

Demo of a Method for Managing Multiple Small Lists

There are many occasions where a database developer suddenly finds that it would be quite helpful to the end users and for the integrity of the data if there were several small lists of information that could be displayed in various combo boxes and/or list boxes from which the user could simply make selections. I realize that the term “small list” is a very relative term and can be interpreted to mean different things. However, I would suggest that a “small list” is a list that normally has less than 20 items and that the list is primarily a static list of values with no requirement for any associated data for each value.

Note: If you, as the developer, feel that there is any requirement for additional items of data for a specific “list type” then you should not include that “list type” in this method for the management of “small lists” but you should consider creating this list in its own table.

Examples of some of these types of list might be:

<i>List Type</i>	<i>Value Examples</i>
Prefix to Name	Mr.; Mrs.; Miss; Dr.; etc.
Suffix for Name	Sr.; Jr.; I; II; III; Esq.; etc.
Title	CEO; Owner; President; etc.
Contact Type	Individual; Corporate; Personal; etc.
Vendor Type	Office Supplies; General Merchandise; etc.
Officers	President; Vice President; Secretary; etc.
Grant Status	Application Sent; Pending; Approved; etc.
Expense Type	Refreshments; Decorations; Operational; etc.

While the development of these types of lists is certainly not a daunting task, if approached using a standard method for the presentation of these small lists, the developer will be faced with the choice of developing several tables to hold each of the desired list types or actually hard coding the list of values in a combo box using the “Value List” option. Not only is the creation of multiple tables normally required, but then the developer is also required to provide some type of user interface that will allow the user to not only add values to each list but also to allow users to edit, delete and/or even be able to flag one value as no longer active. This can require the development of several forms as well as a navigation system.

When observed in the overall scheme of things, utilizing a method that would allow users to manage multiple small lists, like the ones described above, from a single table with only one user interface method required, might prove to be a more streamlined solution and would also reduce the work load on the developer.

This is just the type of solution that I am proposing here. Instead of requiring the creation of multiple tables, this method will only require that we create a single table to hold the data needed to define and identify the various lists needed for one application. We will also define this table in a way that will allow the user to have the capability of defining and managing even the sort order of each small list. Another feature that is easily

implemented with this method is to providing users with the ability to flag individual values within any list as being either “active” or “inactive”. This is a feature that is quiet beneficial to users, especially when values have previously been used and associated with any record in the database, but no longer need to be available for future selection from the list.

For the purpose of this demonstration, you can either create a new database file and create all of the content to recreate the functionality for Managing Small Lists or you might actually want to create this functionality in an existing application that you have already started.

I will assume that you already know how to create a new Access database file or to open your existing application.

Create a Table to Hold Data

Note: *This demo is presented using and describing options available in Access 2002/2003. Access to of some of the options described in this demo will be located in a different place and the screen depictions will appear much different if you are using a later version of Access (2007 or 2010). However, the general method for creating and using these options is basically the same.*

Create a new table and define the following 6 fields in the new table:

Field Name	Field Type	Field Size	Default Value
ListValId	AutoNumber	Long Integer	N/A
ListType	Text	50	N/A
ListValue	Text	50	N/A
SortOrder	Number	Byte	N/A
Active	Yes/No	N/A	Yes
StatusChangeDate	Date/time	N/A	N/A

Below is a graphical representation of how the table should look when defined.

	Field Name	Data Type
	ListValId	AutoNumber
	ListType	Text
	ListValue	Text
	SortOrder	Number
	Active	Yes/No
	StatusChangeDate	Date/Time

Please note that the “ListValId” field has been identified as a Primary Key and the “ListValue” field’s indexed property has been set to “Yes (duplicates OK)”. The “SortOrder” field is defined as a Byte type number. Save your new table, naming it: “tblListInfo”.

It is my opinion that it is always best to have some “controlled” data in tables before starting to design any type of user interface to that data. “Controlled” data means that the

data that you have in your table will reflect the data that will actually be stored in the table but does not necessarily include an extensive amount of data. In our case, we only want a couple of “list types” and a few “list items” for each “list type”. With data like this, it is much easier to determine if the implementation of your user interface is accomplishing the desired goals. (In real world application development you may not always have the luxury of working with a small dataset, but for beginners, it is a good practice and a great method for learning.)

So, for now, let’s add just very a small amount of data to our table. Please use the list to add the new data to the new table:

<i>List Type</i>	<i>List Value</i>	<i>Sort Order</i>	<i>Active</i>
Prefix	Mr.	1	True
Prefix	Mrs.	2	True
Prefix	Miss	3	True
Suffix	Sr.	1	True
Suffix	Jr.	2	True
Suffix	III	3	True

When data entry is complete, save and close the table.

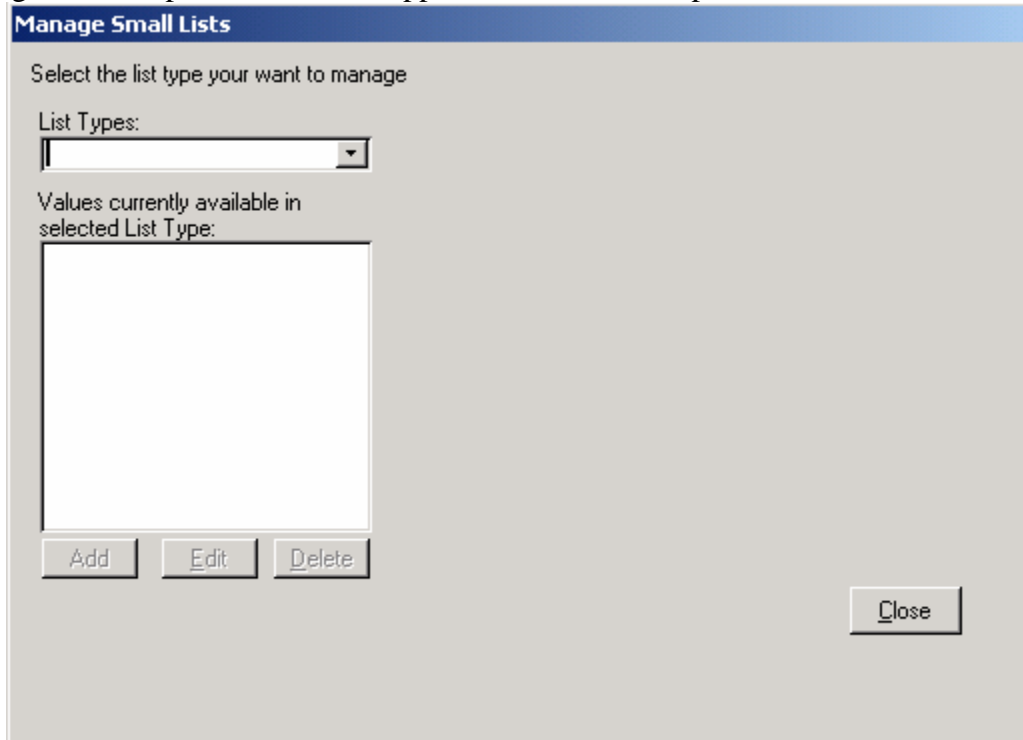
Create the User Interface (Form)

In the world of database applications, users should never have direct access to the tables where the data is stored. All data should be presented to the user through the use of forms. So, the next thing we must have in order to be able to manage these lists is a form to allow users to interact with the data.

It is very helpful if you as the developer can have at least a mental picture of just how you want to present data to your users. You should also have a pretty good idea of just what kind of functionality should be available to the users when working with the data.

In this case, our over all objective is to provide the user with the ability to have and manage multiple small lists of values that will eventually be presented to users as options available from list or combo boxes. These values will provide users with a data entry process that produces much more consistent data. More specifically for this demo, we will need to provide users with a way to see all of the “list types” that are available. Users will need the ability to determine which of the “list types” they want to work with. When they have decided on a “list type” they will need some way to see the entire list of items that are currently available for the selected “list type”. Users will also need some way to modify existing values, add any new values that may be needed, delete any values that have never been used in data entry and a way to “deactivate” specific values in the list to prevent the ability to select that value in the future. Users will also need a method that will allow them to “reactivate” values that have been “deactivated”, should those values again be available for selection. We have also committed to providing the user with the ability to define a custom sort order for each “list type”.

If you are a true beginner, at this point, you may be finding it difficult, if not totally impossible, to get a complete mental picture of the design that we would need for our form, the controls that we might need to accomplish our goals for the project or even how start to design the user interface. So, to give you an idea of just where we are going, here is a screen shot of the form that we are going to build that will actually allow users to manage the multiple lists as it will appear when it is first opened.

The image shows a Windows-style dialog box titled "Manage Small Lists". Inside the dialog, there is a label "Select the list type you want to manage". Below this is a "List Types:" label followed by a dropdown menu. Underneath the dropdown is another label "Values currently available in selected List Type:" followed by a large, empty rectangular box. At the bottom left of the dialog, there are three buttons labeled "Add", "Edit", and "Delete". At the bottom right, there is a "Close" button.

The "frmManageListInfo" form shown in "Form View"

Here is the same form, as it might appear when a user was actually using the options on our form to manage the "Prefix" list values.

The "frmManageListInf" from shown as being used to manage a list


Now, here is the same form in design view. Note the additional controls that are visible in this view that are not visible in the initial view or in the view above.

Form shown in "Design View"

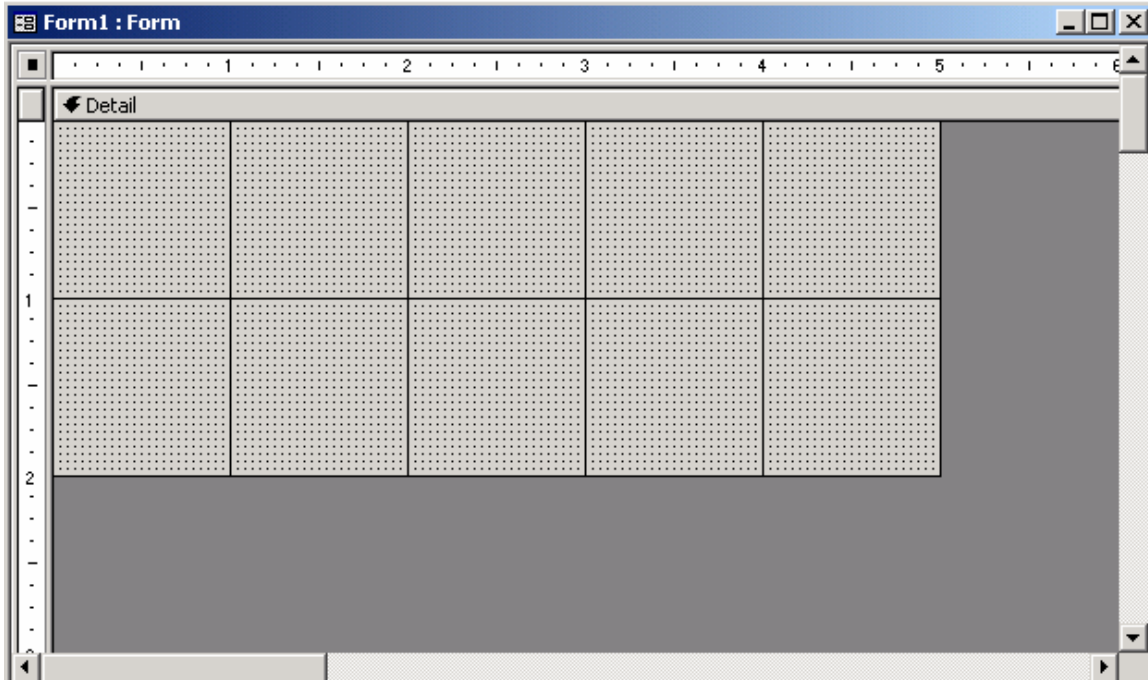
As we proceed through this demo, please keep in mind that I have tried to write this documentation in a way that will make it possible for even a beginner to follow along and

be able complete the building of this management tool. It is also my goal to provide more advanced Access users with some ideas for managing these small lists. So, if I seem to go into too much detail, then you may be a more advanced user. If so, just move along quickly, remembering when you were a beginning Access user and would have loved for someone to provide a detailed explanation of what you were trying to do.

