**A UTOMATED LIC MANAGEMENT SYSTEM**

**PROJECT REPORT**

**ON**

**“AUTOMATED LIC MANAGEMENT SYSTEM“**

**BY:**

**Tushar Bhapkar**

**Nitin Gugale**

**Yogesh Dhome**

**UNDER THE GUIDANCE OF**

PROF. DEEPAK BADGUJAR.

PROF.RUPALI UMBARE

Department Of Information Technology

Sinhgad Institute of Technology, Lonavala.

2007–2008.

**Project Approval Sheet**

The Project Entitled

**“ AUTOMATED LIC MANAGEMENT SYSTEM “**

**BY**

Name Exam No.

**Nitin Gugale**

**Yogesh Dhome**

**Tushar Bhapkar**

Is hereby approved in partial fulfillment

for the third year in Information Technology ( Term -1 ).

**PROF. DEEPAK BADGUJAR PROF. S. P. PATIL**

**PROF.RUPALI UMBARE**

GUIDEH.O.D.

**EXTERNAL EXAMINER DR.S.S.INAMDAR** PRINCIPAL

Department Of Information Technology

Sinhgad Institute of Technology, Lonavala.

2007–2008.

**CERTIFICATE**

**AUTOMATED LIC MANAGEMENT SYSTEM**

This is to certify that the above mentioned

project has been carried out by

Name Exam No.

**Tushar Bhapkar**

**Nitin Gugale**

**Yogesh Dhome**

Students of third year engineering (Term - 1)

2007-2008

Prof. Deepak Badgujar

Prof. Rupali Umbare

Guide

Department of Information Technology

Sinhgad Institute of Technology, Lonavala.

2007–2008.

**ACKNOWLEDGEMENT**

We would like to mention our sincere gratitude towards our principal **Dr.S.S.INAMDAR** and H.O.D. **Mr.S.P.PATIL**, IT- Department, for giving us opportunity to carry out our project.

We would like to express our heart full gratitude towards our guide, **Prof. DEEPAK BADGUJAR**, for his invaluable advice for the successful completion of this dissertation.

We also like to extend our sincere thanks to other staff of the IT - Department for there invaluable help and support.

Finally we take this opportunity to mention our sincere thanks to one and all those who helped us directly or indirectly for the completion of our project.

**Tushar Bhapkar**

**Nitin Gugale**

**Yogesh Dhome**

**1. INTRODUCTION**

As we all know the LIC company involves a lot of database requirement, transactions and other complicated processes. So the work to be done in a LIC management system is lot.

Thus making software and computerizing all the work of the saves a lot of energy and manual work, thus saving cost by reducing the manpower required to do the huge amount of work.

Our software is quite small and simple as well as easy to handle as it requires very low maintenance. This software involves very less number of steps for entering, modifying or retrieving the data. Thus it is quite fast to work with.

One more specialty of this software is the User Accounts facility. This facility makes the software much secure as only users with accounts can perform the appropriate activity. Thus all the information of the customers will be safe, which is one of the very important concerns now-a-days.

**2. PURPOSE**

The purpose of this project is to make such software which is small, efficient and simple to be used in LIC offices to manage information not only on customers but also on the Agents.

And this software serves this purpose successfully.

##### 3. INTODUCTION TO VB 6.0

**UNERSTANDING EVENT DRIVEN MODEL:-**

In traditional or procedural applications, the applications itself controls, which portion of code execute and in what sequence. Execution starts with first line code and follows a predefined path through the application, calling procedures as needed.

In an Event Driven application, the code doesn’t follow the predefined path it executes different code sections in response to event. Event can be triggered by the users action, by message from the system or other application, or even from the application it self. The sequence of the events determines the sequence in which the code executes, thus the path through the applications code defer each time the program runs.

The code can also be triggered events during execution. For example, programmatically changing a text in a text box because the text box change event occur. This would cause the code contained in the change event to execute if assumed that this event would be triggered by user interaction, it might be seen unexpected results. It is for this reason it is important to understand the Event Driven model for this reason that it is important to understand the Event Driven model and keep it in mind when designing the application.

**INTERACTIVE DEVELOPMENT:-**

The traditional application development process can be broken into three distinct steps: Writing, Compiling and Testing Code. Unlike traditional languages, VB uses an interactive approach to development, blurring the distinction between the three steps.

With most languages, if a mistake is made in writing the code the compiler catches the error when you start to compile the application. It must then find and fix the error and begin the compile cycle again, repeating the process for each error found. VB interprets the code as it is entered, caching and highlighting the most syntax or spelling error on the fly. In addition to caching the errors on the fly, VB also partially compiles the code as it is entered. When it is ready to run and test your application, there is only a brief delay to finish compiling. If compiler finds the error, it is highlighted in the code. This way it is tested the effects of the code as the user works rather than waiting the compile latter.

**INTEGRATED DEVELOPMENT ENVIRONMENT:-**

The VB IDE consists of following elements:

1. **Menu Bar:**

Displays the commands to work with VB. Besides the standard file, Edit, View, Window and Help menus, menus are providing to access functions specific to programming such as Project, Format, or Debug.

1. **Context Menus:**

Contains shortcuts to frequently performed actions. To open Context menu, click the right mouse button on the object that using. The specific list of shortcuts available from context menus depends on the part of environment where the right mouse button is clicked. For example, the context menu displayed when right click on the toolbox lets the user display the components dialog box, hide the toolbox, loc or unlock the toolbox, or add custom tab to toolbox.

1. **Tool Bars:-**

Provide quick access to commonly used commands in the programming environment. A button is clicked on toolbar ones to carry out the action represent by the button. By default, the standard toolbar is displayed then VB is started. Additional toolbar for editing, form design, and debugging can be toggled on or off from on the toolbars command on the view menu.

Toolbar can be locked with the menu beneath the menu bar or can ‘float’ if the vertical bar is selected on the left edge and drag it away from the menu bar.

1. **Toolbox:-**

Provides a set of tools that you use at design time to place control on a form. In addition to default toolbox layout, selecting the Add Tab from the context menu and adding controls to the resulting tab can create the custom layouts.

1. **Project Explorer Window:-**

List the forms and module in your current project. A project is a collection of files that is used to build an application.

**6. Properties Window:-**

List the property settings for the selected form or control. A property is a characteristic of an object, such as size, caption, or color.

**7. Object Browser:-**

List object available for use in the project and give a quick way to navigate through the code. Object browser can be used to explore object in VB and other applications see what methods and properties are available for those objects, and paste code procedure into the application.

1. **Form Designer:-**

Serves as a window that you customize to design the interface of any application. By adding controls, graphics and pictures to a form, the look can be created as per requirement. Each form in the application has its own form designer window.

1. **Code Editor Window:-**

Serves as an editor for entering application code. A separate code editor window is created for each form or code module in the application.

1. **Form Layout Window:-**

It allows you to position the forms in the application using a small graphical representation of the screen.

1. **Immediate, Local and Watch Window:-**

These additional windows are provided for use in debugging your application. They are only available when application is running with IDE.

**ENVIRONMENT OPTION:-**

Single or multiple document interfaces can be chosen and the size and positioning if the various IDE elements can be adjusted.

**SDI AND MDI INTERFACES:-**

Two different styles are available for VB IDE: single document interface or multiple documents interfaces. With the SDI option, all of the IDE windows are free to be moved any where on screen; as long VB is current application, they will remain on top of any other application. With the MDI option, all of the IDE windows are contained within a single resizable parent window.

