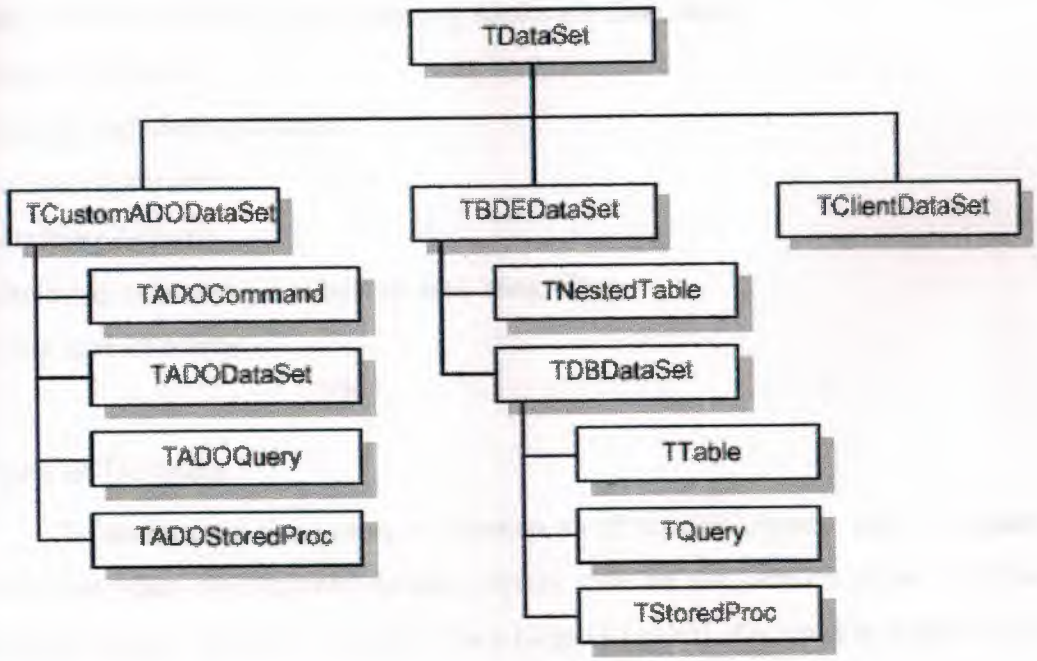
The Borland Database Engine includes drivers to connect to different databases. The Standard version of Delphi includes only the drivers for local databases: Paradox, dBASE, FoxPro, and Access. With the Professional version, you also get an ODBC adapter that allows the BDE to use ODBC drivers. By supplying an ODBC driver, your application can use any ODBC-compliant database. Some versions also include drivers for remote database servers. Use the drivers installed with SQL Links to communicate with remote database servers such as InterBase, Oracle, Sybase, Informix, Microsoft SQL server, and DB2.

Understanding Datasets

In Delphi, the fundamental unit for accessing data is the dataset family of objects. Your application uses datasets for all database access. Generally, a dataset object represents a specific table belonging to a database, or it represents a query or

stored procedure that accesses a database. All dataset objects that you will use in your database applications descend from the virtualized dataset object, *TDataSet*, and they inherit data fields, properties, events, and methods from *TDataSet*. This chapter describes the functionality of *TDataSet* that is inherited by the dataset objects you will use in your database applications. You need to understand this shared functionality to use any dataset object. Figure 1.1 illustrates the hierarchical relationship of all the dataset components:

Figure 1 Delphi Dataset hierarchies



What is TDataSet?

TDataSet is the ancestor for all dataset objects you use in your applications. It defines a set of data fields, properties, events, and methods shared by all dataset objects. *TDataSet* is a virtualized dataset, meaning that many of its properties and methods are virtual or abstract. A *virtual method* is a function or procedure declaration where the implementation of that method can be (and usually is) overridden in descendant objects. An *abstract method* is a function or procedure declaration without an actual implementation. The declaration is a prototype that describes the method (and its parameters and return type, if any) that must be implemented in all descendant dataset

objects, but that might be implemented differently by each of them. Because *TDataSet* contains abstract methods, you cannot use it directly in an application without generating a runtime error. Instead, you either create instances of *TDataSet*'s descendants, such as *TTable*, *TQuery*, *TStoredProc*, and *TClientDataSet*, and use them in your application, or you derive your own dataset object from *TDataSet* or its descendants and write implementations for all its abstract methods. Nevertheless, *TDataSet* defines much that is common to all dataset objects. For example, *TDataSet* defines the basic structure of all datasets: an array of *Tfield* components that correspond to actual columns in one or more database tables, lookup fields provided by your application, or calculated fields provided by your application. For more information about *TField* components, the following topics are discussed:

- Types of datasets
- Opening and closing datasets
- Navigating datasets
- Searching datasets
- Displaying and editing a subset of data using filters
- Using dataset events

Types of Datasets

To understand the concepts common to all dataset objects, and to prepare for developing your own custom dataset objects that do not rely on either the Borland Database Engine (BDE) or ActiveX Data Objects (ADO). To develop traditional, two-tier client/server database applications using the Borland Database Engine (BDE),. That section introduces *TBDEDataSet* and *TDBDataSet*, and focuses on the shared features of *TQuery*, *TStoredProc*, and *TTable*, the dataset components used most commonly in all database applications. With some versions of Delphi, you can develop multi-tier database applications using distributed datasets.

Opening and Closing Datasets

To read or write data in a table or through a query, an application must first open a dataset. You can open a dataset in two ways,

•Set the *Active* property of the dataset to *True*, either at design time in the Object Inspector

• Call the *Open* method for the dataset at runtime,

You can close a dataset in two ways,

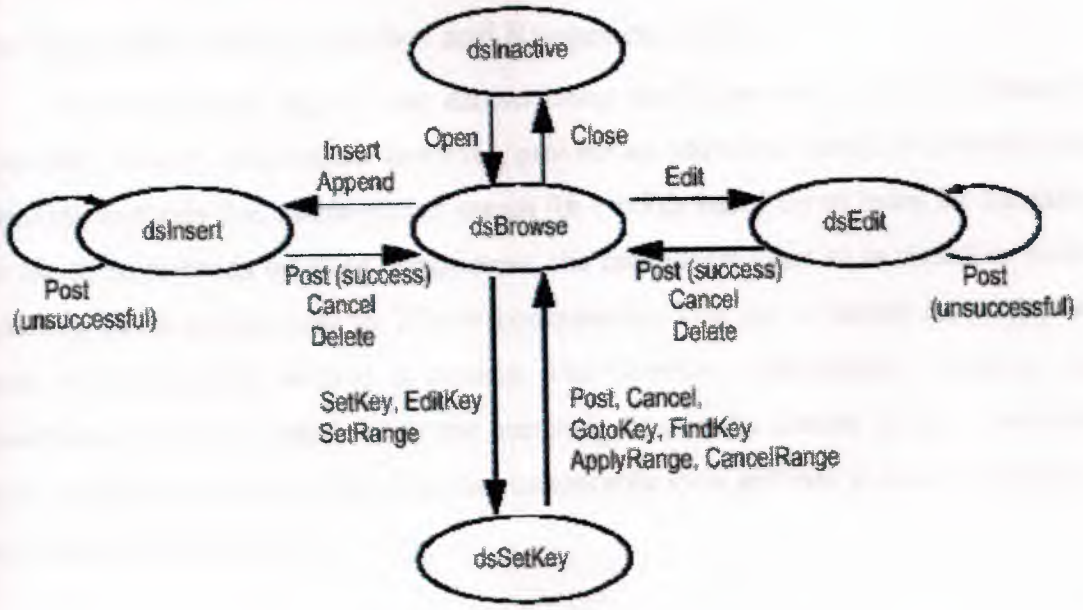
•Set t he *Active* property of the dataset to *False*, either at design time in the Object Inspector,

• Call the *Close* method for the dataset at runtime.

Browsing a Dataset

When an application opens a dataset, the dataset automatically enters *dsBrowse* state. Browsing enables you to view records in a dataset, but you cannot edit records or insert new records. You mainly use *dsBrowse* to scroll from record to record in a dataset. From *dsBrowse* all other dataset states can be set. For example, calling the *Insert* or *Append* methods for a dataset changes its state from *dsBrowse* to *dsInsert* (note that other factors and dataset properties, such as *CanModify*, may prevent this change). Calling *SetKey* to search for records puts a dataset in *dsSetKey* mode. Two methods associated with all datasets can return a dataset to *dsBrowse* state. *Cancel* ends the current edit, insert, or search task, and always returns a dataset to *dsBrowse* state. *Post* attempts to write changes to the database, and if successful, also returns a dataset to *dsBrowse* state. If *Post* fails, the current state remains unchanged. The following diagram illustrates the relationship of *dsBrowse* both to the other dataset modes you can set in your applications, and the methods that set those modes.

Figure 1.2 Relationship of Browse to other dataset states



Enabling Dataset Editing

A dataset must be in *dsEdit* mode before an application can modify records. In your code you can use the *Edit* method to put a dataset into *dsEdit* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, some data-aware controls can automatically put a dataset into *dsEdit* state if:

•The control's *ReadOnly* property is *False* (the default),

•The *AutoEdit* property of the data source for the control is *True*, and

• *CanModify* is *True* for the dataset.

Enabling Insertion of New Records

A dataset must be in *dsInsert* mode before an application can add new records. In your code you can use the *Insert* or *Append* methods to put a dataset into *dsInsert* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, the data-aware grid and navigator controls can put a dataset into *dsInsert* state if

•The control's *ReadOnly* property is *False* (the default),

•The *AutoEdit* property of the data source for the control is *True*, and

• *CanModify* is *True* for the dataset.

Enabling Index-Based Searches and Ranges on Tables

You can search against any dataset using the *Locate* and *Lookup* methods of *TDataSet*. *TTable* components, however, provide an additional family of *GotoKey* and *FindKey* methods that enable you to search for records based on an index for the table. To use these methods on table components, the component must be in *dsSetKey* mode. *dsSetKey* mode applies only to *TTable* components. You put a dataset into *dsSetKey* mode with the *SetKey* method at runtime. The *GotoKey*, *GotoNearest*, *FindKey*, and *FindNearest* methods, which carry out searches, returns the dataset to *dsBrowse* state upon completion of the search. You can temporarily view and edit a subset of data for any dataset by using filters.

Filtering Records

Helphi puts a dataset into *dsFilter* mode whenever an application calls the dataset's *OnFilterRecord* event handler. This state prevents modifications or additions to the records in a dataset during the filtering process so that the filter request is not invalidated. When the *OnFilterRecord* handler finishes, the dataset is returned to *dsBrowse* state.

Updating Records

When performing cached update operations, Delphi may put the dataset into *dsNewValue*, *dsOldValue*, or *dsCurValue* states temporarily. These states indicate that the corresponding properties of a field component *(NewValue*, *OldValue*, and *CurValue*, respectively) are being accessed, usually in an *OnUpdateError* event handler. Your applications cannot see or set these states.

Using the Eof and Bof Properties

Two read-only, runtime properties, *Eof* (End-of-file) and *Bof* (Beginning-of-file), are useful for controlling dataset navigation, particularly when you want to iterate through all records in a dataset.

End of File (Eof)

When *Eof* is *True*, it indicates that the cursor is unequivocally at the last row in a dataset. *Eof* is set to *True* when an application

- Opens an empty dataset.
- Calls a dataset's *Last* method.
- Calls a dataset's *Next* method, and the method fails (because the cursor is currently at the last row in the dataset.
- Calls *SetRange* on an empty range or dataset.

Eof is set to *False* in all other cases; you should assume *Eof* is *False* unless one of the conditions above is met *and* you test the property directly. *Eof* is commonly tested in a loop condition to control iterative processing of all records in a dataset. If you open a dataset containing records (or you call *First*) *Eof* is *False*. To iterate through the dataset a record at a time, create a loop that terminates when *Eof* is *True*. Inside the loop, call *Next* for each record in the dataset. *Eof* remains *False* until you call *Next* when the cursor is already on the last record.

Beginning of File (Bof)

When *Bof* is *True*, it indicates that the cursor is unequivocally at the first row in a dataset. *Bof* is set to *True* when an application

- Opens a dataset.
- Calls a dataset's *First* method.
- Calls a dataset's *Prior* method, and the method fails (because the cursor is currently at the first row in the dataset.
- Calls *SetRange* on an empty range or dataset.