Now, let's create this form. We will be creating what is know as an "Unbound Form". This means that there is no specific record source (table or query) that is "bound" to the form and therefore, no fields are "bound" to any of the controls on the form. For this exercise, we will not be using the Wizard when creating our forms but rather we will just create the form and add the required controls one at a time. I am taking this approach so that you can get a better understanding of just how much control the developer has when developing a user interface. Just keep in mind that you as the developer have a very high level of control over everything relating to the user interface.

If, as you add controls, you are seeing the Wizard popup every time you try to add a control, click on the "Wizard" button, , in the "Toolbox" to turn it off. Later if you need to turn it back on, just click the button again to turn it back on.

First we will create the new form. From the database window, select the "Forms" option from the objects list. Click "New" from the menu. Select "Design View" from the top list. You do not need to select any object or table that the data comes from. Click the "OK" button to create the blank form as show below.




Blank Form shown in "Design View"

Do not be concerned about the initial size of your form. We will soon be changing things about our form by setting various properties of the form. Save this new form as "frmManageListInfo". Please make note of the method being used to "name" this form

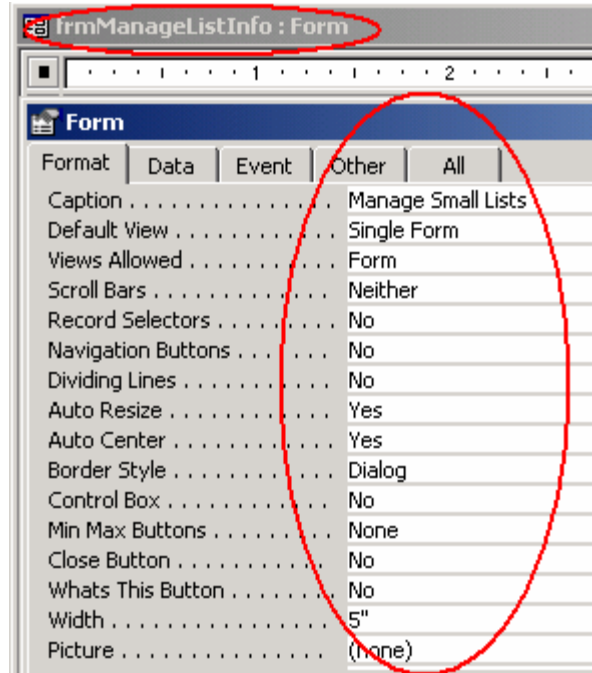
and other objects later on. First, we use a “prefix” of “frm” to identify that the object is a form. Next, we use “camel” case for the actual name. This means that the first character of each word in the name is capitalized and no spaces or underscores are used in the name. In my opinion, it is essential that objects be named in a meaningful way so that they are more easily identified later on.

Note: If you are attempting to create your new form in the demo database file, you will not be able to save your form with the same name as my original form. It is perfectly fine to save your new form using another form name, but later when we are working with VBA code, you will need to remember that your form has a different name and you will need to use your form’s name in place of the original form name.

We will be using something call the “Properties” dialog box to make changes to various properties of our form, various areas of the form and even to the controls that we place on our form. Please note that the “Properties” dialog box will always display the properties for the selected object. So, make sure that you actually have the object that you wish to work with selected. If the “Properties” dialog box is not already displayed, there are several ways to make the “Properties” dialog box visible. One way to display the

“Properties” dialog box is to click the Properties button, , on the toolbar. You can also display the “Properties” dialog box by selecting “Properties” from the “View” menu. You can also place your cursor over the “title bar” of your form and right-click and select “Properties” from the shortcut menu. When you use any of these options they act as a toggle. If the “Properties” dialog box is not visible, using one of these options will make the “Properties” dialog box visible. If it is visible, using one of these options will cause it to be not visible any more.


With the “Properties” dialog box for the form displayed, set the identified properties using the graphic below.

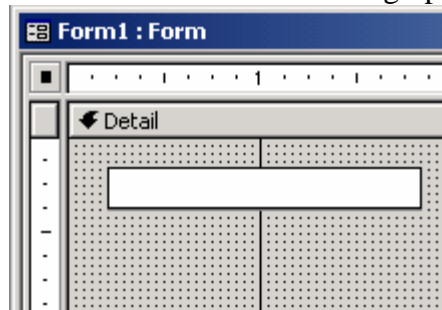


Properties of the "frmManageListInfo" form

Next, click on the "Details" section bar of the form to show the "Properties" of this section. Locate the "Height" property of this section and set its value to: "3.125".

Save your form again.

Now we will begin to add controls to our new form. In the "Toolbox" click on the "Label" control button, . Move your cursor over the upper left of your form and click and drag to create the label control to look similar to the graphic below.

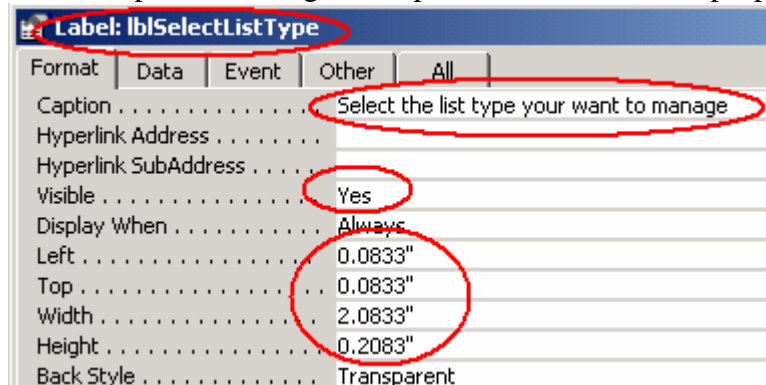


First Label control added

You will notice that your cursor is now blinking in the new Label control. Immediately type "Select the list type you want to manage:" to add the caption for this control.


Note: If you do not provide any text for the caption of a new label control, that new label control will disappear so you need to immediately add at least a space or some character to the label to hold it on the form.

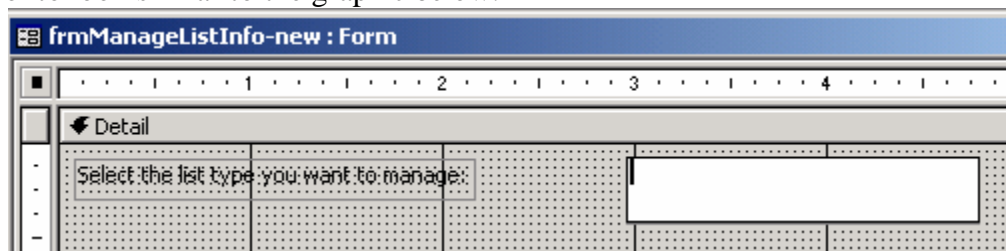
When you have finished typing the caption, press the Tab or Enter key to complete your entry. Click on the new Label to select it and use the identified information on the “Format” tab of the “Properties” dialog box depicted below to set the properties.



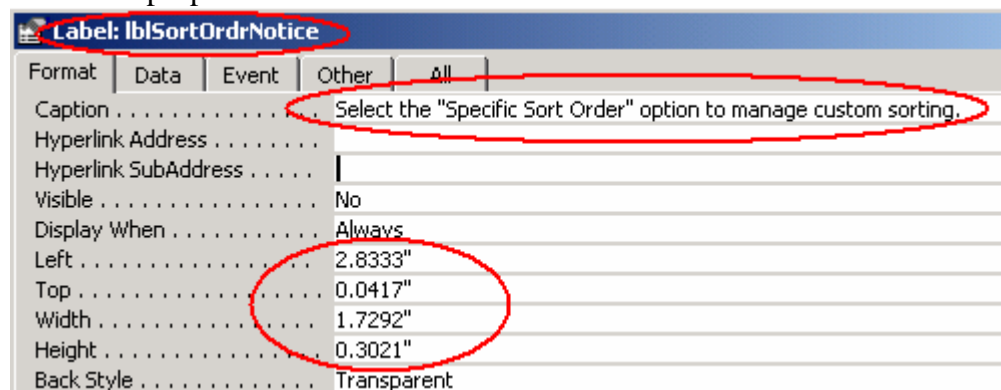
Properties for the "lblSelectListType" label control

On the “Other” tab of the “Properties” dialog box, set the “Name” property to: “lblSelectListType” and save your form. (Notice the “naming convention.”)

Add one more label control to the form. Click on the “Label” control button, . Move your cursor over the upper right of your form and click and drag to create the label control to look similar to the graphic below.




You will again notice that your cursor is now blinking in the new Label control. Immediately type “**Select the "Specific Sort Order" option to manage custom sorting.**” to add the caption for this control. When you have finished typing the caption, press the Tab or Enter key to complete you entry. Click on the new Label to select it and use the identified information on the “Format” tab of the “Properties” dialog box depicted below to set the properties.



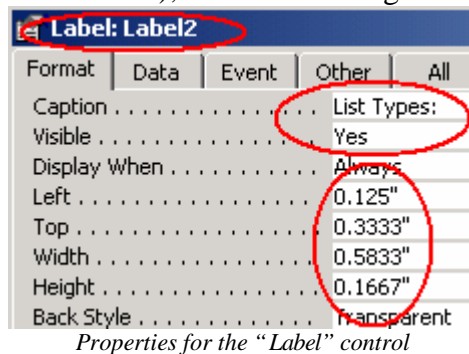
Properties for the "lblSortOrdereNotice" label control

On the “Other” tab of the “Properties” dialog box, set the “Name” property to: “lblSortOrdNotice” and save your form.

Next, we will add a combo box. This combo box will be used to display a unique list of values from the “List Types” field in the “tblListInfo” table. This will provide the user with the ability to select the “List type” that they want to work with. It will also serve as filter criteria for the list box that we will be adding shortly.

To add the combo box, click on the “combo box” button, , in the “toolbox” to select it and move your cursor over the form and using the graphic of the form we are building as shown on page 5 as a guide for where to place the combo box control, click in the general area where you want the combo box. The combo box along with an associated label control will be created. Do not be concerned with the exact size and/or location of the new combo box at this point, as we will set these values using the “properties” of the combo box and its associated label control very shortly. Just be aware that you can click on either of these controls and move them independently to the desired location without using the “properties” dialog box to literally set each property. If you would like, you can use this method to position your combo box and its associated label, but if you prefer, you can use the following graphic depictions of the values from my demo as the value for your controls.

With the Label control associated with your combo box selected (click directly on it to select it, not on the combo box control), set the following identified values for the label.

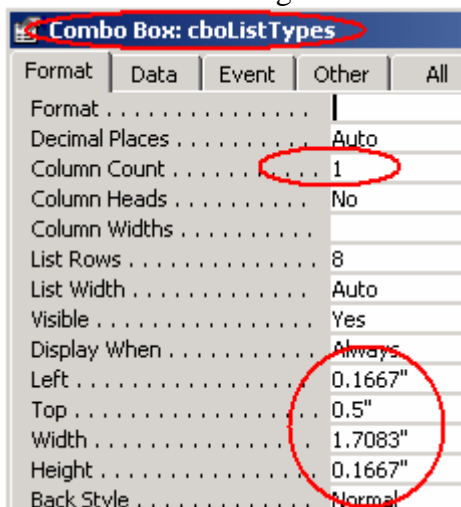


Please be aware that I did not change the “Name” property of this label type control. I just left the default name that was provided by Access. My reasoning for not strictly following naming convention standards in this case is that because this label control is associated with the combo box, I will most likely not be referring directly to this label control.