**TO SWITCH BETWEEN SDI AND MDI MODES:-**

1. Select OPTIONS from the TOOLS menu.
2. Select the ADVANCED tab.
3. Check or uncheck the SDI Development Environment check box.

**DOCKING WINDOWS:-**

Many of the windows in the IDE can be docked, or connected, to each other to the edge of the screen. These include the toolbox, form layout window, project explorer, properties window, color palette.

**TO DOCK OR UNDOCK WINDOW:-**

1. Select a window you wish to dock or undock.
2. Drag the window to the desired location by holding down the left mouse button.
3. The outline of the window will be displayed by dragging.
4. Release the mouse button.

**UNDERSTANDING PROPERTIES, METHODS AND EVENTS:-**

VB forms and controls are objects that expose their own properties, method and events. Properties can be thought of as an object attributes, methods as it’s action, and events as it’s response.

An everyday objects like a child’s helium balloon also has properties, methods and events. A balloon’s properties include visible attributes such as its height, diameter, and color. Other property describes its age. By definition, all balloons have these properties; the settings of these properties may differ from one balloon to another.

A balloon also has inherent methods or actions that it might perform. It has an inflate method, a deflate method and a rise method you were to let go of it. Again all balloons are capable of this method. Balloons also have predefined responses to certain external events. For instance, a balloon would respond to the event of being released by rising into the air. If you were able to program a balloon, a VB code might look like the following. To set the balloons properties:

Balloon. Color= Red

Balloon. Diameter= 10

Balloon. Inflated= True.

**4. INTRODUCTION TO MS-ACCESS**

Essentially Access is the Data Base Management System (DBMS). Like other products in this category, access stores and retrieves data, presents information and automates repetitive tasks such as maintaining account payable, performing Hotel Management Controls and Scheduling.

Access is also a powerful window application. Access runs on the Windows95, Windows98 or Windows NT platform.

As a relational database manager, it provides access to all types of data and allows the use of more than one database table at a time. It can reduce the complexity of your data and make it easier to get your job done.

**API**

**VBA**

**MACROS**

**FUNCTIONS/EXPRESSIONS**

**OBJECTS, TABLES, QUERIES,**

**FORMS & REPORTS.**

**THE ACCESS USABILITY HIERARCHY (CONCEPT FOR ACCESS):**

This simple figure conveys the message that access is useable at all levels.

Beginning at the lowest level of the hierarchy and moving upwards, you see objects list first, these give the end user the capability of creating tables, quires, forms and reports easily.

You can perform simple processing by using expressions, also known as functions, to validate data, enforce a business rule, or display a number with currency symbols. Macros allow the automation without programming, whereas VBA (Visual Basic for application) code lets the user programs complex process. Finally, by using windows API (application programming interface) calls to function or DLL’s (Dynamic Link Libraries) written in other language such as C, JAVA, or VB, a programmer can write interfaces to other programs and data sources.

Access is a set of tools for end user database management. Access has a table creator, a form designer, and a data access page creator and report writer. Access is also an environment for development application using MACROS or modules to automate the task; you can create user-oriented applications as powerful as those created with programming languages complete with the buttons, means and dialog boxes.

**ACCESS OFFERS:-**

**TRUE RELATIONAL DATABASE MANAGEMENT:-**

Access provides true relational database management. Access includes definitions for primary and foreign key and has full referential integrity at the database design engine level itself (which prevents inconsistent updates or deletions).

In additional, tables in access here data-validation rules to prevent inaccurate data regardless of how data is entered and every field in a table has format and default definitions for more productive data entry.

Access supports all the necessary field types, including text, number, auto number (counter), currency, data/time, memo, yes/no, and hyperlink and OLE objects. When values are missing in special processing, access provides full supports for null values.

The relational processing in access files many needs with its flexible architecture. It can also be used as a stand alone database management system, in a file server configuration, or as a front-end client to product such as a SQL server. In addition, access features ODBC (Open Data Base Connectivity), which permits connections to many external formats, such as SQL server, oracle, Sybase or mainframe IBM db/2.

The program provides complete support for transaction, processing, ensuring the integrity of transactions. In addition, user level security provides control over assigning user and group permissions to view to modify database objects.

###### CONTROLS USED IN PROJECT

TextBox Control



A TextBox control, sometimes called an edit field or edit control, displays information entered at design time, entered by the user, or assigned to the control in code at run time.

To display multiple lines of text in a TextBox control, set the MultiLine property to True. If a multiple-line TextBox doesn't have a horizontal scroll bar, text wraps automatically even when the TextBox is resized. To customize the scroll bar combination on a TextBox, set the ScrollBars property.

Scroll bars will always appear on the TextBox when its MultiLine property is set to True, and its ScrollBars property is set to anything except None (0).

If you set the MultiLine property to True, you can use the Alignment property to set the alignment of text within the TextBox. The text is left justified by default. If the MultiLine property is False, setting the Alignment property has no effect.

A TextBox control can also act as a destination link in a DDE conversation.

ListBox Control



A ListBox control displays a list of items from which the user can select one or more. If the number of items exceeds the number that can be displayed, a scroll bar is automatically added to the ListBox control.

If no item is selected, the ListIndex property value is -1. The first item in the list is ListIndex 0, and the value of the ListCount property is always one more than the largest ListIndex value.

To add or delete items in a ListBox control, use the AddItem or RemoveItem method. Set the List, ListCount, and ListIndex properties to enable a user to access items in the ListBox. Alternatively, you can add items to the list by using the List property at design time.

Common Dialog Control



The CommonDialog control provides a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts. The control also has the ability to display help by running the Windows Help engine.

The CommonDialog control provides an interface between Visual Basic and the routines in the Microsoft Windows dynamic-link library Commdlg.dll. To create a dialog box using this control, Commdlg.dll must be in your Microsoft Windows SYSTEM directory.

You use the CommonDialog control in your application by adding it to a form and setting its properties. The dialog displayed by the control is determined by the methods of the control. At run time, a dialog box is displayed or the help engine is executed, when the appropriate method is invoked; at design time, the CommonDialog control is displayed as an icon on a form. This icon can't be sized.

The CommonDialog control can display the following dialogs using the specified method.

|  |  |
| --- | --- |
| Method | Dialog Displayed |
| ShowOpen | Show Open Dialog Box |
| ShowSave | Show Save As Dialog Box |
| ShowColor | Show Color Dialog Box |
| ShowFont | Show Font Dialog Box |
| ShowPrinter | Show Print or Print Options Dialog Box |
| ShowHelp | Invokes the Windows Help Engine |

The CommonDialog control automatically provides context sensitive help on the interface of the dialog boxes by clicking:

The What's This help button in the title bar then clicking the item for which you want more information.

The right mouse button over the item for which you want more information then selecting the What's This command in the displayed context menu.

The operating system provides the text shown in the Windows 95 (or later) Help popup. You can also display a Help button on the dialog boxes with the CommonDialog control by setting the Flags property, however, you must provide the help topics in this situation.

Note   There is no way to specify where a dialog box is displayed.

RichTextBox Control



The RichTextBox control allows the user to enter and edit text while also providing more advanced formatting features than the conventional TextBox control.

The RichTextBox control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents.