Bof is set to *False* in all other cases; you should assume *Bof* is *False* unless one of the conditions above is met *and* you test the property directly. Like *Eof*, *Bof* can be in a loop condition to control iterative processing of records in a dataset.

Using Locate

Locate moves the cursor to the first row matching a specified set of search criteria. In its simplest form, you pass *Locate* the name of a column to search, a field value to match, and an options flag specifying whether the search is case-insensitive or if it can use partial-key matching.

MODIFYING DATA

You can use the following dataset methods to insert, update, and delete data:

Editing Records

A dataset must be in *dsEdit* mode before an application can modify records. In your code you can use the *Edit* method to put a dataset into *dsEdit* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the table(s) underlying a dataset permits read and write privileges. On forms in your application, some data-aware controls can automatically put a dataset into *dsEdit* state if

- The control's *ReadOnly* property is *False* (the default),
- The *AutoEdit* property of the data source for the control is *True*, and
- *CanModify* is *True* for the dataset.

DataSet Actions

The standard dataset actions are designed to be used with a dataset component target. *TDataSetAction* is the base class for descendants that each override the

ExecuteTarget and *UpdateTarget* methods to implement navigation and editing of the target. The *TDataSetAction* introduces a *DataSource* property which ensures actions are performed on that dataset. If *DataSource* is nil, the currently focused data-aware control is used.

- ***TdataSet.Action*** ensures that the target is a *TDataSource* class and has an associated data set.
- ***TdataSet.Cancel*** cancels the edits to the current record, restores the record display to its condition prior to editing, and turns off Insert and Edit states if they are active.
- ***TdataSet.Delete*** deletes the current record and makes the next record the current record.
- ***TdataSet.Edit*** puts the dataset into *Edit* state so that the current record can be modified.
- ***TdataSet.First*** sets the current record to the first record in the dataset.
- *TdataSet.Insert* inserts a new record before the current record, and sets the dataset into Insert and Edit states.
- ***TdataSet.Last*** sets the current record to the last record in the dataset.
- ***TdataSet.Next*** sets the current record to the next record.
- ***TdataSet.Post*** writes changes in the current record to the dataset.
- ***TdataSet.Prior*** sets the current record to the previous record.
- ***TdataSet.Refresh*** refreshes the buffered data in the associated dataset.
- ***TDataSet.Append*** adds a new, empty record to the end of the dataset.

Adding New Records

A dataset must be in *dsInsert* mode before an application can add new records. In code, you can use the *Insert* or *Append* methods to put a dataset into *dsInsert* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, the data-aware grid and navigator controls can put a dataset into *dsInsert* state if

- The control's *ReadOnly* property is *False* (the default), and
- *CanModify* is *True* for the dataset.

Inserting Records

Insert opens a new, empty record before the current record, and makes the empty record the current record so that field values for the record can be entered either by a user or by your application code. When an application calls *Post* (or *ApplyUpdates* when cached updating is enabled), a newly inserted record is written to a database in one of three ways:

- For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.
- For unindexed tables, the record is inserted into the dataset at its current position.
- For SQL databases, the physical location of the insertion is implementation-specific. If the table is indexed, the index is updated with the new record information.

Appending Records

Append opens a new, empty record at the end of the dataset, and makes the empty record the current one so that field values for the record can be entered either by a user or by your application code. When an application calls *Post* (or *ApplyUpdates* when cached updating is enabled), a newly appended record is written to a database in one of three ways:

- For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.
- For unindexed tables, the record is added to the end of the dataset.
- For SQL databases, the physical location of the append is implementation-specific. If the table is indexed, the index is updated with the new record information.

Deleting Records

A dataset must be active before an application can delete records. *Delete* deletes the current record from a dataset and puts the dataset in *dsBrowse* mode. The record that followed the deleted record becomes the current record. If cached updates are enabled for a dataset, a deleted record is only removed from the temporary cache buffer until you call *ApplyUpdates*. If you provide a navigator component on your forms, users can delete the current record by clicking the navigator's Delete button. In code, you must call *Delete* explicitly to remove the current record.

Canceling Changes

An application can undo changes made to the current record at any time, if it has not yet directly or indirectly called *Post*. For example, if a dataset is in *dsEdit* mode, and a user has changed the data in one or more fields, the application can return the record back to its original values by calling the *Cancel* method for the dataset. A call to *Cancel* always returns a dataset to *dsBrowse* state. On forms, you can allow users to cancel edit, insert, or append operations by including the Cancel button on a navigator component associated with the dataset, or you can provide code for your own Cancel button on the form.

Using Dataset Events

Datasets have a number of events that enable an application to perform validation, compute totals, and perform other tasks. The events are listed in the following table. For more information about events for the *TDataSet* component, see the online *VCL Reference*.

Event	Description
BeforeOpen, AfterOpen	Called before/after a dataset is opened.
BeforeClose, AfterClose	Called before/after a dataset is closed.
BeforeInsert, AfterInsert	Called before/after a dataset enters Insert state.
BeforeEdit, AfterEdit	Called before/after a dataset enters Edit state.
BeforePost, AfterPost	Called before/after changes to a table are posted.
BeforeCancel, AfterCancel	Called before/after the previous state is canceled.
BeforeDelete, AfterDelete	Called before/after a record is deleted.
OnNewRecord	Called when a new record is created; used to set default values.
OnCalcFields	Called when calculated fields are calculated.

FARMER INFORMATION DATABASE

The databases were made in a program accompanying the whole Delphi package, and these types of databases are called paradox databases. Two databases are used in this program, which names are Farmer1.db and Farmer3.db. Farmer1.db is used to store farmer information records and Farmer3.db is used to store fixed payment for a decare.

Field Name	Type	Size
City	A	10
Surname	A	10
Tcidentifyno	A	10
Taxidentifyno	A	10
Landno	A	10
Plotoflandno	A	10
Name	A	10
Town	A	15
Village	A	10
Managingno	A	10
Telephone	A	11
Adress	A	50
Fathername	A	10
Bitrhdate	A	10
Bankaccountno	A	15
Personno	A	2
Lessperson	A	2
Moreperson	A	2
Recorddate	A	10
Reportdate	A	10
Landdecare	A	5
Landmetre	A	3
Quality	A	10
Usagetype	A	10
Ownrentingasset	A	10
Locationofvillage	A	11
Solvency	A	5
Paymentamount	A	15
Applicationdate	A	10
Contractdate	A	10
Leasedate	A	10
Certificateno	A	10

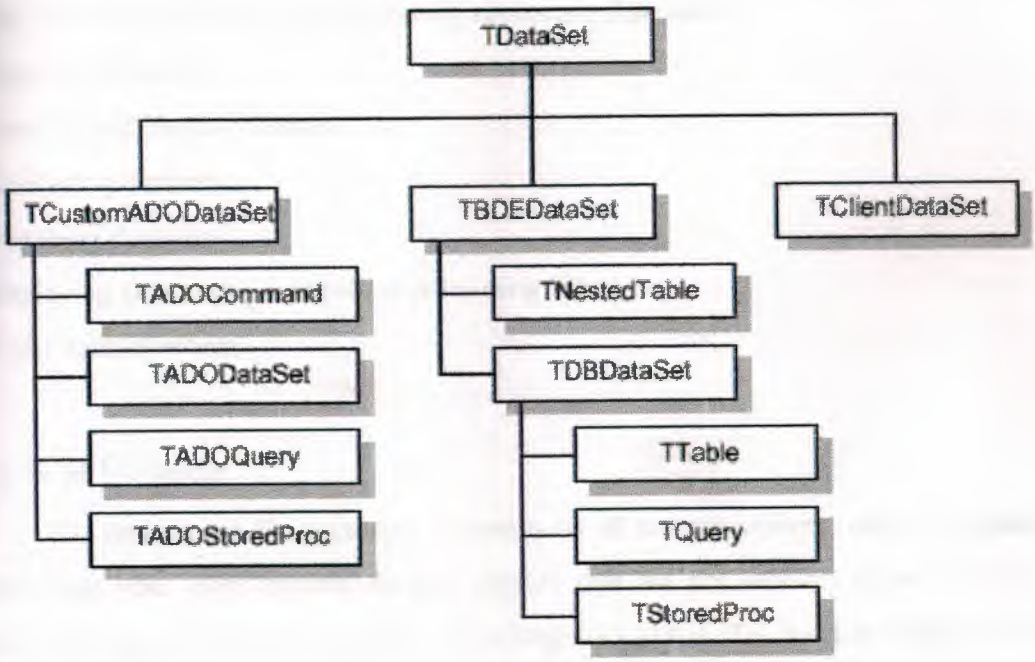
Farmer1.DB

Field Name	Type	Size
Fixedpayment	N	

Farmer3.DB

stored procedure that accesses a database. All dataset objects that you will use in your database applications descend from the virtualized dataset object, *TDataSet*, and they inherit data fields, properties, events, and methods from *TDataSet*. This chapter describes the functionality of *TDataSet* that is inherited by the dataset objects you will use in your database applications. You need to understand this shared functionality to use any dataset object. Figure 1.1 illustrates the hierarchical relationship of all the dataset components:

Figure 1 Delphi Dataset hierarchies



What is TDataSet?

TDataSet is the ancestor for all dataset objects you use in your applications. It defines a set of data fields, properties, events, and methods shared by all dataset objects. *TDataSet* is a virtualized dataset, meaning that many of its properties and methods are virtual or abstract. A *virtual method* is a function or procedure declaration where the implementation of that method can be (and usually is) overridden in descendant objects. An *abstract method* is a function or procedure declaration without an actual implementation. The declaration is a prototype that describes the method (and its parameters and return type, if any) that must be implemented in all descendant dataset

objects, but that might be implemented differently by each of them. Because *TDataSet* contains abstract methods, you cannot use it directly in an application without generating a runtime error. Instead, you either create instances of *TDataSet*'s descendants, such as *TTable*, *TQuery*, *TStoredProc*, and *TClientDataSet*, and use them in your application, or you derive your own dataset object from *TDataSet* or its descendants and write implementations for all its abstract methods. Nevertheless, *TDataSet* defines much that is common to all dataset objects. For example, *TDataSet* defines the basic structure of all datasets: an array of *Tfield* components that correspond to actual columns in one or more database tables, lookup fields provided by your application, or calculated fields provided by your application. For more information about *TField* components, the following topics are discussed:

- Types of datasets
- Opening and closing datasets
- Navigating datasets
- Searching datasets
- Displaying and editing a subset of data using filters
- Using dataset events

Types of Datasets

To understand the concepts common to all dataset objects, and to prepare for developing your own custom dataset objects that do not rely on either the Borland Database Engine (BDE) or ActiveX Data Objects (ADO). To develop traditional, two-tier client/server database applications using the Borland Database Engine (BDE),. That section introduces *TBDEDataSet* and *TDBDataSet*, and focuses on the shared features of *TQuery*, *TStoredProc*, and *TTable*, the dataset components used most commonly in all database applications. With some versions of Delphi, you can develop multi-tier database applications using distributed datasets.