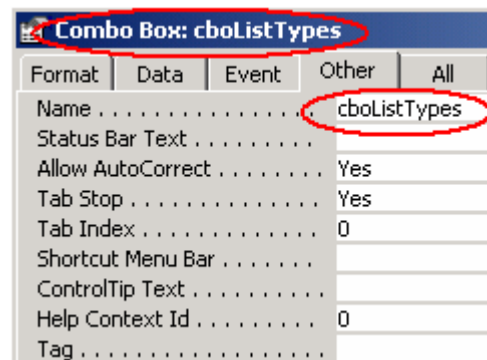
Note: Just for the record, a control like a combo box, a list box, a check box, or a frame or group control, for example, will, when created, have a label control that is created at the same time. This label control is considered to be “associated” with the control with which it was created. The concept is that any label associated with another control will inherit the same value for its “visible” property as the control with which it is “associated”. In other

words when you make a control that has an associated label not visible, that associated label will also become not visible. When you then make that control visible again, the associated label will also again be visible. This behavior is true as long as you do not specifically set the visible property of the “associated” label to false. In this case, even if you make the control visible, the “associated” label would remain not visible. These “associated” label type controls can be deleted without causing any issue what so ever. Also, if you have removed a label that was associated with another control and later find that you would like to again have a label associated with that control, you can create a label using the option from the “toolbox” and then select that label and delete it. Then immediately click on the other control to which you want the label to be “associated”, and press “Ctrl”-“V” to paste the deleted label control on to the form. Because you had selected another control prior to pasting the label control, this label will now be “associated” with the control that you had selected.

With the combo box control selected and the “Properties” dialog box displayed, select the “Format” tab and set the following identified values for the combo box, then select the “Other” tab and change the “Name” property as shown below to “cboListTypes”.

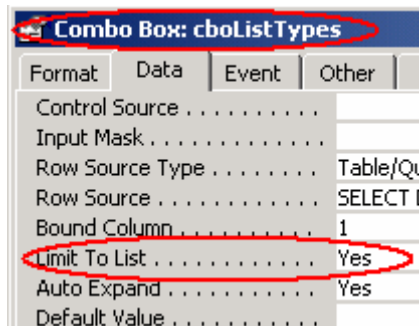


Properties for the Associated Label



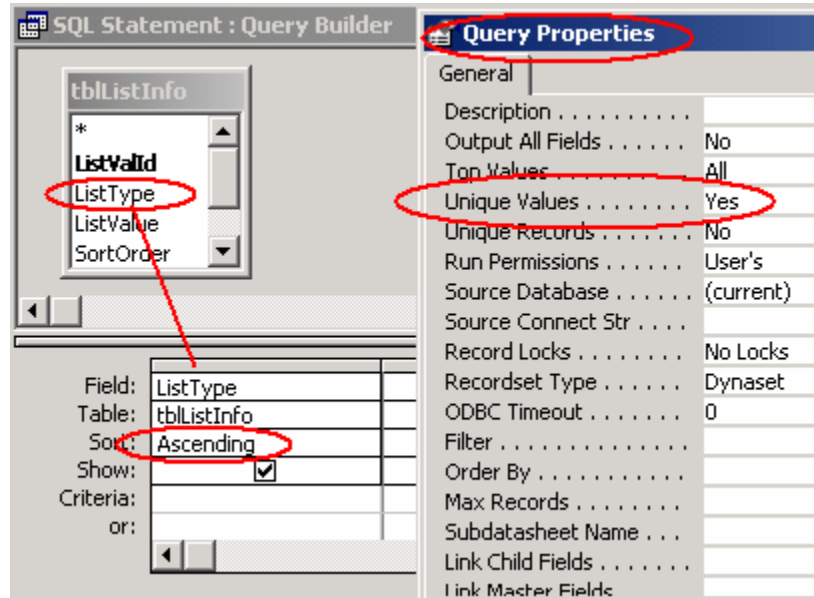
Properties for the “Other” tab of the “cboListTypes” combo box

Next, click on the “Data” tab of the “Properties” dialog box and set the “Limit to List” property to “Yes” as shown below.



Properties for the data tab of the “cboListTypes” combo box

and then click in the “Row Source” property row. Locate the button with the three dots, [...], at the right end of the property row and click it to display the Query Builder window. Use this area to define a query that will produce a unique list of the “List Types” available from the “ListType” field in the “tblListInfo” table.. Below is a screen capture of the query as it should be defined.



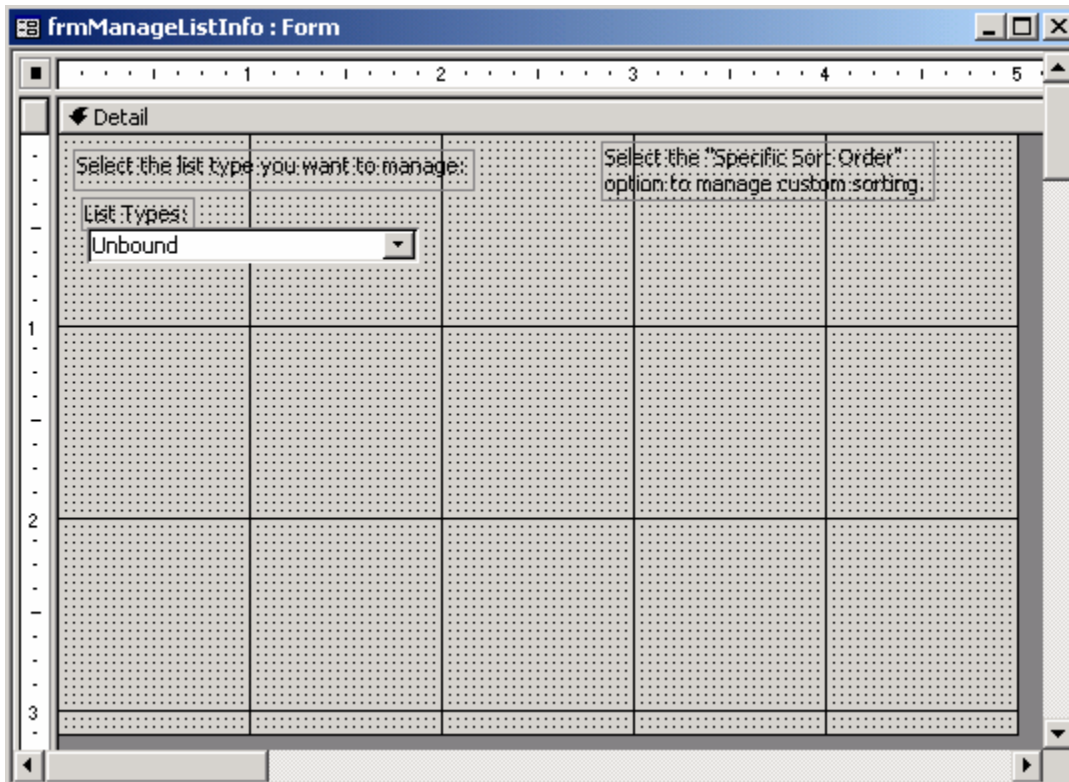
Query Builder and Properties of the Query

The graphic above shows Query Builder and the “Properties” dialog box for the query. To display the “Properties” dialog box for the query, right click anywhere in the Query Builder and select “Properties” from the shortcut menu.


The only field that you need to have in the query is the “ListType” field. Set the “Sort” order to “Ascending” for this field. The only property that you need to set for the query is the “Unique Values” property. Set this property to “Yes”. This will cause the query to only show one instance of each value in the “ListType” field.

With the combo box created and these properties set, we need to again save our form.

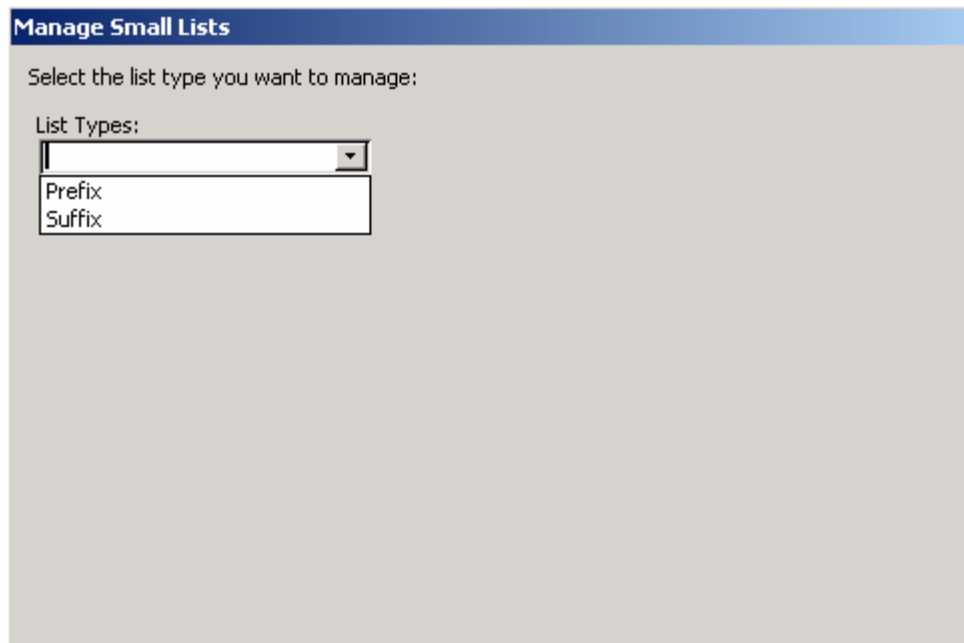
Below is a screen capture of how our form should now appear when viewed in “Design View”.



View of the "frmManageListInfo" from in "Design View" mode

Now we can actually take a look at just how this form will look to the user and how the combo box will work. To view the form, click the "Form View" button. , on the "Form Design Menu" bar.

Below is screen capture of how our form should now appear when viewed in "Form View" with the combo box dropped down. The list should only display the two "List Types", "Prefix" and "Suffix".




View of the "frmManageListInfo" from in "Form View" mode

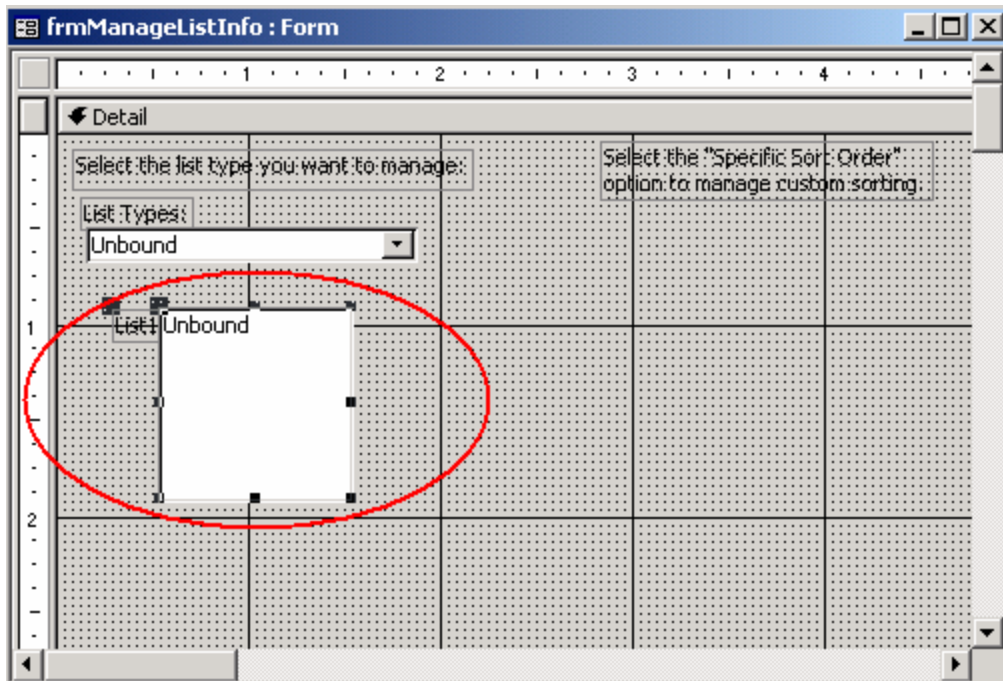
We will now continue to add controls to our form to add functionality for the user.