The RichTextBox control opens and saves files in both the RTF format and regular ASCII text format. You can use methods of the control (LoadFile and SaveFile) to directly read and write files, or use properties of the control such as SelRTF and TextRTF in conjunction with Visual Basic's file input/output statements.

The RichTextBox control supports object embedding by using the OLEObjects collection. Each object inserted into the control is represented by an OLEObject object. This allows you to create documents with the control that contain other documents or objects. For example, you can create a document that has an embedded Microsoft Excel spreadsheet or a Microsoft Word document or any other OLE object registered on your system. To insert objects into the RichTextBox control, you simply drag a file (from the Windows 95 Explorer for example), or a highlighted portion of a file used in another application (such as Microsoft Word), and drop the contents directly onto the control.

The RichTextBox control supports both clipboard and OLE drag/drop of OLE objects. When an object is pasted in from the clipboard, it is inserted at the current insertion point. When an object is dragged and dropped into the control, the insertion point will track the mouse cursor until the mouse button is released, causing the object to be inserted. This behavior is the same as Microsoft Word.

To print all or part of the text in a RichTextBox control use the SelPrint method.

Because the RichTextBox is a data-bound control, you can bind it with a Data control to a Binary or Memo field in a Microsoft Access database or a similar large capacity field in other databases (such as a TEXT data type field in SQL Server).

The RichTextBox control supports almost all of the properties, events and methods used with the standard TextBox control, such as MaxLength, MultiLine, ScrollBars, SelLength, SelStart, and SelText. Applications that already use TextBox controls can easily be adapted to make use of RichTextBox controls. However, the RichTextBox control doesn't have the same 64K character capacity limit of the conventional TextBox control.

Distribution Note   To use the RichTextBox control in your application, you must add the Richtx32.ocx file to the project. When distributing your application, install the Richtx32.ocx file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the Programmer's Guide.

**Image Control**



Use the Image control to display a graphic. An Image control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, JPEG, or GIF files.

The Image control uses fewer system resources and repaints faster than a PictureBox control, but it supports only a subset of the PictureBox properties, events, and methods. Use the Stretch property to determine whether the graphic is scaled to fit the control or vice versa. Although you can place an Image control within a container, an Image control can't act as a container.

Note   The Unisys Corporation has a patent that it alleges covers certain aspects of GIF-LZW compression technology, which the PictureBox and Image controls use. Microsoft Corporation obtained a license to the Unisys LZW patents in September, 1996. Microsoft's license does not, however, extend to software developers or third parties who use any Microsoft toolkit, language development, or operating system products to provide GIF read/write and/or any other LZW capabilities in their own products (for example, by way of DLLs and APIs).

If your commercial application uses one of these controls (and thus, the LZW technology), you may wish to obtain an independent legal opinion on the effect of the patent, or contact Unisys USA at http://www.unisys.com/ for more information.

**Data Grid Control**

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

Syntax

DataGrid

Remarks

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

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

If a cell's text is too long to be displayed in the cell, the text wraps to the next line within the same cell. To display the wrapped text, you must increase the cell's Column object's Width property and/or the DataGrid control's RowHeight property. At design time, you can change the column width interactively by resizing the column or by changing the column's width in the Column object's property page.

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

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

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

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

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

Note   This control is Unicode-enabled. When used on a Unicode-enabled system such as Microsoft Windows NT, the control passes Unicode data with no conversion. On other systems, however, data is converted from ANSI to Unicode and back. For more information, see "ANSI, DBCS, and Unicode: Definitions" in the Programmer's Guide.

**RowColChange Event**

Occurs when the current cell changes to a different cell.

Syntax

Private Sub object\_RowColChange ([index As Integer, lastrow As String, lastcol As Integer])

The RowColChange event syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| Index | An integer that uniquely identifies a control if it is in a control array. |
| last row | (For DataGrid control) A string expression specifying the previous row position. |
| Lastcol | (For DataGrid control) An integer specifying the previous column position. |

Remarks

This event occurs whenever the user clicks a cell other than the current cell or when you programmatically change the current cell within a selection using the Col and Row properties.

The SelChange event also occurs when a user clicks a new cell, but doesn't occur when you programmatically change the selected range without changing the current cell.

For the DataGrid control, the position of the current cell is provided by the Bookmark and ColIndex properties. The previous cell position is specified by lastrow and lastcol. If you edit data and then move the current cell position to a new row, the update events for the original row are completed before another cell becomes the current cell

**IMPORTANT PROPERTIES**

Text Property

ComboBox control (Style property set to 0 [Dropdown Combo] or to 1 [Simple Combo]) and TextBox control — returns or sets the text contained in the edit area.

ComboBox control (Style property set to 2 [Dropdown List]) and ListBox control — returns the selected item in the list box; the value returned is always equivalent to the value returned by the expression List (ListIndex). Read-only at design time; read-only at run time.

Syntax

object.Text [= string]

The Text property syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| String | A string expression specifying text. |

Remarks

At design time only, the defaults for the Text property are:

ComboBox and TextBox controls — the control's Name property.

ListBox control — a zero-length string ("").

For a ComboBox with the Style property set to 0 (Dropdown Combo) or to 1 (Simple Combo) or for a TextBox, this property is useful for reading the actual string contained in the edit area of the control. For a ComboBox or ListBox control with the Style property set to 2 (Dropdown List), you can use the Text property to determine the currently selected item.

The Text setting for a TextBox control is limited to 2048 characters unless the MultiLine property is True, in which case the limit is about 32K.

FileName Property

Returns or sets the path and filename of a selected file. Not available at design time for the FileListBox control.

Syntax

object.FileName [= pathname]

The FileName property syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| Pathname | A string expression that specifies the path and filename. |

Remarks

When you create the control at run time, the FileName property is set to a zero-length string (""), meaning no file is currently selected.

In the CommonDialog control, you can set the FileName property before opening a dialog box to set an initial filename.

Reading this property returns the currently selected filename from the list. The path is retrieved separately, using the Path property. The value is functionally equivalent to List (ListIndex). If no file is selected, FileName returns a zero-length string.

When setting this property:

Including a drive, path, or pattern in the string changes the settings of the Drive, Path, and Pattern properties accordingly.

Including the name of an existing file (without wildcard characters) in the string selects the file.

Filter Property (CommonDialog)

Returns or sets the filters that are displayed in the Type list box of a dialog box.

Syntax

object.Filter [= description1 |filter1 |description2 |filter2...]

The Filter property syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| Description | A string expression describing the type of file. |
| Filter | A string expression specifying the filename extension. |

Remarks

A filter specifies the type of files that are displayed in the dialog box's file list box. For example, selecting the filter \*.txt displays all text files.

Use this property to provide the user with a list of filters that can be selected when the dialog box is displayed.

Use the pipe ( | ) symbol (ASCII 124) to separate the description and filter values. Don't include spaces before or after the pipe symbol, because these spaces will be displayed with the description and filter values.

The following code shows an example of a filter that enables the user to select text files or graphic files that include bitmaps and icons:

Text (\*.txt)|\*.txt|Pictures (\*.bmp;\*.ico)|\*.bmp;\*.ico

When you specify more than one filter for a dialog box, use the FilterIndex property to determine which filter is displayed as the default.

Data Type

String

DataSource Property

Returns or sets a data source through which a data consumer is bound to a database.