Opening and Closing Datasets

To read or write data in a table or through a query, an application must first open a dataset. You can open a dataset in two ways,

•Set the *Active* property of the dataset to *True*, either at design time in the Object Inspector

• Call the *Open* method for the dataset at runtime,

You can close a dataset in two ways,

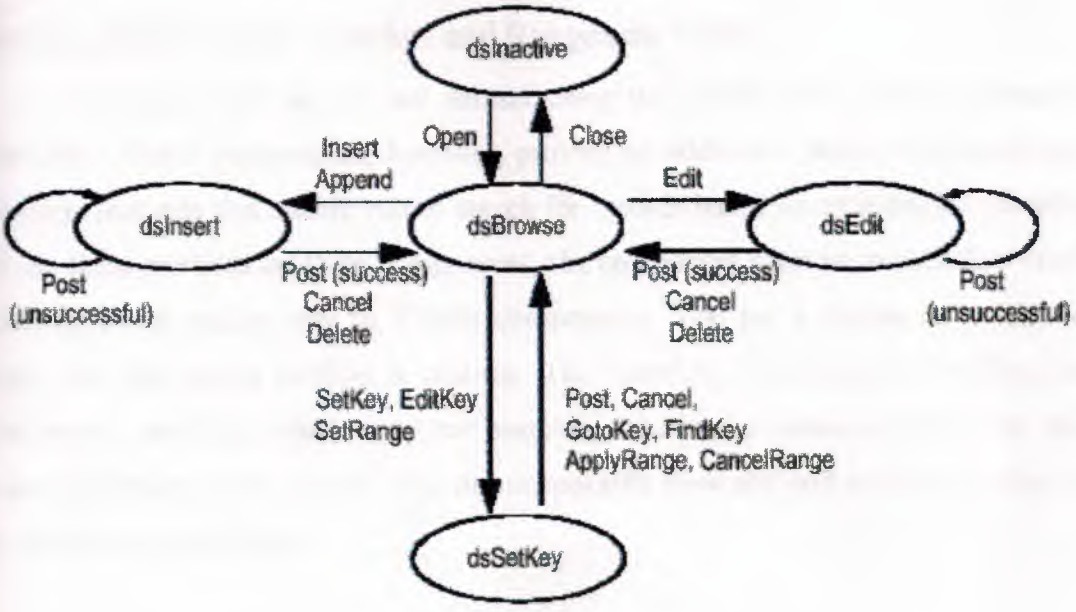
•Set t he *Active* property of the dataset to *Fals*e, either at design time in the Object Inspector,

• Call the *Close* method for the dataset at runtime.

Browsing a Dataset

When an application opens a dataset, the dataset automatically enters *dsBrowse* state. Browsing enables you to view records in a dataset, but you cannot edit records or insert new records. You mainly use *dsBrowse* to scroll from record to record in a dataset. From *dsBrowse* all other dataset states can be set. For example, calling the *Insert* or *Append* methods for a dataset changes its state from *dsBrowse* to *dsInsert* (note that other factors and dataset properties, such as *CanModify*, may prevent this change). Calling *SetKey* to search for records puts a dataset in *dsSetKey* mode. Two methods associated with all datasets can return a dataset to *dsBrowse* state. *Cancel* ends the current edit, insert, or search task, and always returns a dataset to *dsBrowse* state. *Post* attempts to write changes to the database, and if successful, also returns a dataset to *dsBrowse* state. If *Post* fails, the current state remains unchanged. The following diagram illustrates the relationship of *dsBrowse* both to the other dataset modes you can set in your applications, and the methods that set those modes.

Figure 1.2 Relationship of Browse to other dataset states



Enabling Dataset Editing

A dataset must be in *dsEdit* mode before an application can modify records. In your code you can use the *Edit* method to put a dataset into *dsEdit* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, some data-aware controls can automatically put a dataset into *dsEdit* state if:

•The control's *ReadOnly* property is *False* (the default),

•The *AutoEdit* property of the data source for the control is *True*, and

• *CanModify* is *True* for the dataset.

Enabling Insertion of New Records

A dataset must be in *dsInsert* mode before an application can add new records. In your code you can use the *Insert* or *Append* methods to put a dataset into *dsInsert* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, the data-aware grid and navigator controls can put a dataset into *dsInsert* state if

•The control's *ReadOnly* property is *False* (the default),

•The *AutoEdit* property of the data source for the control is *True*, and

• *CanModify* is *True* for the dataset.

Enabling Index-Based Searches and Ranges on Tables

You can search against any dataset using the *Locate* and *Lookup* methods of *TDataSet*. *TTable* components, however, provide an additional family of *GotoKey* and *FindKey* methods that enable you to search for records based on an index for the table. To use these methods on table components, the component must be in *dsSetKey* mode. *dsSetKey* mode applies only to *TTable* components. You put a dataset into *dsSetKey* mode with the *SetKey* method at runtime. The *GotoKey*, *GotoNearest*, *FindKey*, and *FindNearest* methods, which carry out searches, returns the dataset to *dsBrowse* state upon completion of the search. You can temporarily view and edit a subset of data for any dataset by using filters.

Filtering Records

Helphi puts a dataset into *dsFilter* mode whenever an application calls the dataset's *OnFilterRecord* event handler. This state prevents modifications or additions to the records in a dataset during the filtering process so that the filter request is not invalidated. When the *OnFilterRecord* handler finishes, the dataset is returned to *dsBrowse* state.

Updating Records

When performing cached update operations, Delphi may put the dataset into *dsNewValue*, *dsOldValue*, or *dsCurValue* states temporarily. These states indicate that the corresponding properties of a field component *(NewValue*, *OldValue*, and *CurValue*, respectively) are being accessed, usually in an *OnUpdateError* event handler. Your applications cannot see or set these states.

Using the Eof and Bof Properties

Two read-only, runtime properties, *Eof* (End-of-file) and *Bof* (Beginning-of-file), are useful for controlling dataset navigation, particularly when you want to iterate through all records in a dataset.

End of File (Eof)

When *Eof* is *True*, it indicates that the cursor is unequivocally at the last row in a dataset. *Eof* is set to *True* when an application

- Opens an empty dataset.
- Calls a dataset's *Last* method.
- Calls a dataset's *Next* method, and the method fails (because the cursor is currently at the last row in the dataset.
- Calls *SetRange* on an empty range or dataset.

Eof is set to *False* in all other cases; you should assume *Eof* is *False* unless one of the conditions above is met *and* you test the property directly. *Eof* is commonly tested in a loop condition to control iterative processing of all records in a dataset. If you open a dataset containing records (or you call *First*) *Eof* is *False*. To iterate through the dataset a record at a time, create a loop that terminates when *Eof* is *True*. Inside the loop, call *Next* for each record in the dataset. *Eof* remains *False* until you call *Next* when the cursor is already on the last record.

Beginning of File (Bof)

When *Bof* is *True*, it indicates that the cursor is unequivocally at the first row in a dataset. *Bof* is set to *True* when an application

- Opens a dataset.
- Calls a dataset's *First* method.
- Calls a dataset's *Prior* method, and the method fails (because the cursor is currently at the first row in the dataset.
- Calls *SetRange* on an empty range or dataset.

Bof is set to *False* in all other cases; you should assume *Bof* is *False* unless one of the conditions above is met *and* you test the property directly. Like *Eof*, *Bof* can be in a loop condition to control iterative processing of records in a dataset.

Using Locate

Locate moves the cursor to the first row matching a specified set of search criteria. In its simplest form, you pass *Locate* the name of a column to search, a field value to match, and an options flag specifying whether the search is case-insensitive or if it can use partial-key matching.

MODIFYING DATA

You can use the following dataset methods to insert, update, and delete data:

Editing Records

A dataset must be in *dsEdit* mode before an application can modify records. In your code you can use the *Edit* method to put a dataset into *dsEdit* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the table(s) underlying a dataset permits read and write privileges. On forms in your application, some data-aware controls can automatically put a dataset into *dsEdit* state if

- The control's *ReadOnly* property is *False* (the default),
- The *AutoEdit* property of the data source for the control is *True*, and
- *CanModify* is *True* for the dataset.

DataSet Actions

The standard dataset actions are designed to be used with a dataset component target. *TDataSetAction* is the base class for descendants that each override the

ExecuteTarget and *UpdateTarget* methods to implement navigation and editing of the target. The *TDataSetAction* introduces a *DataSource* property which ensures actions are performed on that dataset. If *DataSource* is nil, the currently focused data-aware control is used.

- ***TdataSet.Action*** ensures that the target is a *TDataSource* class and has an associated data set.
- ***TdataSet.Cancel*** cancels the edits to the current record, restores the record display to its condition prior to editing, and turns off Insert and Edit states if they are active.
- ***TdataSet.Delete*** deletes the current record and makes the next record the current record.
- ***TdataSet.Edit*** puts the dataset into *Edit* state so that the current record can be modified.
- ***TdataSet.First*** sets the current record to the first record in the dataset.
- *TdataSet.Insert* inserts a new record before the current record, and sets the dataset into Insert and Edit states.
- ***TdataSet.Last*** sets the current record to the last record in the dataset.
- ***TdataSet.Next*** sets the current record to the next record.
- ***TdataSet.Post*** writes changes in the current record to the dataset.
- ***TdataSet.Prior*** sets the current record to the previous record.
- ***TdataSet.Refresh*** refreshes the buffered data in the associated dataset.
- ***TDataSet.Append*** adds a new, empty record to the end of the dataset.

Adding New Records

A dataset must be in *dsInsert* mode before an application can add new records. In code, you can use the *Insert* or *Append* methods to put a dataset into *dsInsert* mode if the read-only *CanModify* property for the dataset is *True*. *CanModify* is *True* if the database underlying a dataset permits read and write privileges. On forms in your application, the data-aware grid and navigator controls can put a dataset into *dsInsert* state if

- The control's *ReadOnly* property is *False* (the default), and
- *CanModify* is *True* for the dataset.

Inserting Records

Insert opens a new, empty record before the current record, and makes the empty record the current record so that field values for the record can be entered either by a user or by your application code. When an application calls *Post* (or *ApplyUpdates* when cached updating is enabled), a newly inserted record is written to a database in one of three ways:

- For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.
- For unindexed tables, the record is inserted into the dataset at its current position.
- For SQL databases, the physical location of the insertion is implementation-specific.

If the table is indexed, the index is updated with the new record information.

Appending Records

Append opens a new, empty record at the end of the dataset, and makes the empty record the current one so that field values for the record can be entered either by a user or by your application code. When an application calls *Post* (or *ApplyUpdates* when cached updating is enabled), a newly appended record is written to a database in one of three ways:

- For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.
- For unindexed tables, the record is added to the end of the dataset.
- For SQL databases, the physical location of the append is implementation-specific.

If the table is indexed, the index is updated with the new record information.

Deleting Records

A dataset must be active before an application can delete records. *Delete* deletes the current record from a dataset and puts the dataset in *dsBrowse* mode. The record that followed the deleted record becomes the current record. If cached updates are enabled for a dataset, a deleted record is only removed from the temporary cache buffer until you call *ApplyUpdates*. If you provide a navigator component on your forms, users can delete the current record by clicking the navigator's Delete button. In code, you must call *Delete* explicitly to remove the current record.

Canceling Changes

An application can undo changes made to the current record at any time, if it has not yet directly or indirectly called *Post*. For example, if a dataset is in *dsEdit* mode, and a user has changed the data in one or more fields, the application can return the record back to its original values by calling the *Cancel* method for the dataset. A call to *Cancel* always returns a dataset to *dsBrowse* state. On forms, you can allow users to cancel edit, insert, or append operations by including the Cancel button on a navigator component associated with the dataset, or you can provide code for your own Cancel button on the form.

Using Dataset Events

Datasets have a number of events that enable an application to perform validation, compute totals, and perform other tasks. The events are listed in the following table. For more information about events for the *TDataSet* component, see the online *VCL Reference*.

Event	Description
BeforeOpen, AfterOpen	Called before/after a dataset is opened.
BeforeClose, AfterClose	Called before/after a dataset is closed.
BeforeInsert, AfterInsert	Called before/after a dataset enters Insert state.
BeforeEdit, AfterEdit	Called before/after a dataset enters Edit state.
BeforePost, AfterPost	Called before/after changes to a table are posted.
BeforeCancel, AfterCancel	Called before/after the previous state is canceled.
BeforeDelete, AfterDelete	Called before/after a record is deleted.
OnNewRecord	Called when a new record is created; used to set default values.
OnCalcFields	Called when calculated fields are calculated.

FARMER INFORMATION DATABASE

The databases were made in a program accompanying the whole Delphi package, and these types of databases are called paradox databases. Two databases are used in this program, which names are Farmer1.db and Farmer3.db. Farmer1.db is used to store farmer information records and Farmer3.db is used to store fixed payment for a decare.

Field Name	Type	Size
City	A	10
Surname	A	10
Tcidentifyno	A	10
Taxidentifyno	A	10
Landno	A	10
Plotoflandno	A	10
Name	A	10
Town	A	15
Village	A	10
Managingno	A	10
Telephone	A	11
Adress	A	50
Fathername	A	10
Bitrhdate	A	10
Bankaccountno	A	15
Personno	A	2
Lessperson	A	2
Moreperson	A	2
Recorddate	A	10
Reportdate	A	10
Landdecare	A	5
Landmetre	A	3
Quality	A	10
Usagetype	A	10
Ownrentingasset	A	10
Locationofvillage	A	11
Solvency	A	5
Paymentamount	A	15
Applicationdate	A	10
Contractdate	A	10
Leasedate	A	10
Certificateno	A	10

Farmer1.DB

Field Name	Type	Size
Fixedpayment	N	

Farmer3.DB

FARMER INFORMATION SYSTEM

This program is prepared for manipulating the records of farmers and their lands. The main purpose is to use a computer and automate the payments, taxing and other processes which will reduce the risk of being confused. The advantages of automation provide the time saving and protect the government from a crime that is done by giving wrong information.