If your form is still being displayed in "Form View", click the "Design View" button,



, on the "Form Design" menu bar to switch the view to "Design View".

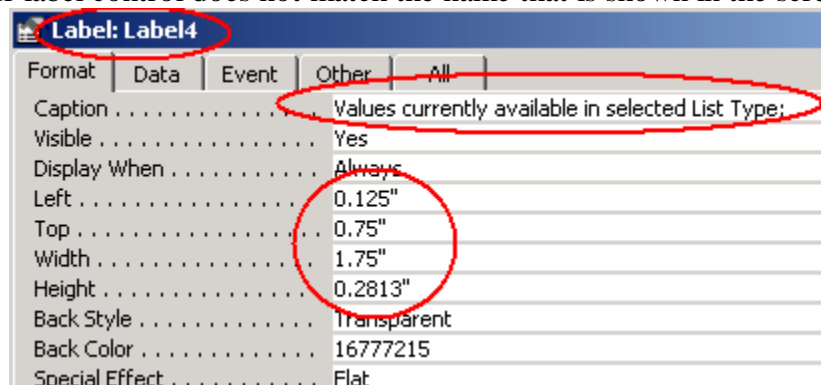
We will now add a "List box" control. Locate the "List box" tool, , from the "Toolbox" and select it by clicking on it. Move your cursor over your form and in an area just below the "combo box" that you previously placed on your form, click to place the list box. A list box as inserted by Access is not very useful, but you as the developer have many options for turning this control into a very useful and helpful tool. See the graphic below for how a list box control that has just been placed on a form might look.



New form with a new List box inserted

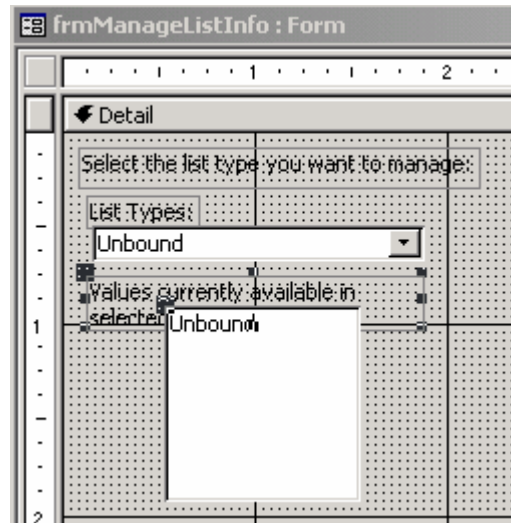
With the “List box” placed on the form, you are now ready to change or reconfigure it to be more like we need it to be. You can use the mouse to drag the associated label to the correct relative location and to resize the list box to your liking, or you can simply use the following values depicted in the next graphic and change the various properties of the label and the list box controls, thus changing its look and functionality.

Again, to change property values for any object, that object must be currently selected. With that in mind, select the label that is associated with the new List Box. Using the graphic below, modify the identified values for the label. Please note that I have not changed the “Name” value that Access assigned to this label control. Do not worry if the name of your label control does not match the name that is shown in the screen captures.



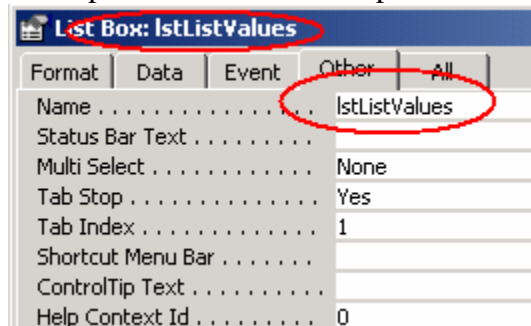
Format tab for label control associated with the list box

After setting these values for the label associated with the List box, your new form might look like the one in the graphic below.



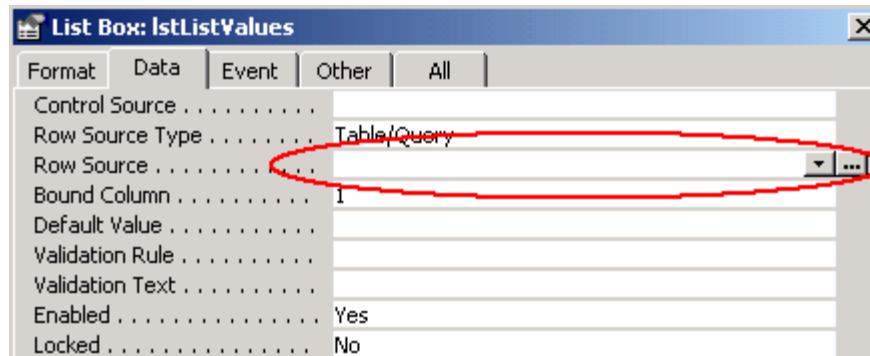
Label with properties set

Next, select the list box control by clicking on it. As you learn to work with various controls, you will begin to develop specific methods for the way that you configure controls. For example, I will usually select the “Other” tab from the Properties dialog box and change the default Name property to a Name that conforms to my naming convention standards and that provides a basic description of the control.




The “Other” tab properties for the list box

The next thing that I normally do when working with a list box or a combo box control is to get the control to acquire the correct data and to display that data in an appropriate manner. First, we need to define the data that is going to be displayed in this list box. Click on the “Data” tab of the “Properties” dialog box and place your cursor in the “Row Source” property text box. Two buttons will appear at the right end of this text box as show below.



Data tab for List box

Click the button with the three dots, , to display the QBE Access form. You will use this tool to design the query that will produce the records that will be displayed in your list box. When the button is clicked, the following dialog box will be presented.

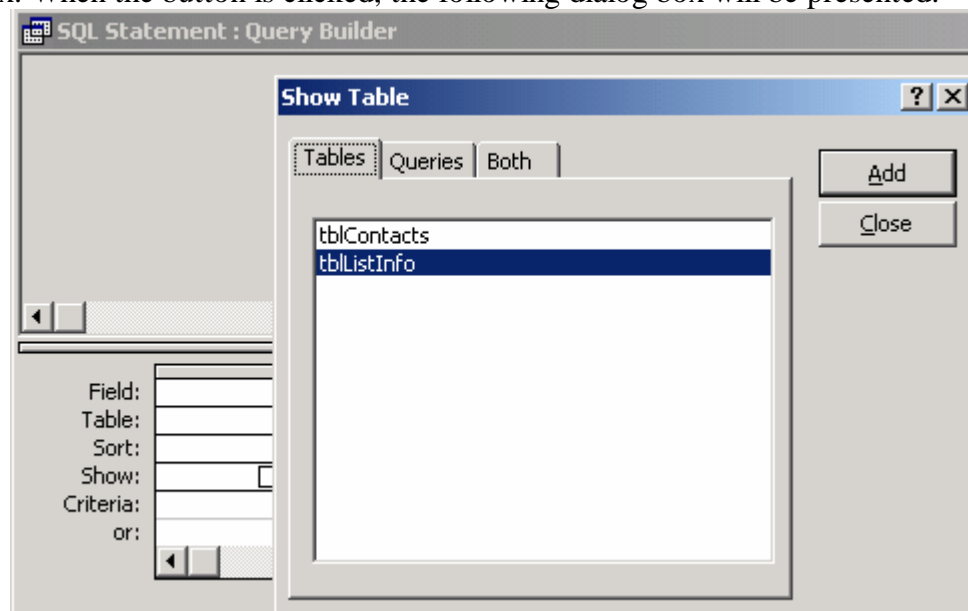
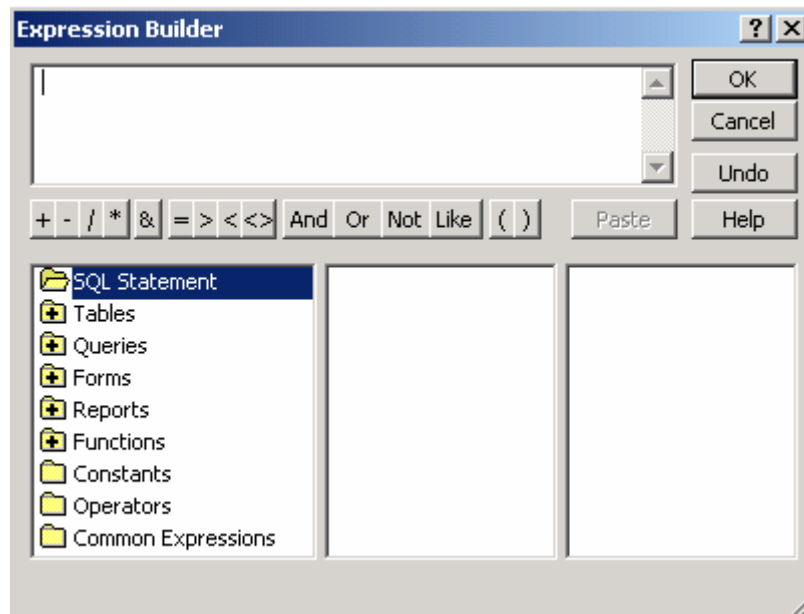


Table selection options for QBE

From this screen you need to select the table from which the data to be displayed in the list box is going to come. Select the “tblListInfo” table and click the “Add” button. Then click the “Close” button.

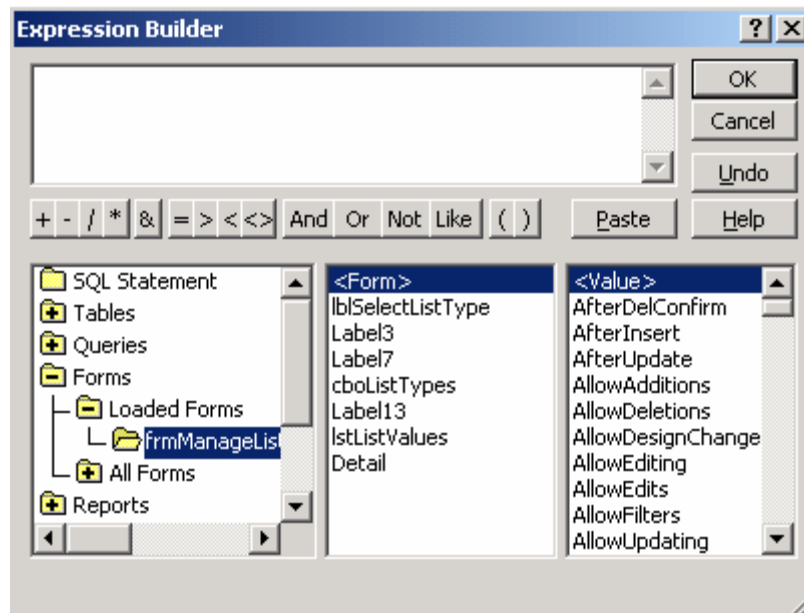
Below is the completed version of the query as built with the QBE. Place the “ListValue”, “SortOrder” and “ListType” fields in the QBE by either selecting each one from the table in the upper area of the QBE and drag it to the desired column location, or by clicking in the “Field” row of each column in the QBE and selecting the desired field. (You can also double click a field in the table in the upper area of the QBE and it will automatically be added to the query in the next available column.) The only other major thing that must be done is to tell the query where to get the value that will serve as the filter criteria to restrict the records for the “List Values” that are to be displayed in our list box, based on the selection made from the “List types” combo box. We must tell the query to use the value selected in the “List types” combo box as the criteria for the query.

To accomplish this in the simplest way, place your cursor in the “Criteria” row in the “ListType” column and right-click to display the shortcut menu. Select the “Build ...” option from this menu. The “Expression Builder” dialog box will be presented as shown below.