Syntax

object.DataSource [=datasource]

The DataSource property syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| datasource | An object reference that qualifies as a data source, including ADO Recordset objects, and classes or user controls defined as data sources (DataSourceBehavior property = vbDataSource). |

Remarks

Use the Set statement to set the DataSource property, as shown below:

Set Text1.DataSource = ADODC1

Note   Two older controls, the Data control and RemoteData control, can be used as data sources, however you cannot set the DataSource property of another control or object to either of these controls at run time. For example, the following code will fail:

Set Text1.DataSource = Data1 ' Will fail! You can't set DataSource at

' run time to an intrinsic Data control.

To use either the Data control or RemoteData control as a data source, you can set the DataSource property of bound controls at design time only.

Visual Basic: DataGrid Control

DefColWidth Property

Returns or sets a value indicating the default column width for all columns in the DataGrid control.

Syntax

object.DefColWidth [= value]

The DefColWidth property syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | An object expression that evaluates to an object in the Applies To list. |
| Value | An integer based on the scale mode of the control. |

Remarks

If you set the DefColWidth property to 0, the control automatically sizes all columns based on either the width of the column heading or the Size property setting of the underlying field, whichever is larger. For example, to set the default column width of all columns to the width of the first column:

DataGrid1.DefColWidth = DataGrid1.Columns(0).Width

**IMPORTANT METHODS**

Add Item Method

Adds an item to a ListBox or ComboBox control or adds a row to a MS Flex Grid control. Doesn't support named arguments.

Syntax

object.AddItem item, index

The AddItem method syntax has these parts:

|  |  |
| --- | --- |
| Part | Description |
| Object | Required. An object expression that evaluates to an object in the Applies To list. |
| Item | Required. string expression specifying the item to add to the object. For the MS Flex Grid control only, use the tab character (character code 09) to separate multiple strings you want to insert into each column of a newly added row. |
| Index | Optional. Integer specifying the position within the object where the new item or row is placed. For the first item in a ListBox or ComboBox control or for the first row in a MS Flex Grid control, index is 0. |

Remarks

A ListBox or ComboBox control that is bound to a Data control doesn't support the AddItem method.

Clear Method (Clipboard, ComboBox, ListBox)

Clears the contents of a ListBox, ComboBox, or the system Clipboard.

Syntax

object.Clear

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

A ListBox or ComboBox control bound to a Data control doesn't support the Clear method.

ShowOpen Method

Displays the CommonDialog control's Open dialog box.

object.ShowOpen

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

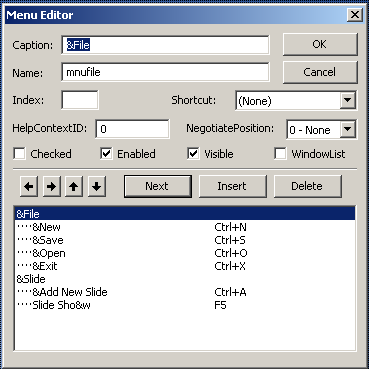
ShowSave Method

Displays the CommonDialog control's Save As dialog box.

object.ShowSave

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

Menu Editor Dialog Box



Allows you to create custom menus for your application and to define their properties.

Dialog Box Options

Caption

Allows you to enter the menu or command name that you want to appear on your menu bar or in a menu.

If you want to create a separator bar in your menu, type a single hyphen (-) in the Caption box.

To give the user keyboard access to a menu item, insert an ampersand (&) before a letter. At run time, this letter is underlined (the ampersand is not visible), and the user can access the menu or command by pressing ALT and the letter. If you need an ampersand to show in the menu, put two consecutive ampersands in the caption.

Name

Allows you to enter a control name for the menu item. A control name is an identifier used only to access the menu item in code; it doesn't appear in a menu.

Index

Allows you to assign a numeric value that determines the control's position within a control array. This position isn't related to the screen position.

Shortcut

Allows you to select a shortcut key for each command.

HelpContextID

Allows you to assign a unique numeric value for the context ID. This value is used to find the appropriate Help topic in the Help file identified by the HelpFile property.

NegotiatePosition

Allows you to select the menu's NegotiatePosition property. This property determines whether and how the menu appears in a container form.

Checked

Allows you to have a check mark appear initially at the left of a menu item. It is generally used to indicate whether a toggle option is turned on or off.

Enabled

Allows you to select whether you want the menu item to respond to events, or clear if you want the item to be unavailable and appear dimmed.

Visible

Allows you to have the menu item appear on the menu.

WindowList

Determines if the menu control contains a list of open MDI child forms in an MDI application.

   Right Arrow

Moves the selected menu down one level each time you click it. You can create up to four levels of submenus.

 Left Arrow

Moves the selected menu up one level each time you click it. You can create up to four levels of submenus.

   Up Arrow

Moves the selected menu item up one position within the same menu level each time you click it.

   Down Arrow

Moves the selected menu item down one position within the same menu level each time you click it.

Menu List

A list box that displays a hierarchical list of menu items. Submenu items are indented to indicate their hierarchical position or level.

Next

Moves selection to the next line.

Insert

Inserts a line in the list box above the currently selected line

Delete

Deletes the currently selected line.

OK

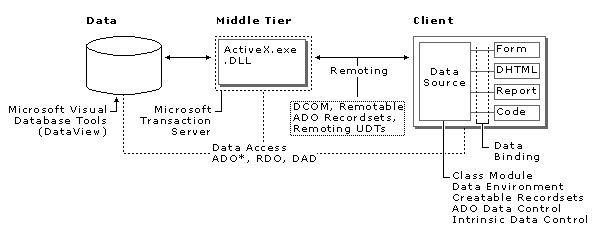
Closes the Menu Editor and applies all changes to the last form you selected. The menu is available at design time, but selecting a menu at design time opens the Code window for that menu's Click event rather than executing any event code.

Cancel

Closes the Menu Editor and cancels all changes.

###### Accessing Data Using Visual Basic

The figure below is a roadmap of data access technologies found in Visual Basic. The figure features "hot" zones, which you can click to find out more information about any particular set of data, access tools or technologies.



**Microsoft Visual Data Tools**

Using Visual Basic 6.0 you can create components that encapsulate every step in a data access system. Beginning with the data source, Microsoft Visual Data Tools (accessible through the Data View window) give you the ability to view and manipulate tables, views, stored procedures, and database schemas on SQL Server and Oracle systems.

Middle Tier Components and Microsoft Transaction Server

The power of Visual Basic is also leveraged to create the middle tier components in your application, as you make your own ActiveX DLLs and EXEs. Visual Basic now includes enhancements that tailor applications to work with Microsoft Transaction Server.

ActiveX Data Objects (ADO)

The bridge between the data providers and data consumers is through data sources created using Microsoft ActiveX Data Objects (ADO), which is the primary method in Visual Basic to access data in any data source, both relational and non-relational. For backward compatibility and project maintenance, Remote Data Objects (RDO) and Data Access Objects (DAO) are still supported.

Data Sources and Data Controls

On the client side, several new data sources are available, including the Data Environment, a graphical designer that allows you to quickly create ADO Connections and Commands to access your data. The Data Environment designer provides a dynamic programmatic interface to the data access objects in your project. In addition, the Data Environment provides advanced data shaping services — the ability to create hierarchies of related data, aggregates, and automatic groupings, all without code.

The new ADO Data control is similar to the intrinsic data control and Remote Data control, except that it uses ADO to access data. You can now use an ADO Recordset as a data source for your controls and objects in Visual Basic.