MAIN MENU

The main window consists of a searching menu. The options or keys which can be used to search for a specific data set are; City, Land No., Plot of Land No. After information is entered to these edits, Find Button is clicked. Clicking the find button locates the dataset specified and if database consists of this record, his name and surnames are written to edits, if not a message is given which is "This Record not found". Clicking the clear button clears the fields. There is also a panel provided in the main window, which serves as the launch pad for various sub programs built into the main window. These are Enter New Record; Search Record, Delete Record, Update Record, List Record, About and Exit. There are also array drops down menus that serve specific purposes. These are Record (Enter New Record); Search (Search Record), Delete (Delete Record), Update (Update Record), List (List Record), About and Exit. This form is shown below

Program Codes;

```
Unit Unit1;
Interface
Uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, Db, DBTables, StdCtrls, TeEngine, Series, ExtCtrls, TeeProcs,
  Chart, jpeg, Mask, DBCtrls, Buttons;
Type
  TForm1 = class (TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
```

```
    procedure SearchRecord1Click(Sender: TObject);
    procedure DelteRecord1Click(Sender: TObject);
    procedure SortREcord1Click(Sender: TObject);
    procedure ListRecord1Click(Sender: TObject);
    procedure EnterNewRocord1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormHide(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure Exit2Click(Sender: TObject);
  private
    { Private declarations }
    Bitmap1, Bitmap2, Bitmap3: TBitmap;
    Image1Loaded, Image2Loaded: Boolean;
  public
  Cut : Boolean;
    { Public declarations }
  end;
var
  Form1: TForm1;
  cut:boolean;
implementation
uses Unit2,Unit4,Unit6, Unit5, Unit7, Unit3;
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
   bitmap1 := Graphics.TBitmap.Create;
```

```
  bitmap2 := Graphics.TBitmap.Create;
  bitmap3 := Graphics.TBitmap.Create;
  bitmap1.PixelFormat := pf8bit;
  bitmap2.PixelFormat := pf8bit;
  bitmap3.PixelFormat := pf8bit;
try
    bitmap1.LoadFromFile('d:\farmer\factory.bmp');
    bitmap2.LoadFromFile('d:\farmer\handshak.bmp');
    Image1Loaded := true;
    Image2Loaded := true;
    bitmap3.Palette := bitmap1.Palette;
    bitmap3.Height := bitmap1.Height;
    bitmap3.Width := bitmap1.Width;
  except
    Image1Loaded := false;
    Image2Loaded := false;
  end;
end;
procedure TForm1.FormActivate(Sender: TObject);
var
 Next: PByteArray;
 ToDisplay: PByteArray;
 i, j,  y: Integer;
begin
if Cut = False Then
begin
 if not (Image1Loaded) then
   ShowMessage('Bitmap1 not loaded');
 if not (Image2Loaded) then
   ShowMessage('Bitmap2 not loaded');
if((Image1Loaded) and (Image2Loaded)) then begin
    bitmap3.Canvas.CopyRect(Rect(0,0,bitmap3.Width,bitmap3.Height),
                  bitmap1.Canvas,Rect(0,0,bitmap1.Width,
                  bitmap1.Height));
```

```
    for i := 0 to bitmap1.Width do begin
      for y := 0 to bitmap1.Height - 1 do begin
        Next := bitmap2.ScanLine[y];
        ToDisplay := bitmap3.ScanLine[y];
        if(i < bitmap1.Width - 15) then begin
          for j := 1 to 14 do
            if((y = 0) or (y = bitmap1.Height - 1)) then
              ToDisplay[i+j] := clBlack   else
              ToDisplay[i+j] := Next[i+10-j];
        end;
        ToDisplay[i] := Next[i];
      end;
      Sleep(2);
      Image1.Canvas.Draw(0,0,bitmap3);
      Application.ProcessMessages();
    end;;
  end;;
end;
Cut:=True;
end;
procedure TForm1.Exit1Click(Sender: TObject);
begin
form3.show;
end;
procedure TForm1.SearchRecord1Click(Sender: TObject);
begin
form4.table1.refresh;
form4.show;
form1.Hide;
end;
procedure TForm1.DelteRecord1Click(Sender: TObject);
begin
form6.table1.refresh;
form6.show;
```

```
form1.Hide;
end;
procedure TForm1.SortREcord1Click(Sender: TObject);
begin
form5.table1.refresh;
form5.show;
form1.Hide;
end;
procedure TForm1.ListRecord1Click(Sender: TObject);
begin
form7.Table1.Refresh;
form7.Show;
form1.Hide;
end;
procedure TForm1.EnterNewRocord1Click(Sender: TObject);
begin
form2.show;
form1.hide;
form2.CleanAll;
form2.BitBtn2.Enabled:=False;
form2.Bitbtn3.Enabled:=False;
form2.BitBtn1.Enabled:=True;
form2.Bitbtn4.Enabled:=True;
end;
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
form2.show;
form1.hide;
form2.CleanAll;
form2.BitBtn2.Enabled:=False;
form2.Bitbtn3.Enabled:=False;
form2.BitBtn1.Enabled:=True;
form2.Bitbtn4.Enabled:=True;
end;
```

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
form4.table1.refresh;
form4.show;
form1.Hide;
end;
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
form6.table1.refresh;
form6.show;
form1.Hide;
end;
procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
form7.Table1.Refresh;
form7.Show;
form1.Hide;
end;
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
Halt;
end;
procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
form5.table1.refresh;
form5.show;
form1.Hide;
end;
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
var
i : integer;
begin
if (key in ['0'..'9']) or (Key=#13) or (key=#8) then
begin
```

```
  for i:=0 to ComponentCount-1 do
  begin
    if Components[i] is TEdit then
    with Components[i] as TEdit do
    begin
    if Tag=0 then ReadOnly := False;
    end;
  end;
end else
  for i:=0 to ComponentCount-1 do
  begin
    if Components[i] is TEdit then
    with Components[i] as TEdit do
    begin
    if Tag=0 then ReadOnly := True;
    end;
  end;
end;
procedure TForm1.FormHide(Sender: TObject);
begin
Edit1.text:='';
Edit2.text:='';
Edit3.text:='';
table1.Close;
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
var
ended,found:boolean;
begin
Table1.open;
found:=false;
ended:=false;
table1.First;
while (not found) do
```

```
begin
if (edit1.text=table1city.Text) and (edit2.text=table1landno.Text)and
(edit3.text=table1parselno.Text) then
   begin   found:=true;   end   else
   begin
      table1.Next;
      if table1.Eof=true then
         begin
         found:=true;
         ended:=true;
         end;
   end;
end;

if (not found) or (not ended) then
else
begin
beep;
messagedlg('This record is not found.  ',mtinformation,[mbok],0);
table1.close;
end;
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
edit1.text:='';
edit2.text:='';
edit3.text:='';
table1.close;
end;
procedure TForm1.SpeedButton7Click(Sender: TObject);
begin
form3.show;
end;
procedure TForm1.Exit2Click(Sender: TObject);
```

```
begin
Halt;
cut:=True;
end;
end.
```

RECORD PART

This is an important unit, which consists of farmer information. Farmer information records data have Residence Information, Farmer's Personal Information, and Land Information. This field has been incorporated into the program as to record the personal data and the data pertaining to the farmer's assets. It is worth mentioning here that there are 3 specific errors designed in the system.

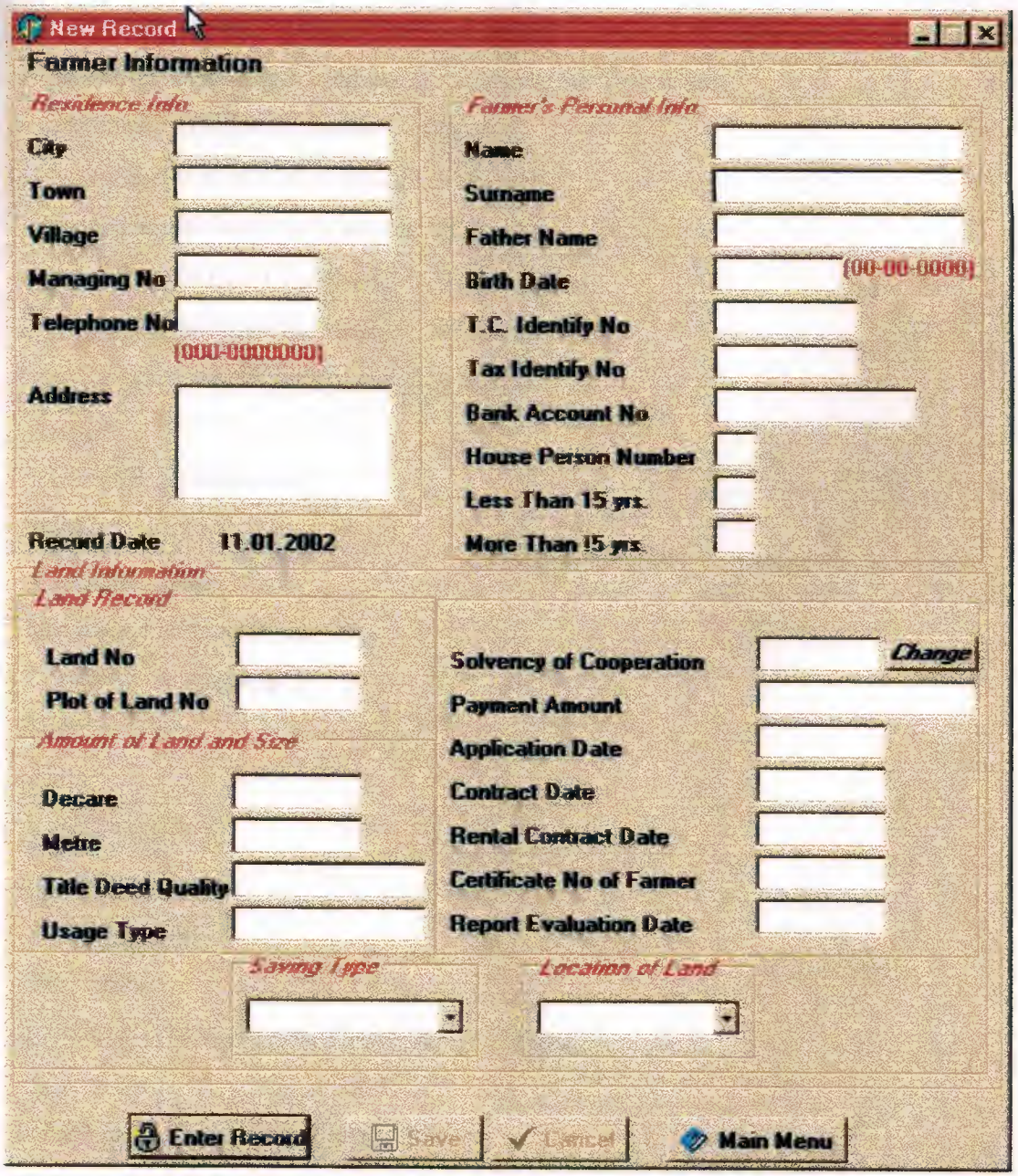
First of all, no two can be alike in the same Identification Number. If it is written same identification number, an error is given which is " Somebody has this Identification Number, please check again this number".

Second of all, the Plot No. and Plot of land no. are unique and no two can be alike in the same city, as this means the existence of the same property under the ownership of two different people. This is impossible. If the Plot No. exists and Plot of land no. are written, an error is given which is " This record is written before.".

Third of all, the field Solvency of corporative, accepts a fixed amount, and gives the respective payment amount. The payment amount can be modified for a decare by pressing the Change button, which is shown in below;

There are also 2 drop down menus, Saving and Location of Land. The farmer depicts the type of ownership (rental or owned) and the farmer shows, the land of place. There are four buttons provided in this form, which are Enter Record, Save, Cancel, Main menu. The Enter record button, when pressed diminishes the main menu button and allows the user to enter data sets. The cancel button nullifies the data. And the main menu, returns the unit to the main menu. The save button saves the data and is also used

to find who has the same land number and plot of land number in the same city. If there is a record according to these fields, a message is given which is "Somebody has these Land and Plot of Land Number". Then, there are two buttons, which are Cancel and Retry. Cancel button is used to clean the editboxes and if wrong number is entered, Retry Button is used to make correct the mistake for this reason, if it is necessary. A figure is shown for new record which is below;



Program Codes;

unit Unit2;

Interface