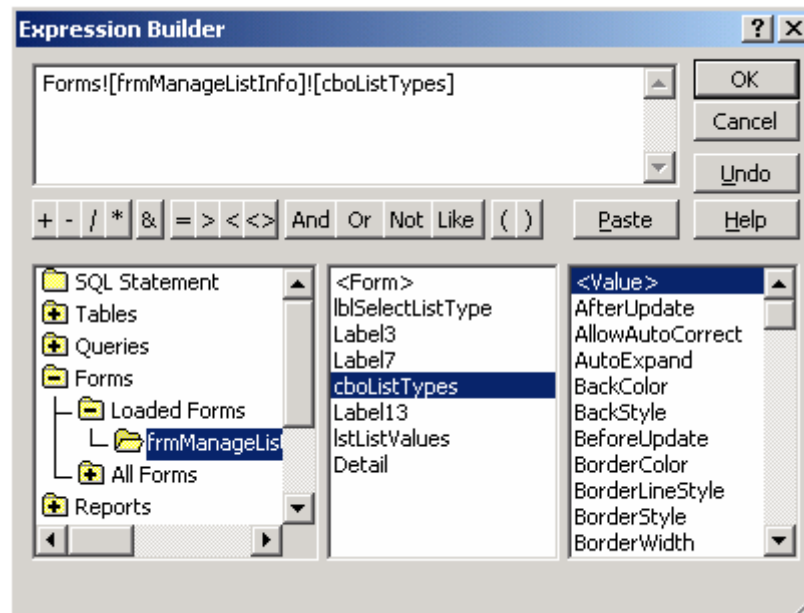
Expression Builder

You can now use the options in this dialog box to help you identify the criteria for the query. Because we want to use a value from a specific control on a form, double click on the “Forms” option in the first list box. This will cause two options to be presented, “Loaded Forms” and “All Forms”. Because the form from which we want our value to be selected is already open, double click on the “Loaded Forms” option. A list of all loaded forms will be displayed. In this case, you may only see the current form you are working on. *Please note that you may not be able to read the complete name of the forms as displayed in this list. It may be necessary to widen the list. To do so, move your cursor to the right side of the “Expression Builder” and click and drag it wider. As you drag it wider, each of the three lists will become wider.* Next, select your form from the list shown below the “Loaded Forms” option. The list of controls available on the selected form will be displayed in the middle list box of the “Expression Builder”, See graphic below.



Expression Builder with form selected

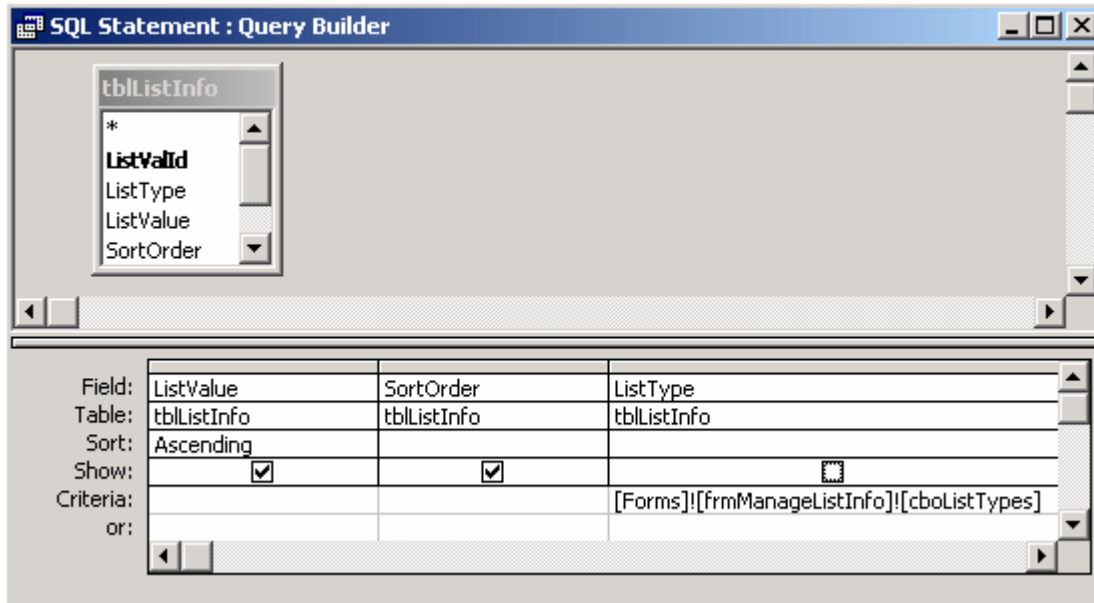
Locate the “List Types” combo box, “cboListTypes” from the list of controls. Please notice that if we are using a fairly consistent and descriptive naming convention to name our controls, it is much easier to locate and select the correct control. When you have located and selected the “cboListTypes” control from the list, click the “Paste” button. This will add the correct expression to the large text box at the top of the “Expression Builder” as shown below.



Expression Builder with expression pasted

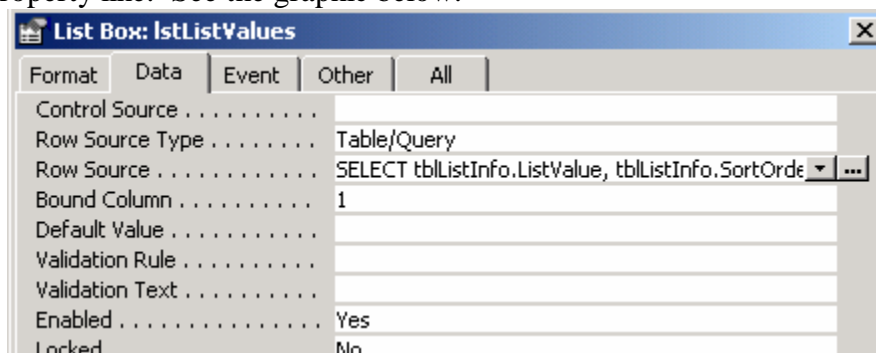
Next, click the “OK” button to accept the expression and close the “Expression Builder” dialog box. The expression that was pasted in the “Expression Builder” will now be added as the criteria for our query. In this case, we are telling the query to only return rows from the “tblListInfo” table where the “ListType” is equal to the value currently

selected in the “List Types” combo box. Uncheck the “Show” check box in the “ListType” column to indicate that we do not want to display this data. We are only using this column as criteria for filtering the records that will be returned by the query. With these steps completed, the Query Builder should look very much like the graphic below.



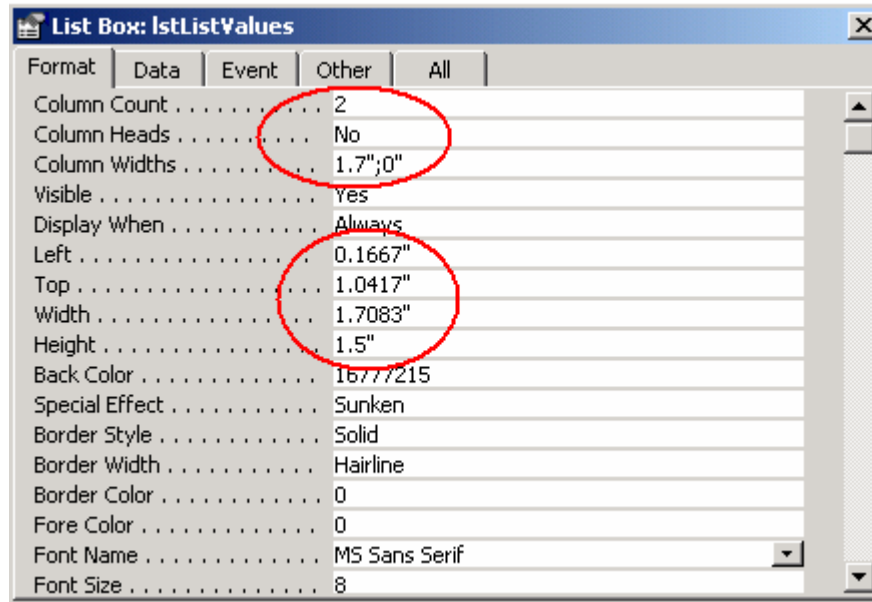
Query Builder with criteria expression added

Click the “X” at the upper right of the QBE window to close it. When prompted, choose to “Save” your changes. The “Properties” dialog box will now be available again and will now have the SQL statement created with the QBE has been added to the “Row Source” property line. See the graphic below.



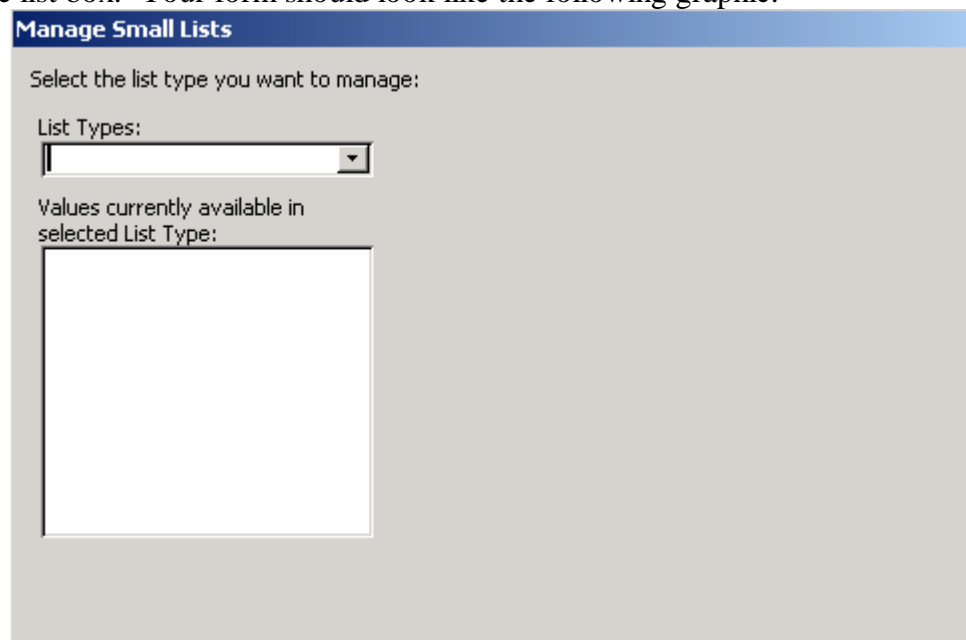
Sql statement added to Row Source Property

We are now ready to make a few more modifications to some of the Properties of our List box. Use the information from the following screen capture to set the various identified properties for the “Format” tab of your list box.




Format tab of the Properties for the List box

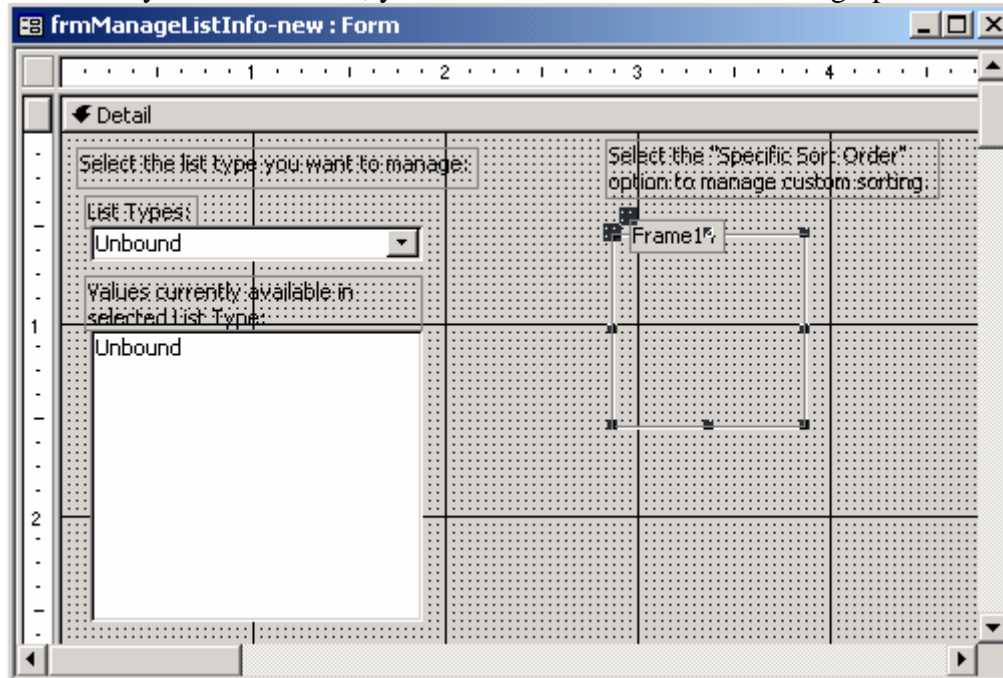
You should now be able to open your new form in “Form View” and see the combo box and the list box. Your form should look like the following graphic.