In Visual Basic you can now create your own data sources either as user controls or classes, to encapsulate business rules or proprietary data structures. The class module now features the DataSourceBehavior property and the GetDataMember event, which allow you to configure a class as a data source.

Dynamic Data Binding

The ability to dynamically bind a data source to a data consumer is now possible in Visual Basic. At run time, you can now set the DataSource property of a data consumer (such as the DataGrid control) to a data source (such as the ADO Data control). This capability, unavailable in previous versions of Visual Basic, allows you to create applications, which can access a multitude of data sources.

Presenting Data to the End User

Visual Basic offers a variety of rich ways to present data to your end users. ADO/OLE DB-based versions of all the data bound controls are included in Visual Basic:

The DataList and DataCombo controls are the ADO/OLE DB equivalents of DBList and

DBCombo controls.

The DataGrid is the successor to DBGrid.

The Chart control is now data bound.

A new version of the FlexGrid control, called the Hierarchical FlexGrid, supports the hierarchical abilities of the Data Environment.

The new DataRepeater control functions as a scrolling container of data bound user controls where each control views a single record.

The Data Report is a new ActiveX designer that creates reports from any data source, including the Data Environment. With the Data Report designer, formatted reports can be viewed online, printed, or exported to text or HTML pages.

Data Formatting and Data Validation

The new DataFormat object allows you to display data with custom formatting, but write it back to the database in the native format. For example, you can now display dates in the format appropriate to a country, while the actual data is stored in a date format. Data is formatted coming out of the source, and unformatted going back in. You can also do custom formatting and perform additional checks using the Format and

Unformat events.

Data validation is also enhanced using the CausesValidation property with the Validate event. By setting the CausesValidation property to True, the Validate event for the previous control in the tab order will occur. Thus, by programming the Validate event, you can prevent a control from losing focus until the information it contains has been validated.

Language Features

New data-related enhancements to the Visual Basic language include the ability to pass User-defined Types (UDTs) and arrays across processes. You can now define a UDT and pass it as a parameter to another process, such as an ActiveX EXE or DLL.

DHTML and Data Access

Using Visual Basic, you can create complete web applications for data access. All of the data tools and technologies can also be used in DHTML pages, and on web server (IIS) applications.

Other Enterprise features

For information on data access programming in the enterprise, see Data Access Strategies in Developing for the Enterprise.

**ADO, DAO and RDO in Visual Basic**

In Visual Basic, three data access interfaces are available to you: ActiveX Data Objects (ADO), Remote Data Objects (RDO), and Data Access Objects (DAO). A data access interface is an object model that represents various facets of accessing data. Using Visual Basic, you can programmatically control the connection, statement builders, and returned data for use in any application.

Why are there three data access interfaces in Visual Basic? Data access technology is constantly evolving, and each of the three interfaces represent a different state of the art. The latest is ADO, which features a simpler — yet more flexible — object model than either RDO or DAO. For new projects, you should use ADO as your data access interface.

Why Use ADO?

ADO is designed as an easy-to-use application level interface to Microsoft's newest and most powerful data access paradigm, OLE DB. OLE DB provides high-performance access to any data source, including relational and non-relational databases, email and file systems, text and graphics, custom business objects, and more. ADO is implemented for minimal network traffic in key Internet scenarios, and a minimal number of layers between the front-end and data source — all to provide a lightweight, high-performance interface. ADO is called using a familiar metaphor — the OLE Automation interface. And ADO uses conventions and features similar to DAO and RDO, with simplified semantics that make it easy to learn.

For a brief overview, see OLE DB Providers.

For detailed information about ADO, see Getting Started with ADO.

DAO and RDO

For backward compatibility, Visual Basic continues to support DAO and RDO for existing projects.

For More Information   For more information on RDO programming, see Using Remote Data Objects and the RemoteData Control. For information on DAO programming, see Using Data Access Objects with Remote Databases. Complete DAO reference can also be found at Microsoft DAO 3.6.

Upgrading from RDO to ADO

Consider upgrading if you decide ADO offers benefits your RDO-based application can use. See ADO Compared with RDO and DAO for a discussion of the differences among the platforms and for guidance on changing an RDO-based project to an ADO project. See Converting from RDO 2.0 to ADO 2.0 for upgrade guidance.

**ADO Programming Model in Detail**

The following are key elements of the ADO programming model:

Connection

Command

Parameter

Recordset

Field

Error

Property

Record

Stream

Collection

Event

Connection

You gain access from your application to a data source through a connection—the environment necessary for exchanging data. Your application can gain access to a data source directly (sometimes called a two-tier system), or indirectly (sometimes called a three-tier system) through an intermediary, such as Microsoft Internet Information

Services (IIS).

The object model embodies the concept of a connection with the Connection object.

A transaction delimits the beginning and end of a series of data access operations that transpire across a connection. ADO ensures that changes to a data source resulting from operations in a transaction either all occur successfully, or not at all.

If you cancel the transaction or one of its operations fails, then the ultimate result will be as if none of the operations in the transaction had occurred. The data source will be as it was before the transaction began.

The object model does not explicitly embody the concept of a transaction, but represents it with a set of Connection object methods.

ADO accesses data and services from OLE DB providers. The Connection object is used to specify a particular provider and any parameters. For example, Remote Data Service (RDS) can be invoked explicitly or implicitly with the Microsoft OLE DB Remoting Provider. (Please see Step 2 of the RDS Tutorial for an example of invoking RDS via the OLE DB Remoting Provider.)

The data source that is the target of a connection may be specified with a connection string or a Uniform Resource Locator (URL).

Command

A command issued across an established connection manipulates the data source in some way. Typically the command adds, deletes, or updates data in the data source, or retrieves data in the form of rows in a table.

The object model embodies the concept of a command with the Command object. The existence of a Command object gives ADO the opportunity to optimize the execution of

the command.

Parameter

Often, commands require variable parts, or parameters, that can be altered before you issue the command. For example, you could issue the same data retrieval command repeatedly, but each time vary your specification of the information to be retrieved.

Parameters are especially useful for executing commands that behave like functions. In this case you know what the command does, but not necessarily how it works. For example, you issue a bank transfer command that debits one account and credits another. You specify the amount of money to be transferred as a parameter.

The object model embodies the concept of a parameter with the Parameter object.

Recordset

If your command is a query that returns data as rows of information in a table (that is, it is a row-returning query), then those rows are placed in local storage.

The object model embodies this storage as a Recordset object.

The Recordset is the primary means of examining and modifying data in the rows. The Recordset object allows you to:

Specify which rows are available for examination.

Traverse the rows.

Specify the order in which the rows may be traversed.

Add, change, or delete rows.

Update the data source with changed rows.

Manage the overall state of the Recordset.

Field

A row of a Recordset consists of one or more fields. If you envision the Recordset as a two-dimensional grid, the fields line up to form columns. Each field (column) has among its attributes a name, a data type, and a value. It is this value that contains the actual data from the data source.

The object model embodies a field as a Field object.

In order to modify data in the data source, you modify the value of Field objects in Recordset rows. Ultimately, changes to a Recordset are propagated to the data source. As an option, the transaction management methods on the Connection object can guarantee that the changes succeed or fail in unison.

Error

Errors can occur at any time in your application, usually as the result of not being able to establish a connection, execute a command, or perform an operation on an object in a suitable state (for example, attempting to use a Recordset object that has not been initialized).

The object model embodies an error as an Error object.