```
Uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  StdCtrls,
ComCtrls, Tabnotbk, Mask, Db, DBTables, ExtCtrls, Buttons;
Type
  TForm2 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure Edit19Exit(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Edit11Exit(Sender: TObject);
  private
    { Private declarations }
  public
  Procedure CleanAll;
  Procedure RecordTable;
    { Public declarations }
  end;
var
  Form2: TForm2;
implementation
uses Unit1, Unit11;
{$R *.DFM}
procedure TForm2.FormCreate(Sender: TObject);
begin
label84.caption:=datetostr(date);
end;
procedure TForm2.CleanAll;
var i : integer;
begin
  for i:=0 to ComponentCount-1 do
```

```
 begin
   if Components[i] is TEdit then
   with Components[i] as TEdit do
   begin
   if Tag=0 then text := '';
   end;
  end;
memo1.text:='';
Groupbox1.Enabled:=False;
Groupbox2.Enabled:=False;
Groupbox4.Enabled:=False;
end;
procedure TForm2.RecordTable;
begin
table1.append;
table1city.text:=edit1.text;
table1town.text:=edit2.text;
table1village.text:=edit3.text;
table1Managingno.text:=edit4.text;
table1telephone.text:=edit10.text;
table1address.text:=memo1.Text;
table1name.text:=edit7.text;
table1surname.text:=edit8.text;
table1fathername.text:=edit9.text;
table1birthdate.text:=edit79.text;
table1tcidentifyno.text:=edit11.text;
table1taxidentifyno.text:=edit12.text;
table1bankaccountno.text:=edit13.text;
table1personno.text:=edit14.text;
table1lessperson.text:=edit15.text;
table1moreperson.text:=edit16.text;
table1landno.text:=edit17.text;
table1parselno.text:=edit18.text;
table1landdecare.text:=edit20.text;
```

```
table1landmetre.text:=edit21.text;
table1quality.text:=edit22.text;
table1usage.text:=edit23.text;
table1solvency.text:=edit19.text;
table1paymentamount.text:=edit24.text;
table1appdate.text:=edit80.text;
table1contractdate.text:=edit81.Text;
table1leasedate.text:=edit82.Text;
table1certificateno.text:=edit25.Text;
table1reportdate.text:=edit83.Text;
table1recorddate.Value:=label84.Caption;
if combobox2.ItemIndex=1 then
begin
table1ownrentingasset.text:='Rented';
end
else
begin
table1ownrentingasset.text:='Own Asset';
end;
if combobox1.ItemIndex=1 then
begin
table1villageposition.Text:='Out Village';
end
else
begin
table1villageposition.Text:='In Village';
end;
table1.refresh;
end;
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
CleanAll;
BitBtn2.Enabled:=False;
Bitbtn3.Enabled:=False;
```

```
BitBtn1.Enabled:=True;
Bitbtn4.Enabled:=True;
end;
procedure TForm2.FormKeyPress(Sender: TObject; var Key: Char);
var
i : integer;
begin
if (key in ['0'..'9']) or (key=#8) or (key='-') then
begin
  for i:=0 to ComponentCount-1 do
  begin
    if Components[i] is TEdit then
    with Components[i] as TEdit do
    begin
    if Tag=0 then ReadOnly := False;
    end;
  end;
end else
  for i:=0 to ComponentCount-1 do
  begin
    if Components[i] is TEdit then
    with Components[i] as TEdit do
    begin
    if Tag=0 then ReadOnly := True;
    end;
  end;
end;
procedure TForm2.Edit19Exit(Sender: TObject);
var
check1,check2,check3:variant;
result:Currency;
begin
table3.Refresh;
if edit19.text<>'' then
```

```
begin
if edit21.text='' then
check1:=(strtoint(Edit20.text)) else
check1:=(strtoint(Edit20.text))+(strtoint(Edit21.text)/1000);
check2:=strtoint(Edit19.text);
if check1<check2 then
  begin
  messagedlg('Amount of land is less then solvency of
cooperative',mtWarning,[mbok],0);
  Edit19.text:='';
  Edit19.setfocus;
  end else
begin
check3:=strtoint(table3fixedpayment.text);
result:=check2*check3;
edit24.text:=formatCurr('#,###',Result)+'TL';
Edit80.setfocus;
end;
end;
end;
procedure TForm2.Button3Click(Sender: TObject);
begin
form11.show;
form11.refresh;
end;
procedure TForm2.BitBtn3Click(Sender: TObject);
var
found,ended:boolean;
Button:integer;
begin
found:=false;
ended:=false;
table1.First;
if messagedlg('Save this record?',mtwarning,[mbyes,mbno],0)=mryes then
```

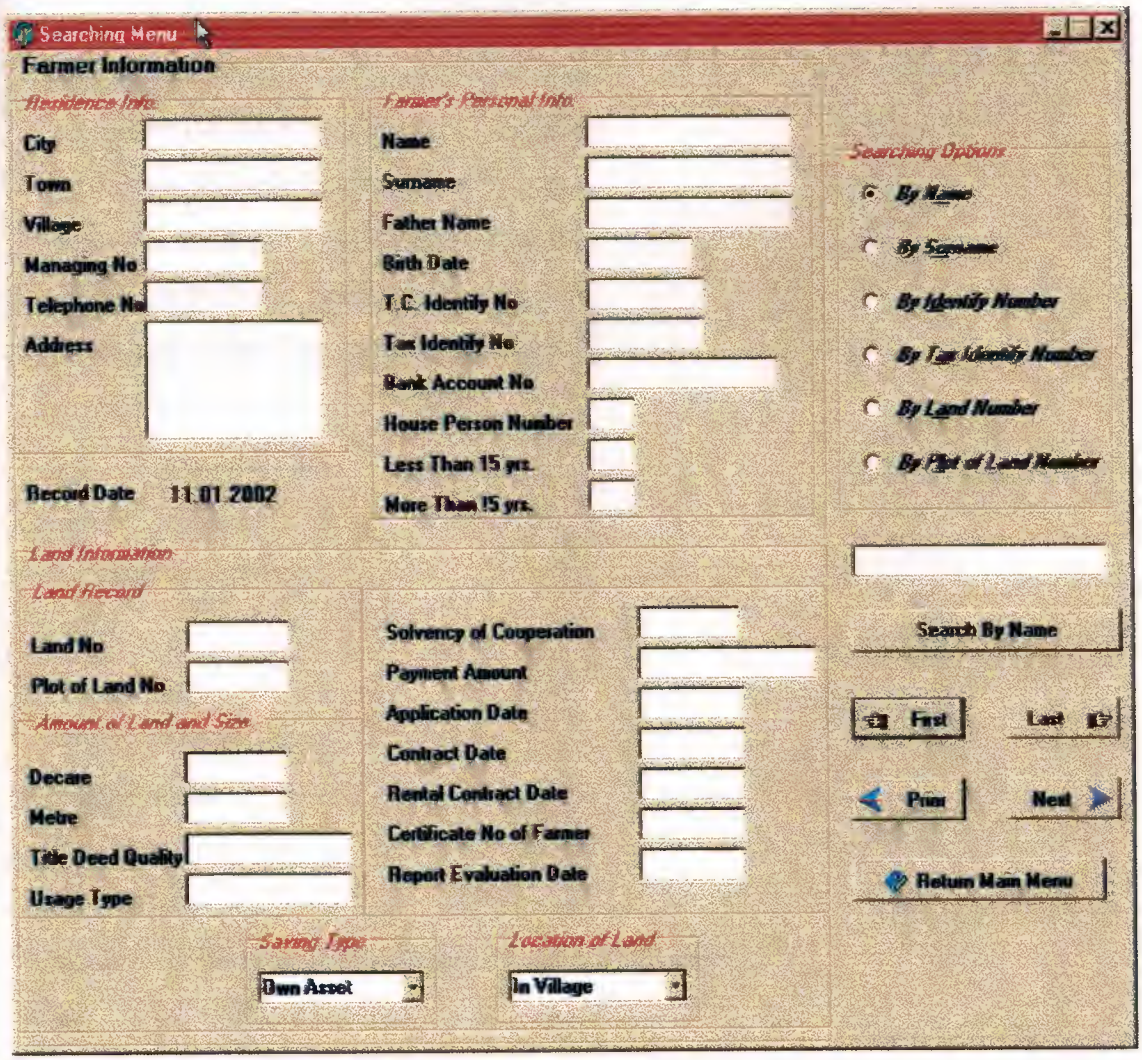
```
begin
while (not found) do
begin
if (edit1.text=table1city.Text) and (edit17.text=table1landno.Text)and
(edit18.text=table1parselno.Text) or (edit11.text=table1tcidentifyno.text) then
    begin
    found:=true;
    end
    else
    begin
      table1.Next;
      if table1.Eof=true then
      begin
      found:=true; ended:=true;
      end;
   end;
end;
if (not found) or (not ended) then
begin
Button:=messagedlg('Somebody has these land and plot of land
number!',mtWarning,[mbcancel,mbRetry],0);
  if Button=2 Then
  begin
    CleanAll;
    BitBtn2.Enabled:=False;
    Bitbtn3.Enabled:=False;
    BitBtn1.Enabled:=True;
    Bitbtn4.Enabled:=True;
  end;
end
else
RecordTable;
end;
end;
```

```
procedure TForm2.BitBtn1Click(Sender: TObject);
begin
Groupbox1.Enabled:=true;
Groupbox2.Enabled:=true;
Groupbox4.Enabled:=true;
BitBtn2.Enabled:=True;
Bitbtn3.Enabled:=True;
BitBtn1.Enabled:=False;
Bitbtn4.Enabled:=False;
edit1.SetFocus;
end;
procedure TForm2.BitBtn4Click(Sender: TObject);
begin
Form2.Close;
form1.show;
Groupbox1.Enabled:=False;
Groupbox2.Enabled:=False;
Groupbox4.Enabled:=False;
end;
procedure TForm2.Edit11Exit(Sender: TObject);
begin
if (table1.Locate('tcidentifyno',edit11.text,[]))=true then
begin
Showmessage('Somebody has this identification number, please check again this
number');
edit11.text:='';
edit11.setfocus;
end;
end;
end.
```

SEARCHING PART

There is a whole range of searching options are available, such as by name, by surname, by identify number, by tax identify number, by land number and by plot of land number. After one of searching options is clicked, which name is written to instead of the search button's name. Then this button is used to search the record in the database. If it consists of this record, farmer informations are shown in the edit boxes. If it does not consists of a message is given which is " This record not found".

There are also buttons, which can show first, last, prior, and next data sets available to the user from the table. They are linked to the specific database. First button sets the current record to the first record in the dataset. Last button sets the current record to the last record in the dataset. Next button sets the current record to the next record. Prior button sets the current record to the previous record. Whenever this menu is opened, fields' enabled is false. Because of this, the user cannot change the records over the form. This form is shown below;



Program Codes;

```
unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  ExtCtrls,
Db, DBTables, StdCtrls, Mask, ComCtrls, DBCtrls, DBActns,  ActnList, Buttons, jpeg;
type
  TForm4 = class(TForm)
    procedure RadioButton5Click(Sender: TObject);
    procedure RadioButton6Click(Sender: TObject);
    procedure RadioButton7Click(Sender: TObject);
    procedure RadioButton8Click(Sender: TObject);
    procedure RadioButton9Click(Sender: TObject);
    procedure RadioButton10Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form4: TForm4;
implementation
uses Unit1;
{$R *.DFM}
procedure TForm4.RadioButton5Click(Sender: TObject);
begin
button4.Caption:='Search By Name';
```