If you select a “List Type” from the combo box, the list box you will see that there is still no list displayed in the list box. This is because we do not have any method for telling the list box that we have made a selection from the combo box. This functionality will be added when we start to add VBA code to our project.

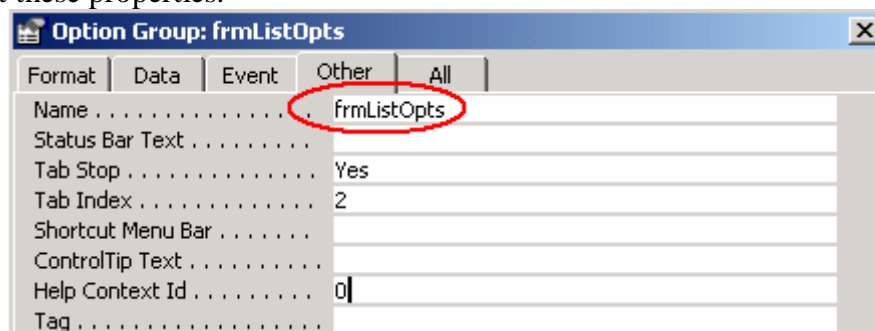
Next, we will add an “Option Group” control. This control will allow our users to determine if they want to see the “List Values” list box sorted in “Alphabetical” sort order, sorted by the values, or if they want to see it sorted in a user definable sort order.

To add the “Option Group” control, select this option from the “Toolbox” by clicking on the “Option Group” tool, , and move your cursor to an area just below the existing label in the upper right of your form and click to add the control. You will not that when you have added your new control, your form should now resemble the graphic below.



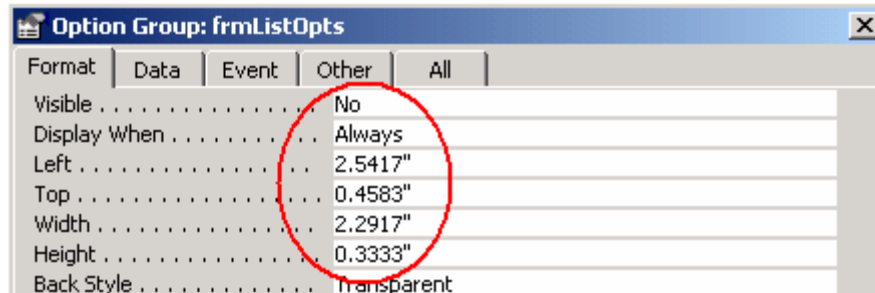
Form with “Group Option” control added

You will also notice that this new control is identified as a “Frame” control. Actually this type of control is normally referred to as either a “Group” or “Frame” control. Please notice that when we added the “Option Group” control, a “frame” and a “label” associated with the frame were both added to the form. We will be setting properties for both of these controls. First, make sure that the “frame” part of the control is selected by clicking on the outline of the control. Display the Properties dialog box and using the graphic below, set the identified properties for the “Frame” control. Select the “Other” tab and set these properties.



The “Other” tab for the “Group” control

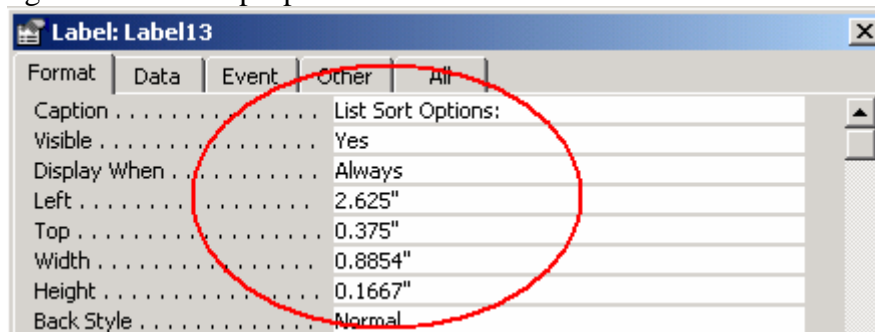
Select the “Format” tab and make these changes to the properties.



The "Format" tab for the "Group" control

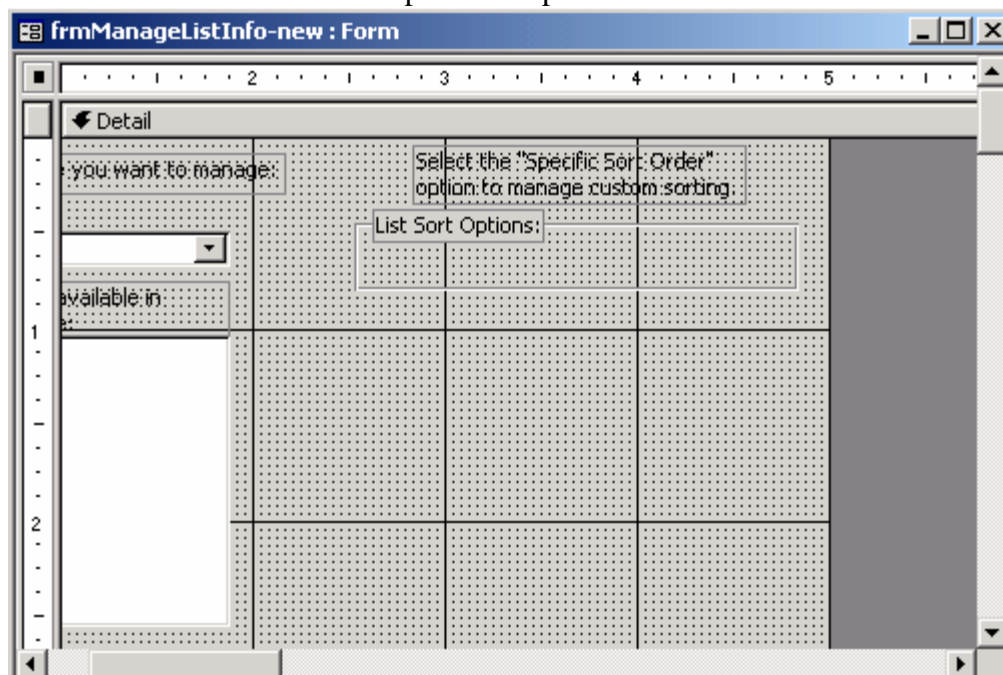
Select the "Data" tab and locate the "Default Value" property and enter a "1" (one) for this property.

Next, select the Label that is associated with the "Option Group" control, and click on the "Format" tab of the Properties dialog box. Using the information from the graphic below, change the identified properties of the Label control.




The "Format" tab of the properties dialog box for the Label

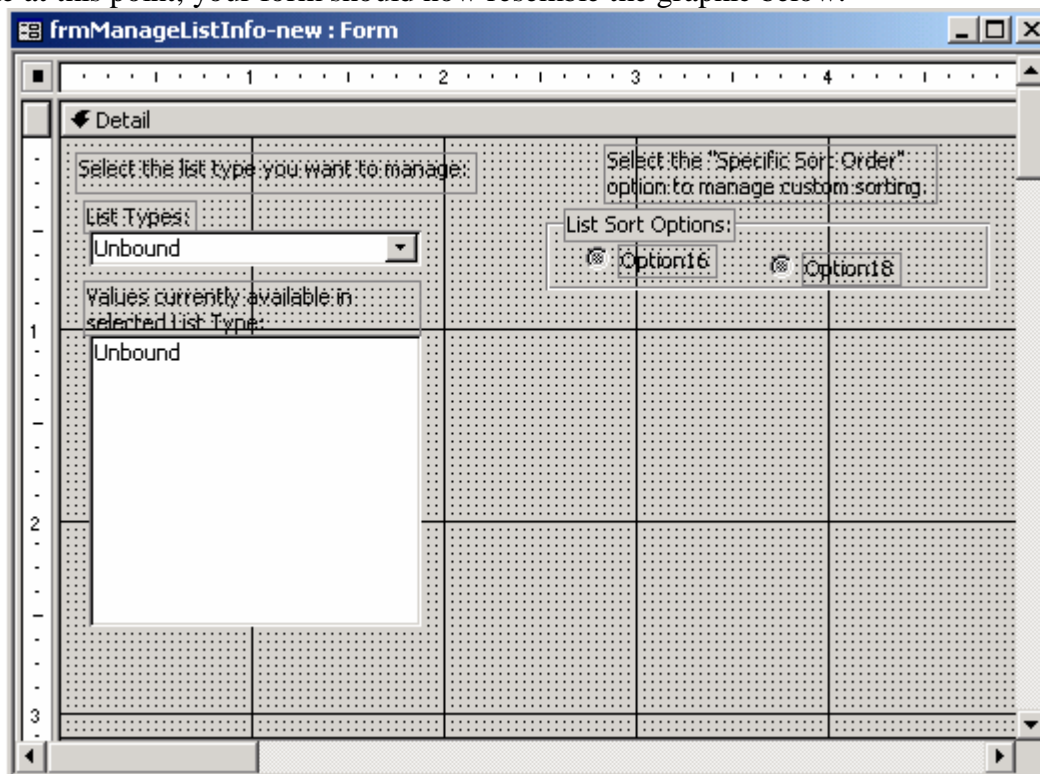
This is another case where the associated label has not been renamed. Below is a screen capture of the form with the new "Option Group" control added.



Now that we have our "Option Group" control in place, we need to add some options buttons to it. Yes, that is right, will now be adding controls to a control. This is really not that out of the ordinary with Access. In fact, it is quite commonplace. If you ever have the need to use a "Tab" control, you will really understand adding controls to a control.

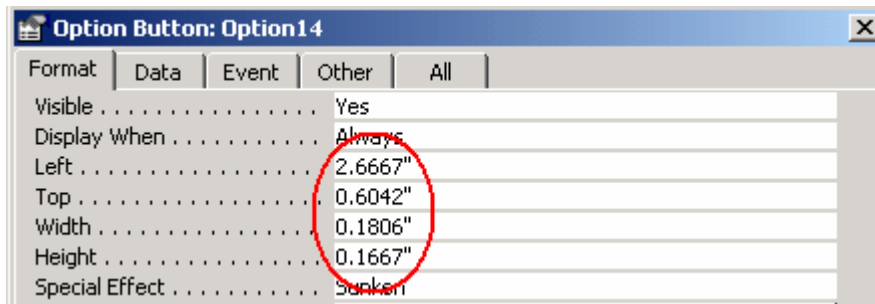
Ok, let's add a couple of "Option Buttons" to the "Option Group" control. In the "Toolbox", locate and double-click the "Option Buttons" tool, . When you double-click a tool in the "Toolbox", you then have the capability of adding more than one of the same type of control to your form without having to again click on the tool in the "Toolbar". Move your cursor over the area within the "Option Group" control that we just added and configured. When you move your cursor into this area, the interior of the "Option Group" control the interior of that control will turn very dark, indicating that you are in the area where you can add the "Option Button" controls. Click once near the inside left of the "Option Group" control to add one "Option Button" control. Then move your cursor over to the right, staying within the "Option Group" control and click again to add a second "Option Button" control. If necessary, you could just continue adding "Option Button" controls to the "Option Group" control, but in this case we will only need two options. To stop adding "Option Button" controls, go to the "Toolbox" and again click the "Option Button" tool to unselect it.

Although the actual placement of your "Option Button" controls may not be exactly like mine at this point, your form should now resemble the graphic below.