Any given error produces one or more Error objects. The next error that occurs will discard the previous set of Error objects.

Property

Each ADO object has a set of unique properties that either describe or control the behavior of that object.

There are two types of properties: built-in and dynamic. Built-in properties are part of the ADO object and are always available. Dynamic properties are added to the ADO object's Properties collection by the underlying data provider or service provider, and exist only while that provider is being used.

The object model embodies a property as a Property object.

Record

Not all data sources exist as tables in a database. Information storage systems, such as file and e-mail systems, consist of container and content components. A container may hold content or other, subordinate containers.

In a file system, the containers and contents are directories and files; in an e-mail system, the containers and contents are folders and messages.

The object model embodies a container or content as a Record object. Furthermore, a row of a Recordset may be embodied as a Record object.

The Record object provides the means to:

Copy, delete, or move the item it represents.

Create a new Record suitable for representing items such as a directory or file, or a row of a Recordset.

A Record object is used in conjunction with other ADO objects, such as the Connection and Recordset objects. However, a Record object is designed to solve a different set of issues than the ADO objects described in the Basic ADO Programming Model. See Records and Streams for more information.

Stream

The content of an information storage system, such as a file in a file system, consists of a stream of bytes. Also, a buffer in memory consists of a stream of bytes.

The object model embodies a stream of bytes as a Stream object.

The Stream object provides means to:

Read or write a series of bytes, or lines of text.

Populate itself from, or persist itself to, a file.

A Stream object is used only in conjunction with the Record object. A Stream object is designed to solve a different set of issues than the ADO objects described in the Basic ADO Programming Model. See Records and Streams for more information.

Collection

ADO provides collections, a type of object that conveniently contains other objects of a particular type. The objects in the collection can be retrieved with a collection property either by name, as a text string, or by ordinal, as an integer number.

ADO provides four types of collections:

The Connection object has the Errors collection, which contains all Error objects created in response to a single failure involving the data source.

The Command object has the Parameters collection, which contains all Parameter objects that apply to that Command object.

The Recordset and Record objects have the Fields collection, which contains all Field objects that define the columns of that Recordset object.

In addition, the Connection, Command, Recordset, and Field objects all have the Properties collection, which contains all the Property objects that apply to their respective containing objects.

ADO objects possess properties in which you set or retrieve values with common data types like INTEGER, CHARACTER, or BOOLEAN. However, it is useful to think of certain properties as returning values of data type "COLLECTION OBJECT." The collection object, in turn, has methods to store and retrieve other objects suitable for the collection.

For example, you can think of the Recordset object as having a Properties property that returns a collection object. That collection object has methods to store and retrieve Property objects describing attributes of that Recordset.

Events

Events are notifications that certain operations are about to occur, or have already occurred. You can use events to efficiently orchestrate an application consisting of several asynchronous tasks.

The object model does not explicitly embody events, but represents them as calls to event handler routines.

Event handlers called before the operation starts offer you the opportunity to examine or modify the operation parameters, then either cancel or allow the operation to complete.

Event handlers called after an operation completes notify you at the completion of an asynchronous operation. Several operations have been enhanced to optionally execute asynchronously. For example, an application that starts an asynchronous Recordset.Open operation is notified by an execution complete event when the operation concludes.

**Triggers in the SQL Editor**

A trigger is a special kind of stored procedure that goes into effect when you modify data in a specified table using one or more data modification operations: UPDATE, INSERT, or DELETE. Triggers can query other tables and can include complex SQL statements. They are primarily useful for enforcing complex business rules or requirements. For example, you could control whether to allow an order to be inserted based on a customer's current account status.

Triggers are also useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update, or delete the rows in those tables. However, the best way to enforce referential integrity is to define primary key and foreign key constraints in the related tables. If you use database diagrams, you can create a relationship between tables to automatically create a foreign key constraint.

Advantages of Using Triggers

**Triggers are useful in these ways**:

Triggers are automatic: they are activated immediately after any modification to the table's data, such as a manual entry or an application action.

Triggers can cascade changes through related tables in the database. For example, you can write a delete trigger on the title\_id column of the titles table to cause a deletion of matching rows in other tables. The trigger uses the title\_id column as a unique key to locate matching rows in the titleauthor, sales, and roysched tables.

Triggers can enforce restrictions that are more complex than those defined with check constraints. Unlike check constraints, triggers can reference columns in other tables. For example, a trigger can roll back updates that attempt to apply a discount (stored in the discounts table) to books (stored in the titles table) with a price of less than $10.

Creating a Trigger

A trigger is a database object that you create by specifying:

The current table.

The data modification transactions that activate the trigger: adding new data (INSERT), updating existing data (UPDATE), or deleting existing data (DELETE).

The actions that the trigger will take immediately following the transactions you specified.

You write triggers in Transact-SQL for Microsoft® SQL Server™ databases or PL/SQL for Oracle databases.

To create a trigger

In Data View (available from the Standard toolbar or the View menu), expand the Tables folder.

Right-click the name of the table that you want to create a trigger on. Choose New Trigger from the shortcut menu.

A new trigger is created with the following SQL statements already defined for you:

Create Trigger /\*Trigger\_Name\*/

on /\*Table\_name\*/

For /\*Insert, Update, Delete\*/

As

print 'Trigger Fired'

Modify the trigger text as follows:

|  |  |  |
| --- | --- | --- |
| Line | Replace | With |
| 1 | /\*Trigger\_Name\*/ | The name you want to assign to the trigger |
| 2 | /\*Table\_name\*/ | The name of the table you want to attach the trigger to |
| 3 | /\*Insert, Update, Delete\*/ | The type of transactions that will activate this trigger |

For example, to create a trigger named employee\_insupd for insert and update transactions on the employee table, you would change the first three lines of the trigger text to the following:

Create Trigger employee\_insupd

on employee

For Insert, Update

Write the remaining trigger text in SQL.

**Opening a Trigger**

You can open a trigger to view or edit the text of an existing trigger that is stored in your database. Triggers are scripted in Transact-SQL for Microsoft® SQL Server™ databases or PL/SQL for Oracle databases.

**To open a trigger**

In Data View, expand the Tables folder.

Expand the table whose trigger you want to open.

Right-click the name of the trigger that you want to open and click Design on the shortcut menu.

-or-

Double-click the name of the trigger that you want to open.

**Saving a Trigger**

To add a new trigger to the database or to update an existing trigger that you have modified, you can save a trigger.

**To save a trigger**

In the SQL Editor, choose Save to Database from the File menu.

If you are updating an existing trigger, a message box prompts you to confirm the save action. Choose Yes.

A saved trigger appears in the Tables folder in Data View under the table that it's attached to.

Deleting a Trigger

To disable the actions defined in the trigger that are automatically carried out on your database immediately following the specified transactions, you can delete a trigger.

You might also want to delete any triggers that enforce referential integrity between related tables if you use database diagrams to design your database. Database diagrams use relationships instead of triggers for this purpose. Thus, if a trigger duplicates a relationship in a database diagram, you should delete either the trigger or the relationship.

**To delete a trigger**

In Data View, expand the Tables folder.

Expand the table whose trigger you want to delete.

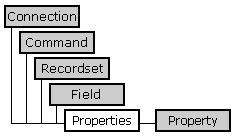
Right-click the trigger that you want to delete and choose Delete from the shortcut menu.

A message prompts you to confirm the deletion. Choose Yes.