```
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.RadioButton6Click(Sender: TObject);
begin
button4.Caption:='Search By Name';
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.RadioButton7Click(Sender: TObject);
begin
button4.Caption:='Search By Identify Number';
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.RadioButton8Click(Sender: TObject);
begin
button4.Caption:='Search By Tax Identify Number';
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.RadioButton9Click(Sender: TObject);
begin
button4.Caption:='Search By Land Number';
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.RadioButton10Click(Sender: TObject);
begin
button4.Caption:='Search By Plot of Land Number';
edit1.text:='';
edit1.SetFocus;
end;
procedure TForm4.Button4Click(Sender: TObject);
```

```
begin
if radiobutton5.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Farmer Name');
edit1.SetFocus;
end else
if (table1.Locate('name',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.SetFocus;
 table1.cancel;
 end;
end else
if radiobutton6.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Farmer Surname');
edit1.SetFocus;
end else
if (table1.Locate('surname',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.SetFocus;
  table1.cancel;
  end;
 end else
 if radiobutton7.Checked=true then
 begin
 if edit1.text='' then
```

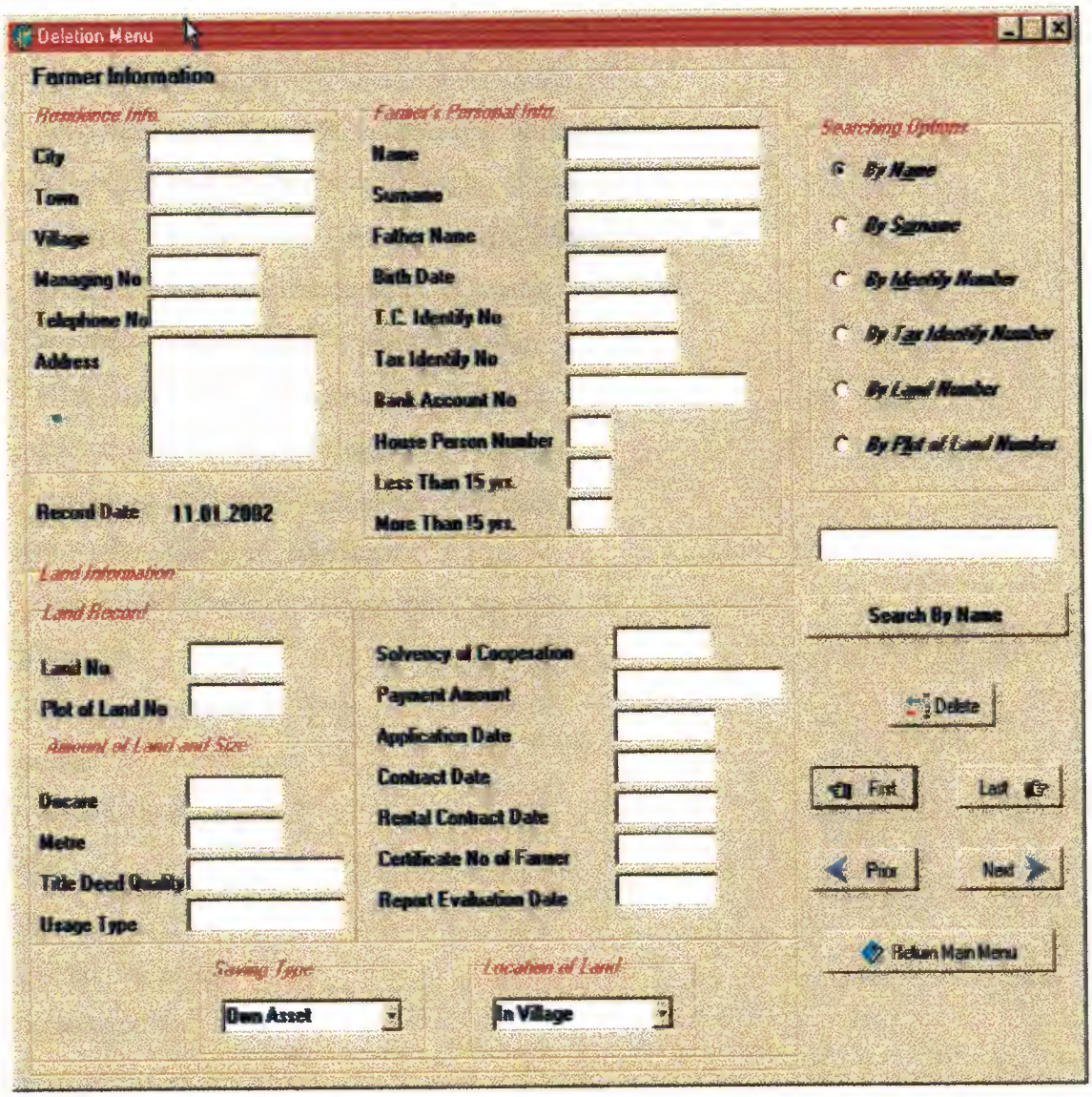
```
begin
showmessage('Please Enter Farmer T.C. Identify Number');
edit1.SetFocus;
end else
if (table1.Locate('tcidentifyno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.SetFocus;
 table1.cancel;
 end;
end else
if radiobutton8.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Tax Identify Number');
edit1.SetFocus;
end else
if (table1.Locate('taxidentifyno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.SetFocus;
 table1.cancel;
 end;
end else
if radiobutton9.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Land Number');
edit1.SetFocus;
end else
```

```
if (table1.Locate('landno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.SetFocus;
 table1.cancel;
 end;
end else
if radiobutton10.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Plot of Land Number');
edit1.SetFocus;
end else
if (table1.Locate('parselno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.SetFocus;
 table1.cancel;
 end;
end else
showmessage(' Choice one of Options');
edit1.text:='';
end;
procedure TForm4.BitBtn1Click(Sender: TObject);
begin
table1.refresh;
form4.close;
form1.show;
end;
procedure TForm4.BitBtn2Click(Sender: TObject);
begin
```

```
table1.Prior;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else
if Table1.Bof=true then
showmessage('Beginning of the file encountered');
end;
procedure TForm4.BitBtn3Click(Sender: TObject);
begin
table1.next;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else
if Table1.eof=true then
showmessage('Ending of the file encountered');
end;
procedure TForm4.BitBtn4Click(Sender: TObject);
begin
table1.First;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm4.BitBtn5Click(Sender: TObject);
begin
table1.Last;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm4.FormActivate(Sender: TObject);
begin
Edit1.Text:='';
Edit1.SetFocus;
radiobutton5.Checked:=True;
end;
```

DELETION PART

Searching options are also used to delete the record in this menu, as it is in the searching menu. There is a button, which is Delete Record. It deletes the current record from the database. Before the record is deleted, a message is given to user which is "Delete this record?". There are two alternatives which are Yes and No. If user presses to yes, record is removed from the database and to No, process is cancelled. First, last, prior, and next buttons are also used to set available record to the user from the table. This menu of fields' enabled is also false, whenever this menu is opened. This form is shown below;



Program Codes;

```
unit Unit6;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  StdCtrls,
DBCtrls, Mask, ComCtrls, Db, DBTables, Buttons, jpeg, ExtCtrls;
type
  TForm6 = class(TForm)
    procedure Button4Click(Sender: TObject);
    procedure RadioButton5Click(Sender: TObject);
    procedure RadioButton6Click(Sender: TObject);
    procedure RadioButton7Click(Sender: TObject);
    procedure RadioButton8Click(Sender: TObject);
    procedure RadioButton9Click(Sender: TObject);
    procedure RadioButton10Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form6: TForm6;
implementation
uses Unit1, Unit4;
{$R *.DFM}
procedure TForm6.Button4Click(Sender: TObject);
begin
```

```
if radiobutton5.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Farmer Name');
edit1.setfocus;
end else
if (table1.Locate('name',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.setfocus;
 table1.cancel;
 end;
end else
if radiobutton6.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Farmer Surname');
edit1.setfocus;
end else
if (table1.Locate('surname',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.setfocus;
 table1.cancel;
 end;
end else
if radiobutton7.Checked=true then
begin
if edit1.text='' then
begin
```

```
showmessage('Please Enter Farmer T.C. Identify Number');
edit1.setfocus;
end else
if (table1.Locate('tcidentifyno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.setfocus;
 table1.cancel;
 end;
end else
if radiobutton8.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Tax Identify Number');
edit1.setfocus;
end else
if (table1.Locate('taxidentifyno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.setfocus;
 table1.cancel;
 end;
end else if radiobutton9.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Land Number');
edit1.setfocus;
end else
if (table1.Locate('landno',edit1.text,[]))=False then
 begin
```

NEAR EAST UNIVERSITY LIBRARY 1988 - LEFKOŞA

```
showmessage('This Record Not Found');
edit1.Text:='';
edit1.setfocus;
table1.cancel;
end;
end else
if radiobutton10.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Plot of Land Number');
edit1.setfocus;
end else
if (table1.Locate('parselno',edit1.text,[]))=False then
 begin
 showmessage('This Record Not Found');
 edit1.Text:='';
 edit1.setfocus;
 table1.cancel;
 end;
end else
begin
showmessage(' Choice one of options');
edit1.text:='';
end;
end;
procedure TForm6.RadioButton5Click(Sender: TObject);
begin
button4.Caption:='Search By Name';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.RadioButton6Click(Sender: TObject);
begin
```

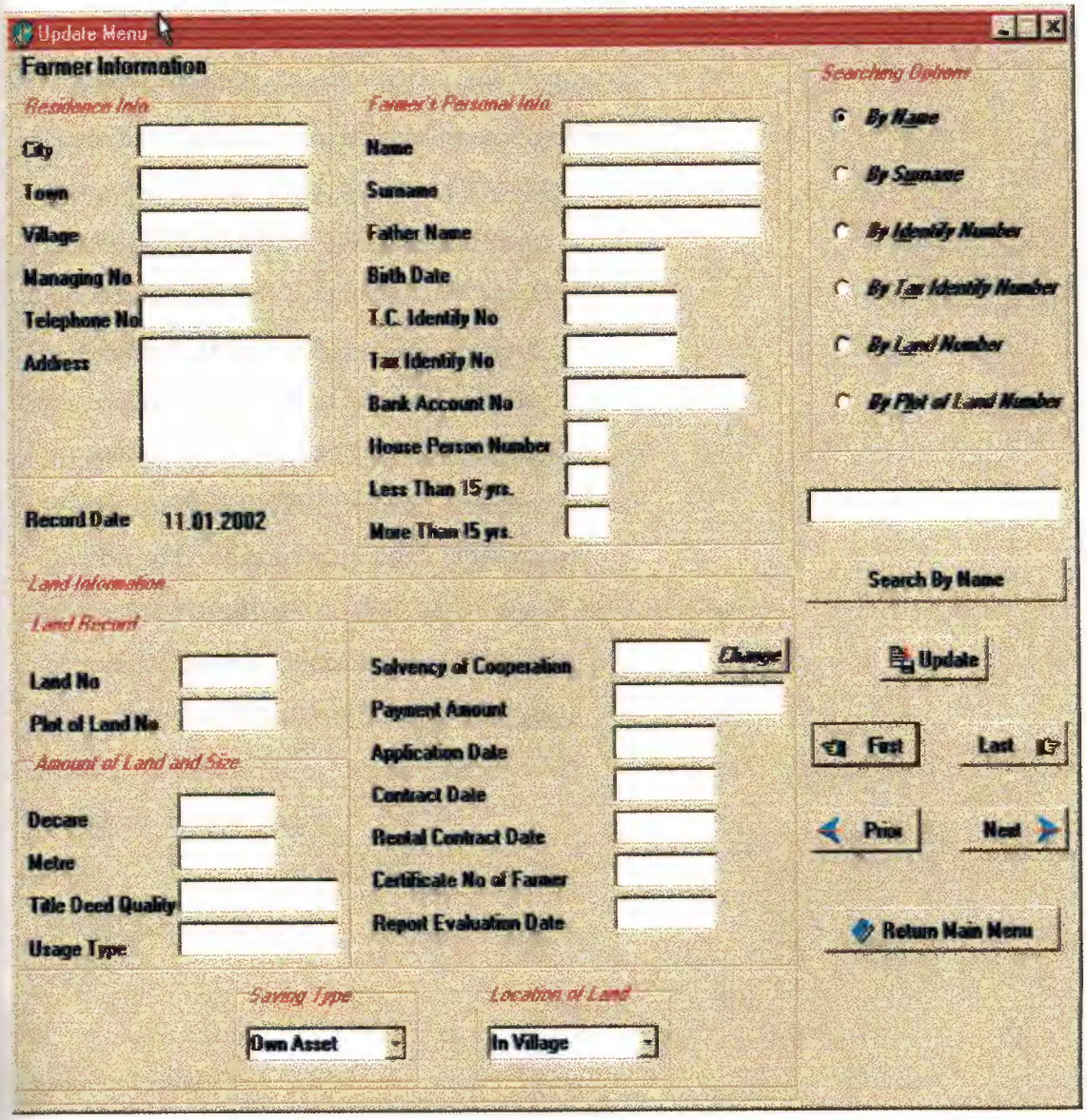
```
button4.Caption:='Search By Surname';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.RadioButton7Click(Sender: TObject);
begin
button4.Caption:='Search By Identify Number';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.RadioButton8Click(Sender: TObject);
begin
button4.Caption:='Search By Tax Identify Number';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.RadioButton9Click(Sender: TObject);
begin
button4.Caption:='Search By Land Number';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.RadioButton10Click(Sender: TObject);
begin
button4.Caption:='Search By Plot of Land Number';
edit1.text:='';
Edit1.SetFocus;
end;
procedure TForm6.BitBtn1Click(Sender: TObject);
begin
table1.refresh;
form6.close;
form1.show;
end;
```