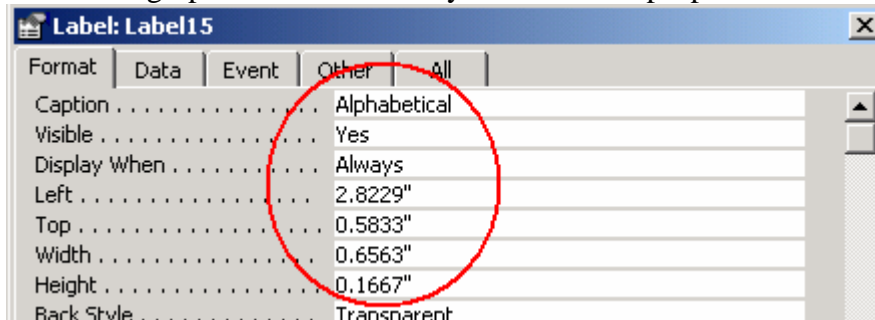
Form after adding two "Option Button" controls to the "Option Group" control

Notice that each of the “Option Button” controls that we added to the “Option Group” control have a “button” and an associated “label”. The combination of these make up the “Option Button” control. We will now need to configure both of the “Option Button” controls by modifying the properties for the “button” and the properties for the “label” for each of the controls. First, select the “Option Button” control on the left by clicking on it and then use the information from the graphics below to modify the identified properties for it.



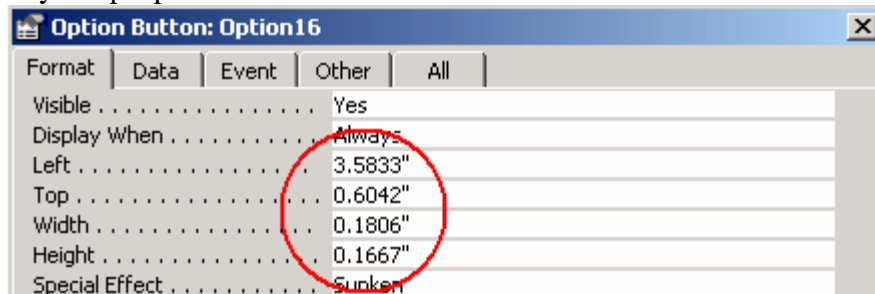
Format tab for the left Option Button

Next select the label associated with this “Option Button” by clicking on it. Use the information from the graphic below to modify the identified properties for this label.



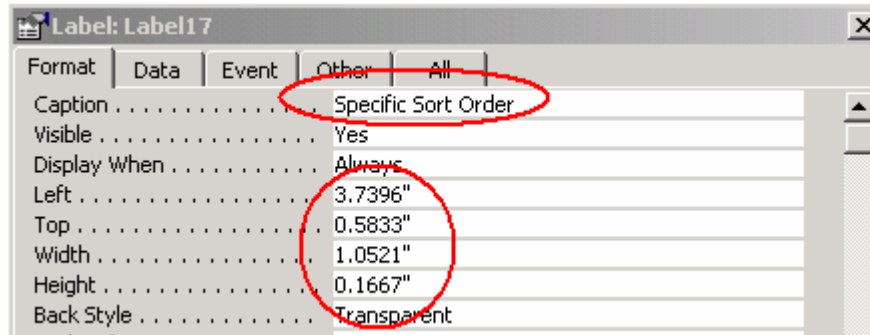
Format tab of properties for the label

We will now set the properties for the “Option Button” on the right in the “Options Group” control. First, select the “button” and using the identified values in the graphic below, modify the properties for this control.



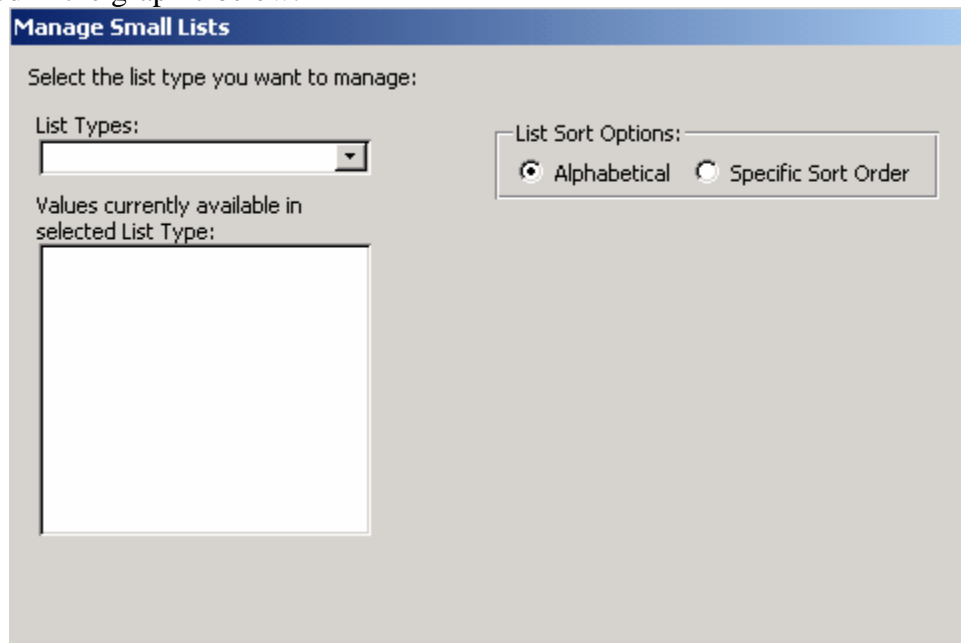
Format tab of properties for the right “Option Button” control

Next select the label associated with this “Option Button” by clicking on it. Use the information from the graphic below to modify the identified properties for this label.



Format tab of properties for the label

At this point, if you view your form in “Form View” it should appear like the one depicted in the graphic below.



Form with “Group Option” and “Option Button” controls added

Save your changes to your form.

The next step is to add a “Text box” control to our form. This text box control will be used when the user needs to add and new value to a list or to edit an existing list value.

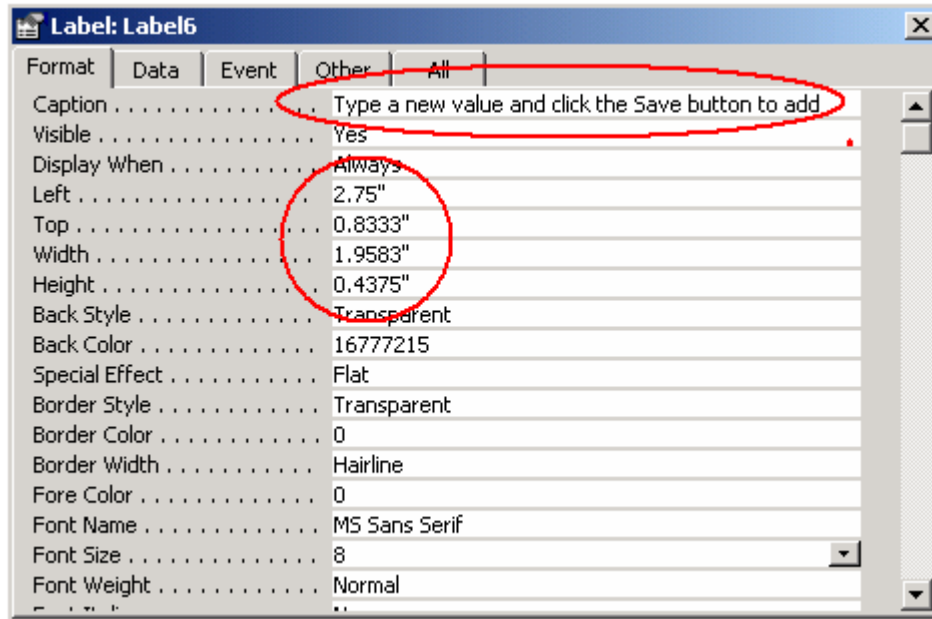
To add this new “Text box” control, first view your form in “Design View”. Then click on the “Text box” tool, **abl**, in the “Toolbox”. With the “Text box” tool activated, move your cursor over your form to an area below the “Option Group” control that we just completed, and click to add the new control. Your form should now resemble the one pictured below.

Form with new “Text box” control just added

Note that when the new control was added, it also included the associated label control. Do not be concerned, at this point, about the exact location of the control or its associated label control. We are about to modify the properties of these controls that will include the actual location and size of each control.

Let’s first make our modifications to the associated label for this new text box control. Please note that the default name that was provided for this new control has not been changed. Do not be concerned if the default name of this label control on your form is not the same as show in the demo.

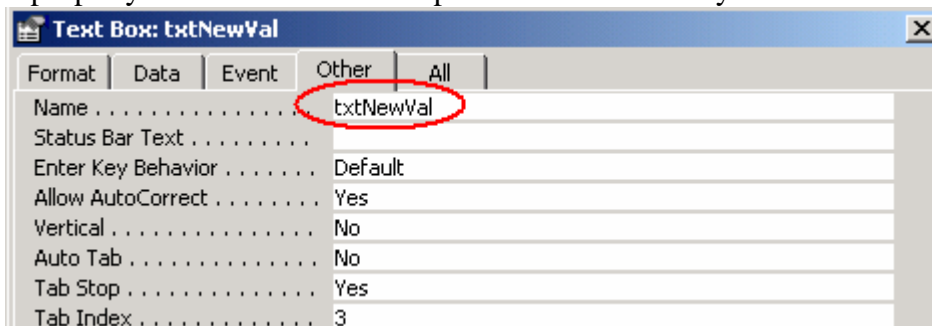
As before, select the label by click on it. With the “Properties” dialog box displayed for the label, click on the “Format” tab and use the value identified in the graphic below to make the necessary modifications to the properties of the label control.



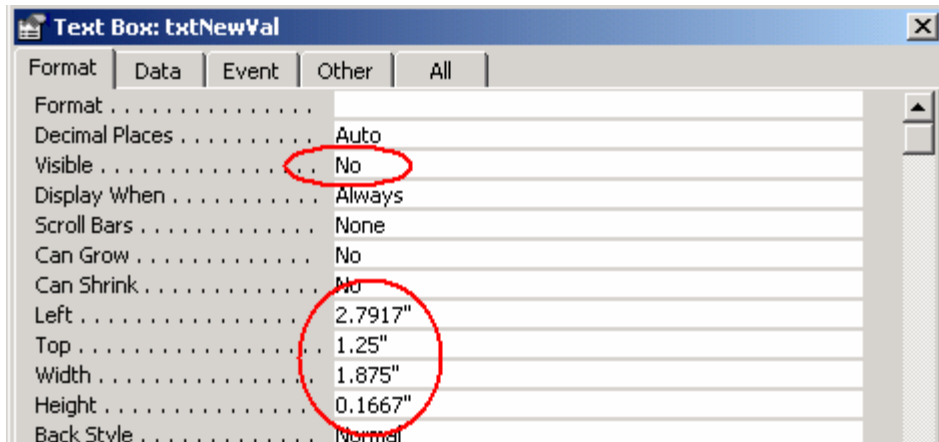
Format tab of properties for label control associated with the "txtNewVal" text box control

Be aware that the entire string value for the "Caption" property of this control is not entirely visible in the graphic above. Please add the following string as the "Caption" property: "Type a new value and click the Save button to add a new value to the selected list:". This is an exceptionally long caption. However, it does serve to demonstrate that you do have flexibility when it comes to using labels.

Next, we will modify the properties of the new text box control. Again, select the control before attempting to make any changes to any of its properties. With the new text box selected, and the "Properties" dialog box displayed, click on the "Other" tab and use the identified property values in the screen captures below to modify this control.