The trigger is deleted from the database and Data View.

**Recordset Object**

Represents the entire set of records from a base table or the results of an executed command. At any time, the Recordset object refers to only a single record within the set as the current record.



Remarks

You use Recordset objects to manipulate data from a provider. When you use ADO, you manipulate data almost entirely using Recordset objects. All Recordset objects consist of records (rows) and fields (columns). Depending on the functionality supported by the provider, some Recordset methods or properties may not be available.

ADODB.Recordset is the ProgID that should be used to create a Recordset object. Existing applications that reference the outdated ADOR.Recordset ProgID will continue to work without recompiling, but new development should reference ADODB.Recordset.

There are four different cursor types defined in ADO:

**Dynamic cursor** — allows you to view additions, changes, and deletions by other users; allows all types of movement through the Recordset that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.

**Keyset cursor** — behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the Recordset.

**Static cursor** — provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the Recordset. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.

**Forward-only cursor** — allows you to only scroll forward through the Recordset. Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a Recordset.

Set the CursorType property prior to opening the Recordset to choose the cursor type, or pass a CursorType argument with the Open method. Some providers don't support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

**CursorLocation property**

If the CursorLocation property is set to adUseClient to open a Recordset, the UnderlyingValue property on Field objects is not available in the returned Recordset object. When used with some providers (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create Recordset objects independently of a previously defined Connection object by passing a connection string with the Open method. ADO still creates a Connection object, but it doesn't assign that object to an object variable. However, if you are opening multiple Recordset objects over the same connection, you should explicitly create and open a Connection object; this assigns the Connection object to an object variable. If you do not use this object variable when opening your Recordset objects, ADO creates a new Connection object for each new Recordset, even if you pass the same connection string.

You can create as many Recordset objects as needed.

## ***Open Method (ADO Recordset)***

Opens a cursor.

**Syntax**

*recordset***.Open** *Source***,** *ActiveConnection***,** *CursorType***,** *LockType***,** *Options*

**Parameters**

*Source*Optional. A **Variant** that evaluates to a valid Command object, an SQL statement, a table name, a stored procedure call, a URL, or the name of a file or Stream object containing a persistently stored Recordset.

*ActiveConnection*Optional. Either a **Variant** that evaluates to a valid Connection object variable name, or a **String** that contains ConnectionString parameters.

*CursorType*Optional. A CursorTypeEnum value that determines the type of cursor that the provider should use when opening the **Recordset**. The default value is **adOpenForwardOnly**.

## ***CursorTypeEnum***

Specifies the type of cursor used in a Recordset object.

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| **adOpenDynamic** | 2 | Uses a dynamic cursor. Additions, changes, and deletions by other users are visible, and all types of movement through the **Recordset** are allowed, except for bookmarks, if the provider doesn't support them. |
| **adOpenForwardOnly** | 0 | Default. Uses a forward-only cursor. Identical to a static cursor, except that you can only scroll forward through records. This improves performance when you need to make only one pass through a **Recordset**. |
| **adOpenKeyset** | 1 | Uses a keyset cursor. Like a dynamic cursor, except that you can't see records that other users add, although records that other users delete are inaccessible from your **Recordset**. Data changes by other users are still visible. |
| **adOpenStatic** | 3 | Uses a static cursor. A static copy of a set of records that you can use to find data or generate reports. Additions, changes, or deletions by other users are not visible. |
| **adOpenUnspecified** | -1 | Does not specify the type of cursor. |

## ***LockTypeEnum***

  Specifies the type of lock placed on records during editing.

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| **adLockBatchOptimistic** | 4 | Indicates optimistic batch updates. Required for batch update mode. |
| **adLockOptimistic** | 3 | Indicates optimistic locking, record by record. The provider uses optimistic locking, locking records only when you call the Update method. |
| **adLockPessimistic** | 2 | Indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately after editing. |
| **adLockReadOnly** | 1 | Indicates read-only records. You cannot alter the data. |
| **adLockUnspecified** | -1 | Does not specify a type of lock. For clones, the clone is created with the same lock type as the original. |

## ***AddNew Method***

Creates a new record for an updatable Recordset object.

**Syntax**

*recordset***.AddNew** *FieldList***,** *Values*

**Parameters**

*recordset*A **Recordset** object.

*FieldList*Optional. A single name, or an array of names or ordinal positions of the fields in the new record.

*Values*Optional. A single value, or an array of values for the fields in the new record. If *Fieldlist* is an array, *Values* must also be an array with the same number of members; otherwise, an error occurs. The order of field names must match the order of field values in each array.

**Remarks**

Use the **AddNew** method to create and initialize a new record. Use the Supports method with **adAddNew** (a CursorOptionEnum value) to verify whether you can add records to the current **Recordset** object.

After you call the **AddNew** method, the new record becomes the current record and remains current after you call the Update method. If the **Recordset** object does not support bookmarks, you may not be able to access the new record once you move to another record. Depending on your cursor type, you may need to call the Requery method to make the new record accessible.

If you call **AddNew** while editing the current record or while adding a new record, ADO calls the **Update** method to save any changes and then creates the new record.

The behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether you pass the *Fieldlist* and *Values* arguments.

In *immediate update mode* (in which the provider writes changes to the underlying data source once you call the **Update** method), calling the **AddNew** method without arguments sets the EditMode property to **adEditAdd** (an EditModeEnum value). The provider caches any field value changes locally. Calling the **Update** method posts the new record to the database and resets the **EditMode** property to **adEditNone** (an **EditModeEnum** value). If you pass the *Fieldlist* and *Values* arguments, ADO immediately posts the new record to the database (no **Update** call is necessary); the **EditMode** property value does not change (**adEditNone**).

In *batch update mode* (in which the provider caches multiple changes and writes them to the underlying data source only when you call the UpdateBatch method), calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method adds the new record to the current **Recordset** and resets the **EditMode** property to **adEditNone**, but the provider does not post the changes to the underlying database until you call the **UpdateBatch** method. If you pass the *Fieldlist* and *Values* arguments, ADO sends the new record to the provider for storage in a cache; you need to call the **UpdateBatch** method to post the new record to the underlying database.

If the Unique Table dynamic property is set, and the **Recordset** is the result of executing a JOIN operation on multiple tables, then the **AddNew** method can insert fields only into the table named in the **Unique Table** property.

Update Method

Saves any changes you make to the current row of a Recordset object, or the Fields collection of a Record object.

**Syntax**

*recordset***.Update** *Fields***,** *Values*

*record.Fields***.Update**

**Parameters**

*Fields*Optional. A **Variant** that represents a single name, or a **Variant** array that represents names or ordinal positions of the field or fields you wish to modify.

*Values*Optional. A **Variant** that represents a single value, or a **Variant** array that represents values for the field or fields in the new record.

**Remarks**

**Recordset**

Use the **Update** method to save any changes you make to the current record of a **Recordset** object since calling the AddNew method or since changing any field values in an existing record. The **Recordset** object must support updates.

To set field values, do one of the following:

* Assign values to a Field object's Value property and call the **Update** method.
* Pass a field name and a value as arguments with the **Update** call.
* Pass an array of field names and an array of values with the **Update** call.

When you use arrays of fields and values, there must be an equal number of elements in both arrays. Also, the order of field names must match the order of field values. If the number and order of fields and values do not match, an error occurs.