```
procedure TForm6.BitBtn2Click(Sender: TObject);
begin
table1.Prior;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else
if Table1.Bof=true then
showmessage('Beginning of the file encountered');
end;
procedure TForm6.BitBtn3Click(Sender: TObject);
begin
table1.next;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else
if Table1.eof=true then
showmessage('Ending of the file encountered');
end;
procedure TForm6.BitBtn4Click(Sender: TObject);
begin
table1.first;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm6.BitBtn5Click(Sender: TObject);
begin
table1.last;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm6.BitBtn6Click(Sender: TObject);
begin
if messagedlg('Delete this record?',mtinformation,[mbyes,mbno],0)=mryes then
begin
```

```
if (table1.IsEmpty= true) then
showmessage('Record is Empty')
else
begin
table1.delete;
table1.prior;
table1.refresh;
end;
edit1.text:='';
end;
end;
procedure TForm6.FormActivate(Sender: TObject);
begin
Edit1.Text:='';
Edit1.SetFocus;
radiobutton5.Checked:=True;
end;
end.
```

UPDATE PART

The same searching options are also used to update the record in this menu, as it is in the searching menu. Whenever this menu is activated, fields' enabled is true. So, if the user wants to change some information or if there is any mistake in the record, there is button, which is Update Record. It updates the current record in the database. Before the record is updated, a message is given which is " Update this record?". There are also two alternatives which are Yes and No. If user presses to yes, record is updated in the database and to No, process is cancelled. Before returning the main menu, if the user does not press the update button, changes are not updated, this is cancelled. This form is shown below;



Program Codes;

```
unit Unit5;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  Db,
DBTables, StdCtrls, DBCtrls, Mask, ComCtrls, Buttons, jpeg, ExtCtrls;
type
  TForm5 = class(TForm)
    procedure RadioButton5Click(Sender: TObject);
    procedure RadioButton7Click(Sender: TObject);
    procedure RadioButton6Click(Sender: TObject);
```

```
    procedure RadioButton8Click(Sender: TObject);
    procedure RadioButton9Click(Sender: TObject);
    procedure RadioButton10Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure Button1Click(Sender: TObject);
    procedure DBEdit20Exit(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form5: TForm5;
implementation
uses Unit1, Unit11;
{$R *.DFM}
procedure TForm5.RadioButton5Click(Sender: TObject);
begin
button4.Caption:='Search By Name';
edit1.text:='';
edit1.setfocus;
end;
procedure TForm5.RadioButton7Click(Sender: TObject);
begin
button4.Caption:='Search By Identify Number';
edit1.text:='';
```

```
edit1.setfocus;
end;
procedure TForm5.RadioButton6Click(Sender: TObject);
begin
button4.Caption:='Search By Surname';
edit1.text:='';
edit1.setfocus;
end;
procedure TForm5.RadioButton8Click(Sender: TObject);
begin
button4.Caption:='Search By Tax Identify Number';
edit1.text:='';
edit1.setfocus;
end;
procedure TForm5.RadioButton9Click(Sender: TObject);
begin
button4.Caption:='Search By Land Number';
edit1.text:='';
edit1.setfocus;
end;
procedure TForm5.RadioButton10Click(Sender: TObject);
begin
button4.Caption:='Search By Plot of Land Number';
edit1.text:='';
edit1.setfocus;
end;
procedure TForm5.BitBtn1Click(Sender: TObject);
begin
table1.edit;
table1.Cancel;
table1.refresh;
form5.close;
form1.show;
end;
```

```
procedure TForm5.BitBtn4Click(Sender: TObject);
begin
table1.first;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm5.BitBtn5Click(Sender: TObject);
begin
table1.last;
if table1.IsEmpty=true then
showmessage('Record is Empty')
end;
procedure TForm5.BitBtn2Click(Sender: TObject);
begin
table1.Prior;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else if Table1.Bof=true then
showmessage('Beginning of the file encountered');
end;
procedure TForm5.BitBtn3Click(Sender: TObject);
begin
table1.next;
if table1.IsEmpty=true then
showmessage('Record is Empty')
else if Table1.eof=true then
showmessage('Ending of the file encountered');
end;
procedure TForm5.Button4Click(Sender: TObject);
begin
if radiobutton5.Checked=true then
begin
if edit1.text='' then
begin
```

```
 showmessage('Please Enter Farmer Name');
 edit1.setfocus;
 end else
 if (table1.Locate('name',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
  end;
 end else
 if radiobutton6.Checked=true then
 begin
 if edit1.text='' then
 begin
 showmessage('Please Enter Farmer Surname');
 edit1.setfocus;
 end else
 if (table1.Locate('surname',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
  end;
 end else
 if radiobutton7.Checked=true then
 begin
 if edit1.text='' then
 begin
 showmessage('Please Enter Farmer T.C. Identify Number');
 edit1.setfocus;
 end else
 if (table1.Locate('tcidentifyno',edit1.text,[]))=False then
```

```
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
  end;
end else
if radiobutton8.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Tax Identify Number');
edit1.setfocus;
end else
if (table1.Locate('taxidentifyno',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
  end;
end else if radiobutton9.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Land Number');
edit1.setfocus;
end else
if (table1.Locate('landno',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
```

```
  end;
end else
if radiobutton10.Checked=true then
begin
if edit1.text='' then
begin
showmessage('Please Enter Plot of Land Number');
edit1.setfocus;
end else
if (table1.Locate('parselno',edit1.text,[]))=False then
  begin
  showmessage('This Record Not Found');
  edit1.Text:='';
  edit1.setfocus;
  table1.cancel;
  end;
end else
begin
showmessage(' Choice one of Options');
edit1.text:='';
end;
end;
procedure TForm5.BitBtn6Click(Sender: TObject);
begin
  if messagedlg('Update this record?',mtinformation,[mbyes,mbno],0)=mryes then
  begin
   table1.edit;
   table1.UpdateRecord;
   table1.refresh;
  end;
end;
procedure TForm5.FormActivate(Sender: TObject);
begin
Edit1.Text:='';
```

```
Edit1.SetFocus;
radiobutton5.Checked:=True;
end;
procedure TForm5.FormKeyPress(Sender: TObject; var Key: Char);
var
i : integer;
begin
if (key in ['0'..'9']) or (key=#8) or (key='-') then
begin
 for i:=0 to ComponentCount-1 do
 begin
  if Components[i] is TDbEdit then
  with Components[i] as TDbEdit do
  begin
  if Tag=0 then ReadOnly := False;
  end;
 end;
end else
 for i:=0 to ComponentCount-1 do
 begin
  if Components[i] is TDbEdit then
  with Components[i] as TDbEdit do
  begin
  if Tag=0 then ReadOnly := True;
  end;
 end;
end;
procedure TForm5.Button1Click(Sender: TObject);
begin
form11.show;
form11.refresh;
end;
procedure TForm5.DBEdit20Exit(Sender: TObject);
var
```

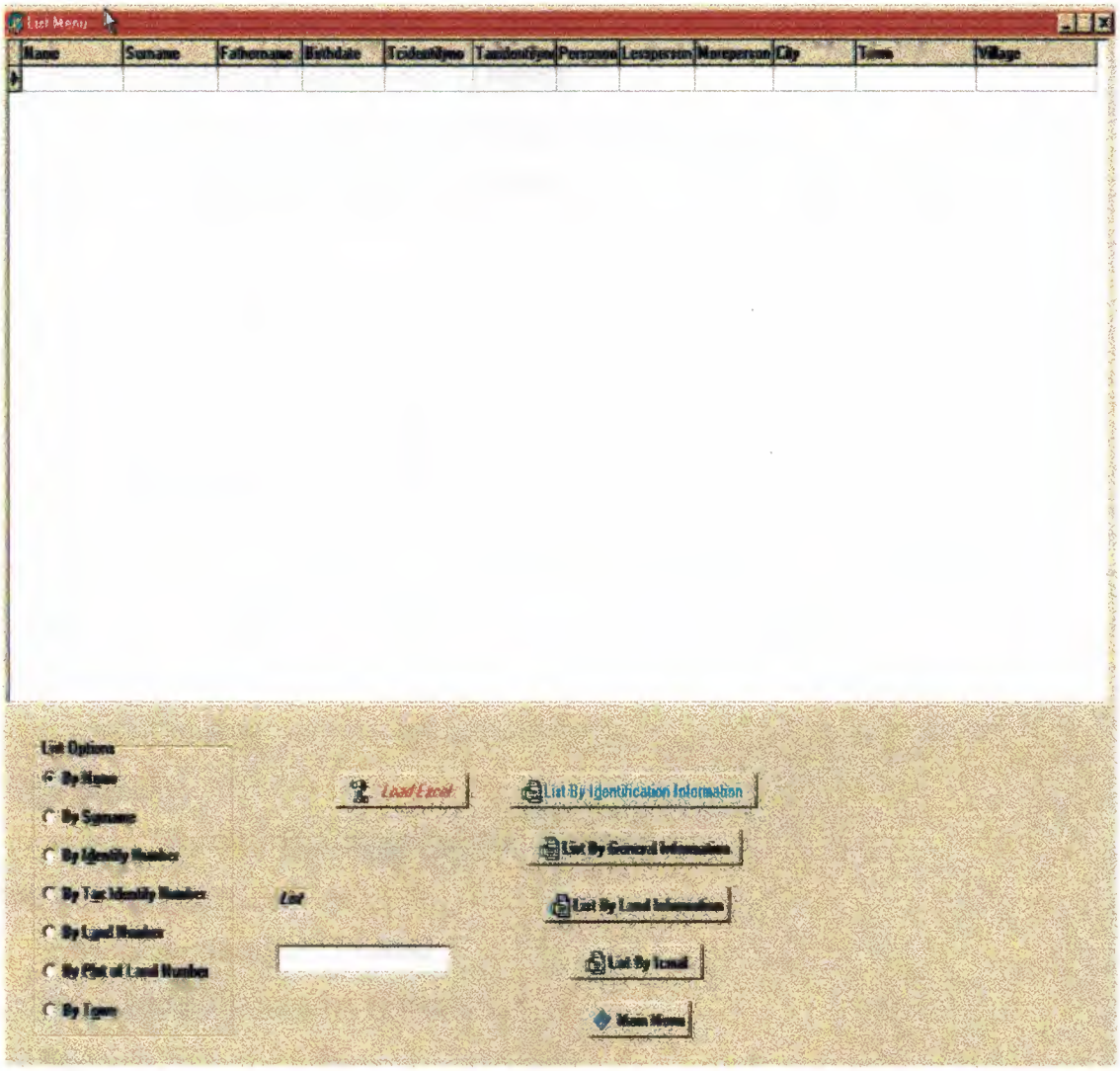
```
check1,check2,check3:longint;
result:Currency;
begin
table3.Refresh;
if dbedit20.text<>'' then
begin
check1:=strtoint(DbEdit20.text);
check2:=strtoint(DbEdit16.text);
if check1<check2 then
  begin
  messagedlg('Amount of land is less then solvency of
cooperative',mtWarning,[mbok],0);
  DbEdit20.text:='';
  DbEdit20.setfocus;
  end;
check2:=strtoint(DbEdit20.text);
check3:=strtoint(table3fixedpayment.text);
result:=check2*check3;
Dbedit21.text:=formatCurr('#,###',Result)+'TL';
DbEdit79.setfocus;
end;
end;
end.
```

LIST MENU

This is unit laced with a grid that directly shows the contents of the database. This is basically a listing tool; it has many options that let the user choose the mode of the search such as searches as by name, by surname, by identify number, by tax identify number, by land number and by plot of land number. This results in the depiction of the results in the grid according to the method chosen.

There are also several buttons provided with the unit, which are List by Identification Information, List by General Information, List by Land Information and List by Summary. They result in the depiction of the results in the grid.

There is also a button used to load the data obtained after performing a search and loading it into an excel application. Where it can be manipulated and stored, printed in a hard copy format or otherwise saved in another format as deemed suitable by the user. This form is shown below;



Program Codes;