If the **Recordset** object supports batch updating, you can cache multiple changes to one or more records locally until you call the UpdateBatch method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider.

If you move from the record you are adding or editing before calling the **Update** method, ADO will automatically call **Update** to save the changes. You must call the CancelUpdate method if you want to cancel any changes made to the current record or discard a newly added record.

The current record remains current after you call the **Update** method.

If the Unique Table dynamic property is set, and the **Recordset** is the result of executing a JOIN operation on multiple tables, then the **Update** method cannot update any primary key of the multiple tables. Furthermore, the **Update** method can update fields only in the table specified in the Unique Table property.

**Record**

The **Update** method finalizes additions, deletions, and updates to fields in the Fields collection of a **Record** object.

For example, fields deleted with the **Delete** method are marked for deletion immediately but remain in the collection. The **Update** method must be called to actually delete these fields from the provider's collection.

## ***MoveFirst, MoveLast, MoveNext, and MovePrevious Methods***

Moves to the first, last, next, or previous record in a specified Recordset object and makes that record the current record.

**Syntax**

*recordset***.**{**MoveFirst** | **MoveLast** | **MoveNext** | **MovePrevious**}

**Remarks**

Use the **MoveFirst** method to move the current record position to the first record in the **Recordset**.

Use the **MoveLast** method to move the current record position to the last record in the **Recordset**. The **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

Use the **MoveNext** method to move the current record position one record forward (toward the bottom of the **Recordset**). If the last record is the current record and you call the **MoveNext** method, ADO sets the current record to the position after the last record in the **Recordset** (EOF is **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

In cases where the **Recordset** has been filtered or sorted and the current record's data is changed, the position may also change. In such cases the **MoveNext** method works normally, but you should be aware that the position is moved one record forward from the new position, not the old position. For example, changing the data in the current record, such that the record is moved to the end of the sorted **Recordset,** would mean that calling **MoveNext** results in ADO setting the current record to the position after the last record in the **Recordset** (**EOF** = **True**).

Use the **MovePrevious** method to move the current record position one record backward (toward the top of the **Recordset**). The **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the **MovePrevious** method, ADO sets the current record to the position before the first record in the **Recordset** (BOF is **True**). An attempt to move backward when the **BOF** property is already **True** generates an error. If the **Recordset** object does not support either bookmarks or backward cursor movement, the **MovePrevious** method will generate an error.

If the **Recordset** is forward only and you want to support both forward and backward scrolling, you can use the Cache Size property to create a record cache that will support backward cursor movement through the Move method. Because cached records are loaded into memory, you should avoid caching more records than is necessary. You can call the **MoveFirst** method in a forward-only **Recordset** object; doing so may cause the provider to re-execute the command that generated the **Recordset** object.

**RECORDSET PROPERTIES**:

## ***BOF, EOF Properties***

* **BOF** — Indicates that the current record position is before the first record in a Recordset object.
* **EOF** — Indicates that the current record position is after the last record in a **Recordset** object.

**Return Value**

The **BOF** and **EOF** properties return **Boolean** values.

**Remarks**

Use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you've gone beyond the limits of a **Recordset** object when you move from record to record.

The **BOF** property returns **True** (-1) if the current record position is before the first record and **False** (0) if the current record position is on or after the first record.

The **EOF** property returns **True** if the current record position is after the last record and **False** if the current record position is on or before the last record.

If either the **BOF** or **EOF** property is **True**, there is no current record.

If you open a **Recordset** object containing no records, the **BOF** and **EOF** properties are set to **True**, and the **Recordset** object's RecordCount property setting is zero. When you open a **Recordset** object that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

If you delete the last remaining record in the **Recordset** object, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

This table shows which **Move** methods are allowed with different combinations of the **BOF** and **EOF** properties.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **MoveFirst, MoveLast** | **MovePrevious, Move < 0** | **Move 0** | **MoveNext, Move > 0** |
| **BOF=True,** **EOF=False** | Allowed | Error | Error | Allowed |
| **BOF=False,** **EOF=True** | Allowed | Allowed | Error | Error |
| Both **True** | Error | Error | Error | Error |
| Both **False** | Allowed | Allowed | Allowed | Allowed |

Allowing a **Move** method doesn't guarantee that the method will successfully locate a record; it only means that calling the specified **Move** method won't generate an error.

The following table shows what happens to the **BOF** and **EOF** property settings when you call various **Move** methods but are unable to successfully locate a record.

|  |  |  |
| --- | --- | --- |
|  | **BOF** | **EOF** |
| **MoveFirst**, **MoveLast** | Set to **True** | Set to **True** |
| **Move** 0 | No change | No change |
| **MovePrevious**, **Move** < 0 | Set to **True** | No change |
| **MoveNext**, **Move** > 0 | No change | Set to **True** |

## ***RecordCount Property***

Indicates the number of records in a Recordset object.

**Return Value**

Returns a **Long** value that indicates the number of records in the **Recordset**.

**Remarks**

Use the **RecordCount** property to find out how many records are in a **Recordset** object. The property returns -1 when ADO cannot determine the number of records or if the provider or cursor type does not support **RecordCount**. Reading the **RecordCount** property on a closed **Recordset** causes an error.

If the **Recordset** object supports approximate positioning or bookmarks—that is, **Supports (adApproxPosition)** or **Supports (adBookmark)**, respectively, return **True**—this value will be the exact number of records in the **Recordset**, regardless of whether it has been fully populated. If the **Recordset** object does not support approximate positioning, this property may be a significant drain on resources because all records will have to be retrieved and counted to return an accurate **RecordCount** value.

The cursor type of the **Recordset** object affects whether the number of records can be determined. The **RecordCount** property will return -1 for a forward-only cursor; the actual count for a static or keyset cursor; and either -1 or the actual count for a dynamic cursor, depending on the data source.

**5. Project Description**

This software is implemented in programming language Visual Basic and database stored in MS-Access. Connection between this projects and database are done in ODBC of control panels.

Active-x controls are used in the project that inbuilt applications are available for the project.

**SYSTEM:**

1. **Policy : -**

This form gives us the functionality of maintaining a proper record of all the clients those have take the policy of different plans. It helps to keep the record of new client and there policy plans.

1. **Premium: -**

This form gives facility to pay premium amount to client and also it helps to get claim amount to nominee.

1. **Premium Receipt:**

This form givespremium receipt to client that contains details about branch, premium status, client name, premium amount, due date, amount paid after due date.

1. **Search: -**

This form gives facility to user to search particular record by client id, client name, agent id, agent name, nominee id, nominee name etc.

1. **Details: -**

This form gives detail information about client, agent, policy, nominee, premium, claim, etc.

1. **Help : -**

This form gives facility to user to view the features of each policy. It also shows the benefits of each policy.

1. **MDI: -**

This form contains file strip of to select one of the form mentioned above.

**Advantages :**

**1. User accounts facility: -**

This software has user accounts facility which makes the information stored in this software safe and secure.

**2. Simple usability and user friendly: -**

This software is quite easy to use and user friendly as the steps to be performed for any operation is quite simple and straight-forward.

**8. BIBLIOGRAPHY**

Books used :

1. Black Book Of VB 6.0 - Steven Holzner

2. Database Management Systems - Koarth

3. SQL for professionals - Rajesh Naik

4. Database Management systems - Bipin Desai

Softwares used :

1. Oracle iSQL
2. Microsoft Visual Basic 6.0

Language used :

SQL (Structured Query Language)