```
unit Unit7;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  StdCtrls, ExtCtrls, DBCtrls, Grids, DBGrids, Db, DBTables, Buttons;
type
  TForm7 = class(TForm)
    procedure Edit1Change(Sender: TObject);
```

```
    procedure RadioGroup1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
  V : Variant;
    { Public declarations }
  end;
var
  Form7: TForm7;
implementation
uses Unit1,ComObj;
{$R *.DFM}
procedure TForm7.Edit1Change(Sender: TObject);
begin
    //Enumerate according byname
    if radiogroup1.ItemIndex=0 then
    begin
     table1.IndexName:='byname';
     table1.EditRangeStart;
     table1.FieldByName('name').asstring:=edit1.text;
     table1.EditRangeEnd;
     table1.FieldByName('name').asstring:=edit1.text+ chr(212);
     table1.ApplyRange;
    end;
    //Enumerate acoording surname
   if radiogroup1.ItemIndex=1 then
   begin
```

```
table1.IndexName:='bysurname';
table1.EditRangeStart;
table1.FieldByName('surname').asstring:=edit1.text;
table1.EditRangeEnd;
table1.FieldByName('surname').asstring:=edit1.text+ chr(212);
table1.ApplyRange;
end;
//Enumerate acoording T.C. Identify Number
if radiogroup1.ItemIndex=2 then
begin
table1.IndexName:='byidentifyno';
table1.EditRangeStart;
table1.FieldByName('tcidentifyno').asstring:=edit1.text;
table1.EditRangeEnd;
table1.FieldByName('tcidentifyno').asstring:=edit1.text+ chr(212);
table1.ApplyRange;
end;
//Enumerate acoording Tax Identify Number
if radiogroup1.ItemIndex=3 then
begin
table1.IndexName:='bytaxno';
table1.EditRangeStart;
table1.FieldByName('taxidentifyno').asstring:=edit1.text;
table1.EditRangeEnd;
table1.FieldByName('taxidentifyno').asstring:=edit1.text+ chr(212);
table1.ApplyRange;
end;
//Enumerate acoording Land Number
if radiogroup1.ItemIndex=4 then
begin
table1.IndexName:='bylandno';
table1.EditRangeStart;
table1.FieldByName('landno').asstring:=edit1.text;
table1.EditRangeEnd;
```

```
  table1.FieldByName('landno').asstring:=edit1.text+ chr(212);
  table1.ApplyRange;
 end;
 //Enumerate acoording Plot of Land Number
if radiogroup1.ItemIndex=5 then
begin
  table1.IndexName:='byparselno';
  table1.EditRangeStart;
  table1.FieldByName('parselno').asstring:=edit1.text;
  table1.EditRangeEnd;
  table1.FieldByName('parselno').asstring:=edit1.text+ chr(212);
  table1.ApplyRange;
end;
 //Enumerate acoording Town
if radiogroup1.ItemIndex=6 then
begin
  table1.IndexName:='bytown';
  table1.EditRangeStart;
  table1.FieldByName('town').asstring:=edit1.text;
  table1.EditRangeEnd;
  table1.FieldByName('town').asstring:=edit1.text+ chr(212);
  table1.ApplyRange;
 end;
end;
procedure TForm7.RadioGroup1Click(Sender: TObject);
begin
edit1.text:='';
if radiogroup1.ItemIndex=0 then
begin
statictext1.Caption:='List By Name';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=1 then
```

```
begin
statictext1.Caption:='List By Surname';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=2 then
begin
statictext1.Caption:='List By Identify Number';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=3 then
begin
statictext1.Caption:='List By Tax Identify Number';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=4 then
begin
statictext1.Caption:='List By Land Number';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=5 then
begin
statictext1.Caption:='List By Plot of Land Number';
edit1.setfocus;
end
else
if radiogroup1.ItemIndex=6 then
begin
statictext1.Caption:='List By Town Name';
edit1.setfocus;
end;
```

```
end;
procedure TForm7.BitBtn1Click(Sender: TObject);
begin
form7.close;
form1.show;
end;
procedure TForm7.FormActivate(Sender: TObject);
begin
table1.Refresh;
radiogroup1.ItemIndex:=0;
DbGrid4.Visible:=False;
DbGrid3.Visible:=False;
DbGrid2.Visible:=False;
DbGrid1.Visible:=True;
BitBtn2.Font.Color:=ClBlue;
BitBtn4.Font.Color:=ClWindowtext;;
BitBtn3.Font.Color:=ClWindowtext;;
BitBtn5.Font.Color:=ClWindowtext;;
end;
procedure TForm7.BitBtn4Click(Sender: TObject);
begin
DbGrid1.Visible:=False;
DbGrid3.Visible:=False;
DbGrid4.Visible:=False;
DbGrid2.Visible:=True;
BitBtn4.Font.Color:=ClBlue;
BitBtn2.Font.Color:=ClWindowtext;;
BitBtn3.Font.Color:=ClWindowtext;;
BitBtn5.Font.Color:=ClWindowtext;;
end;
procedure TForm7.BitBtn3Click(Sender: TObject);
begin
DbGrid1.Visible:=False;
DbGrid2.Visible:=False;
```

```
DbGrid4.Visible:=False;
DbGrid3.Visible:=True;
BitBtn3.Font.Color:=ClBlue;
BitBtn2.Font.Color:=ClWindowtext;;
BitBtn4.Font.Color:=ClWindowtext;;
BitBtn5.Font.Color:=ClWindowtext;;
end;
procedure TForm7.BitBtn5Click(Sender: TObject);
begin
DbGrid1.Visible:=False;
DbGrid3.Visible:=False;
DbGrid2.Visible:=False;
DbGrid4.Visible:=True;
BitBtn5.Font.Color:=ClBlue;
BitBtn2.Font.Color:=ClWindowtext;;
BitBtn4.Font.Color:=ClWindowtext;;
BitBtn3.Font.Color:=ClWindowtext;;
end;
procedure TForm7.BitBtn2Click(Sender: TObject);
begin
DbGrid4.Visible:=False;
DbGrid3.Visible:=False;
DbGrid2.Visible:=False;
DbGrid1.Visible:=True;
BitBtn2.Font.Color:=ClBlue;
BitBtn3.Font.Color:=ClWindowtext;;
BitBtn4.Font.Color:=ClWindowtext;;
BitBtn5.Font.Color:=ClWindowtext;;
end;
procedure TForm7.BitBtn6Click(Sender: TObject);
var x,j,i:integer;
begin
  table1.open;
  V := CreateOleObject('Excel.Application');
```

```
V.Visible := True;
V.Workbooks.Add;
V.cells.borders.linestyle:=1;
if DbGrid1.Visible=True then
x:=DbGrid1.FieldCount  else
if DbGrid2.Visible=True then
x:=DbGrid2.FieldCount  else
if DbGrid3.Visible=True then
x:=DbGrid3.FieldCount  else
if DbGrid4.Visible=True then
x:=DbGrid4.FieldCount;
if DbGrid3.Visible=True then
begin
 for j :=0 to x-1 do
 begin
 V.Cells[1, j + 1] :=DbGrid3.Columns.Items[j].Title.Caption;
 V.Rows.Item[j+1].Font.Bold := True;
 V.Columns[j+1].Font.size :=8 ;
 end;
 V.Columns[1].ColumnWidth := 8;
 V.Columns[2].ColumnWidth := 8;
 V.Columns[3].ColumnWidth := 8;
 V.Columns[4].ColumnWidth := 8;
 V.Columns[5].ColumnWidth := 9;
 V.Columns[6].ColumnWidth := 7;
 V.Columns[7].ColumnWidth := 7;
 V.Columns[8].ColumnWidth := 5;
 V.Columns[9].ColumnWidth := 4;
 V.Columns[10].ColumnWidth := 10;
 V.Columns[11].ColumnWidth := 10;
 V.Columns[12].ColumnWidth := 8;
 V.Columns[13].ColumnWidth := 9;
 V.Columns[14].ColumnWidth := 7;
 V.Columns[15].ColumnWidth := 14;
```

```
end else begin
for j :=0 to x-1 do
 begin
 if DbGrid1.Visible=True then
 V.Cells[1, j + 1] :=DbGrid1.Columns.Items[j].Title.Caption
 else if DbGrid2.Visible=True then
 V.Cells[1, j + 1] :=DbGrid2.Columns.Items[j].Title.Caption
 else if DbGrid4.Visible=True then
 V.Cells[1, j + 1] :=DbGrid4.Columns.Items[j].Title.Caption;
 V.Rows.Item[j+1].Font.Bold := True;
 V.Columns[j+1].ColumnWidth := 10;
 V.Columns[j+1].Font.size :=8 ;
 end;
end;
i := 1;
while not Table1.Eof do
begin
 Inc(i);
 for j :=0 to x-1 do
 begin
 if DbGrid1.Visible=True then
 v.Cells[i, j + 1] := DbGrid1.Fields[j].AsString
 else if DbGrid2.Visible=True then
 v.Cells[i, j + 1] := DbGrid2.Fields[j].AsString
 else if DbGrid3.Visible=True then
 v.Cells[i, j + 1] := DbGrid3.Fields[j].AsString
 else if DbGrid4.Visible=True then
 v.Cells[i, j + 1] := DbGrid4.Fields[j].AsString;
 v.Rows.Item[i].Font.Bold := False;;
 end;
 Table1.Next;
end;
table1.Close;
table1.open;end;  end.
```

CONCLUSION

During the course of the preparation of this program the programmer learned some very imperative facts and lessons. The easy of use of visual based languages over base-type languages was made apparent to the programmer on the very first day. This project helped immensely in the programming instinct required by a programmer as to efficiently attain his goal. Efficiency is an important goal as programmers stride to keep their programs as goal oriented as possible. Sometimes it is very easy to get lost in the hazardous by products of a problem, but a good programmer always sets his eyes on the ultimate prize; the achievement of his goal. In the due course of the completion of this project, the instructors' knowledge and expertise was of immense help, because as the programmer is a student and lacks sufficient knowledge and experience, an experienced peer can be very helpful.

The student was made aware of some features of this programming language previously unknown to him. Thus, increasing and stimulating his programming abilities. Programming projects such as the one undertaken help materialize the programmers logic and problem solving abilities. In order to survive in a global competitive environment, one has to stride to perfect oneself in these qualities

REFERENCES

References to Books

[1] Charlie Calvert, Delphi 5, Sistem Yayıncılık, Istanbul, April 1999

[2] Memik Yanık, Delphi 5 Pro, Beta Basım Yayım Dağıtım, Istanbul, May 2000

[3] Ihsan Karagülle and Zeydin Pala, Delphi 5 Professional Edition, Beta Basım Yayım Dağıtım, Istanbul, 2000

References to Programming Codes - Online Sources from Web

[1] http://www.programlama.com

[2] Delphi Super Page, http://delphi.icm.edu

[3] Torry's Delphi Pages, http://www.torry.net

[4] VCL, Components http://www.vclcomponents.com

[5] Delphi Türk, http://www.delphiturk.com

[6] Efg's Delphi Library, http://www.efg2.com

[7] Undu - Unofficial Newsletter for Delphi Users, http://www.undu.com

[8] Borland, http://www.borland.com

[9] Bimeks Online, http://www.bimeks.com.tr

[10] Experts Exchange, http://www.experts- exchange.com