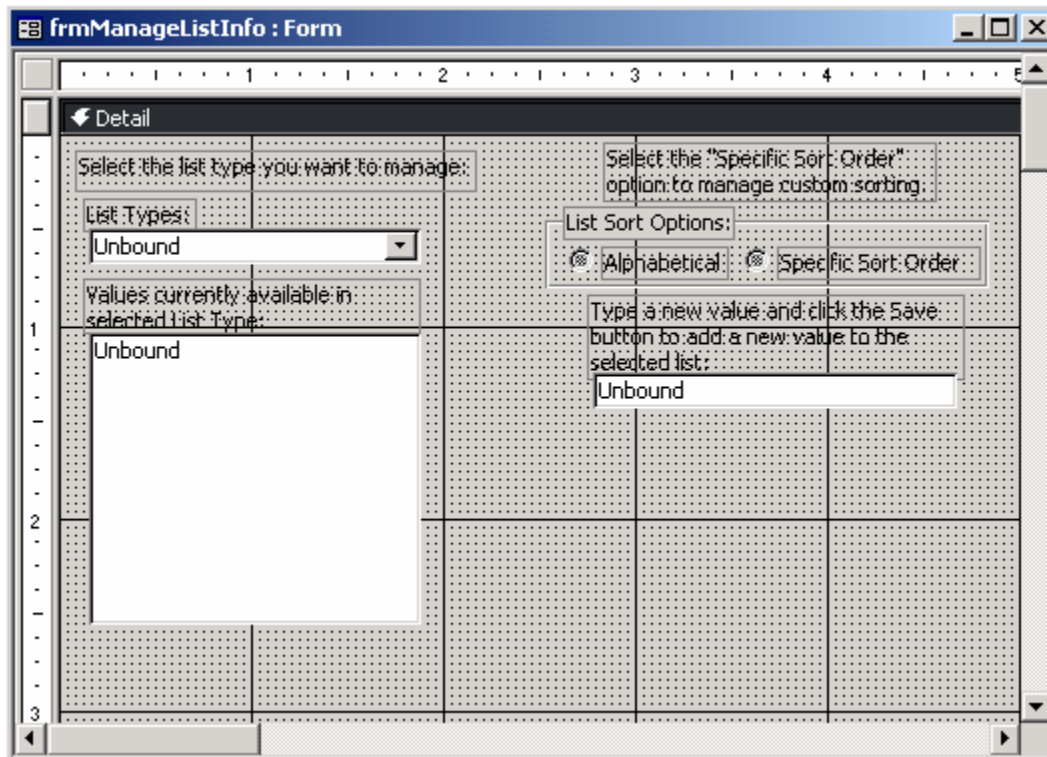


"Other" tab of properties for the "txtNewVal" text box



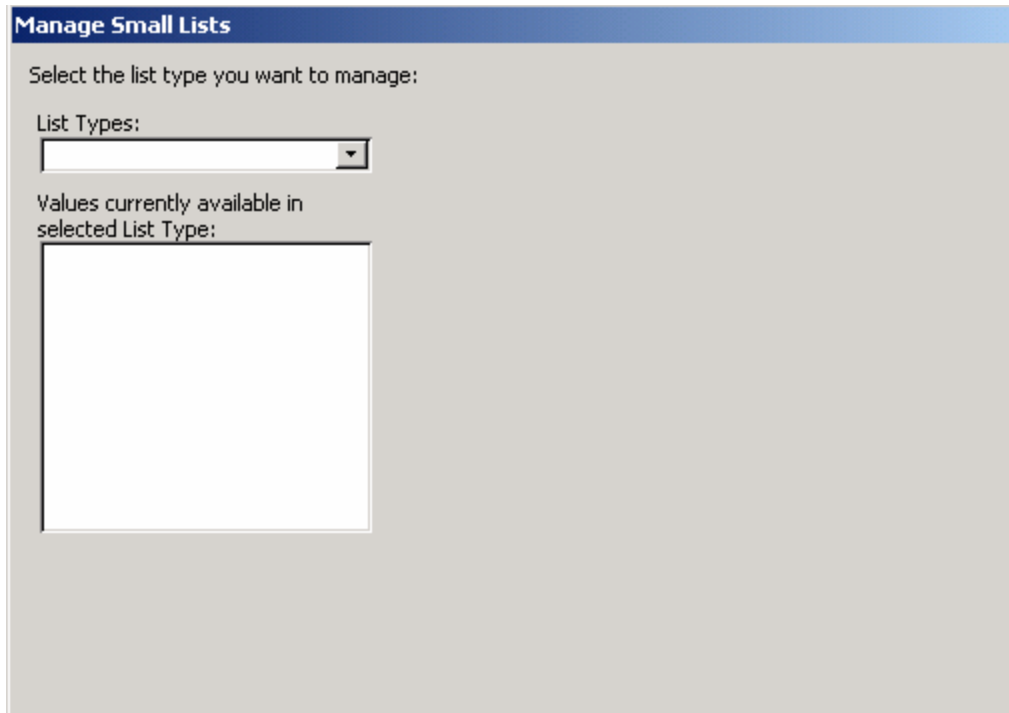
"Format" tab of properties for the "txtNewVal" text box

Below is a graphical representation of how your form should now appear in "Design View".



Form in "Design View" after the "txtNewVal" text box has been added

Below is the depiction of this same form, as it would appear at this time in "Form View", provided you have correctly set all of the properties for all controls that have been added to the form.




Form in “Form View” as it should appear at this point in the demo

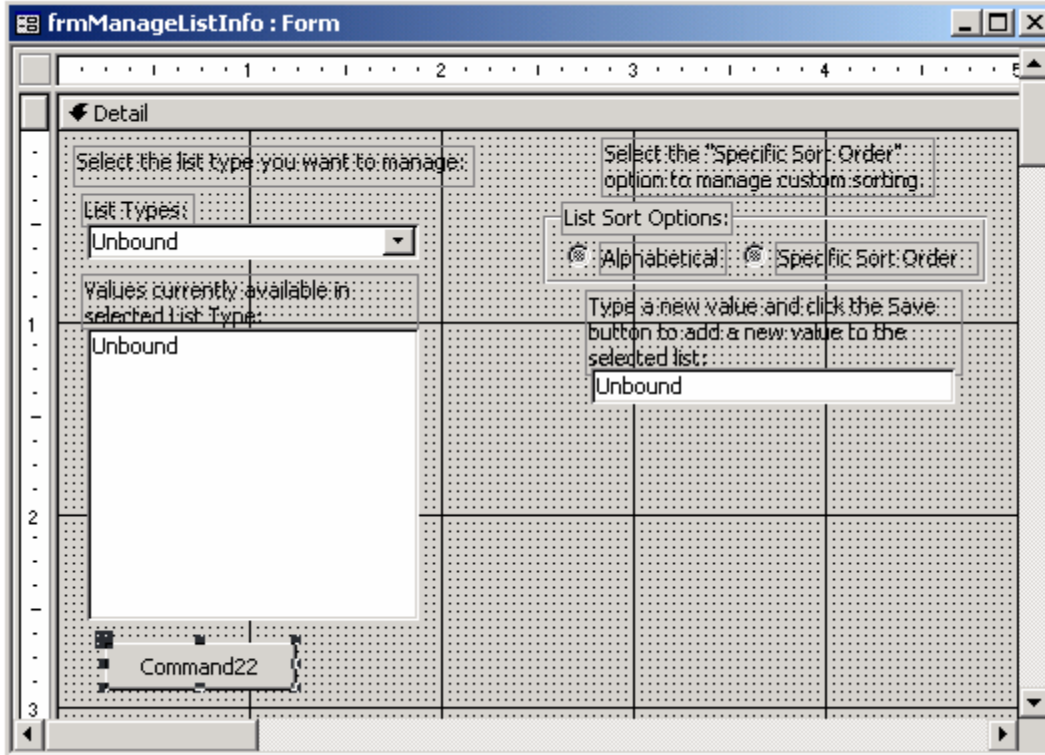
At this point, please notice that when viewed in “Design View” all of the controls that we had added to this point are visible. However, when viewed in “Form View”, which is the normal view that users will have of our form, several of the controls that we have already added are not visible. This is as it should be. The ability to have various controls exist on our forms but not be available to users is a typical method for the development of a good user interface. The controls that are not available right now will be utilized later and will become visible and usable when we start adding VBA coding later in this demo.

Note: The ability to flag values as Active or Inactive was not utilized in this demo, however, this functionality can be added by adding a check box control and some additional VBA code.

With your form again in “Design View” we will now add eight different “Command Buttons” to our form. These command buttons will be used to allow users to interact with our form to indicate that they want to perform specific tasks. Some of these buttons will be immediately visible while some will not initially be visible but will be made visible through the user of VBA code as the user uses the various options on our form. Some controls will be visible and will be “enabled” while some will be visible but not enabled. (When a button, or any object for that matter, is enabled it is available to the user for some action. When it is “disabled” (has it’s “enabled” property set to “No”) it is not available for user interaction. The availability of these controls for user interaction, can and will be managed through the use of VBA code.

If you have not done so, please save your changes to your form at this time.

Now, let's add our first "Command Button" to our form. Locate the "Command Button" tool, , in the "Toolbox" and click it to activate it. Move your cursor over you form to a location just below and to the left side of the "lstListValues" list box and click to place the command button on the form. When you have added the command button to the form, the form should now look very much like the graphic depiction below.



The screenshot shows a Windows form titled "frmManageListInfo : Form". The form has a grid-like layout with several controls. On the left, there is a "List Types" dropdown menu currently set to "Unbound". Below it is a text box labeled "Values currently available in selected List Type:" containing the text "Unbound". To the right of this, there is a "List Sort Options" section with two radio buttons: "Alphabetical" (selected) and "Specific Sort Order". Below the sort options is a text box labeled "Type a new value and click the Save button to add a new value to the selected list:" containing the text "Unbound". At the bottom left of the form, a new command button labeled "Command22" has been added. The form has a standard Windows interface with a title bar, menu bar, and scrollbars.

From after first command button has been added

When adding these buttons, do not initially be concerned with the exact size or placement of each button. In fact, you could just add the button any where on the form and then make the necessary modifications to the properties for each button and this would determine the size, placement and caption (wording that appears on the button) for each button. The button we just added will become the "Add" button on our form. This button will provide users with the ability to add a new List Value to the list of values for the currently selected List Type. Please use the following list of properties and modify the properties of this new button.

Properties for the "cmdAdd" command button:

Other Tab

Property

Value

Name

cmdAdd

Control Tip Text

Click to add the new value to the selected list

Data Tab

Enabled

No

Format Tab

Caption

&Add (include the ampersand)

Visible

Yes

Left	0.1667"
Top	2.5833"
Width	0.5"
Height	0.2083"

Save your work.

Next, add another command button. This button will be come the “Edit” button. This button will provide users with the capability of making changes to the currently selected List Value. Use the following list and modify the properties for this button.

Properties for the “cmdEdit” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdEdit
Control Tip Text	Edit the selected item

Data Tab

Enabled	No
---------	----

Format Tab

Caption	&Edit (include the ampersand)
Visible	Yes
Left	0.7917"
Top	2.5833"
Width	0.5"
Height	0.2083"

Save your work.

Add another command button which will be come the “Delete” button. This button will provide users with the ability to remove the currently selected value from the list.

Properties for the “cmdRemove” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdRemove
Control Tip Text	Remove selected item from list

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	No

Format Tab

<u>Property</u>	<u>Value</u>
Caption	&Delete (include the ampersand)
Visible	Yes
Left	1.375"
Top	2.5833"
Width	0.5"

Height	0.2083"
--------	---------

Save your work.

Add a command button that will become the “Close” button. Users will use this button to close the current form.

Properties for the “cmdOk” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdOk
Control Tip Text	Close this form

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	Yes

Format Tab

<u>Property</u>	<u>Value</u>
Caption	Cl&ose (include the ampersand)
Visible	Yes
Left	4.375"
Top	2.8333"
Width	0.5833"
Height	0.25"

Save your work.

Add a command button that will be come the “Save” button. This button will allow users to save any changes made to the currently selected record or to save any new record that is being added to the current list of values.

Properties for the “cmdSave” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdSave
Control Tip Text	Click to save the changes to the selected item

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	No

Format Tab

<u>Property</u>	<u>Value</u>
Caption	&Save (include the ampersand)
Visible	No
Left	3.2083"
Top	1.4583"
Width	0.5"
Height	0.2083"

Save your work.

Add another command button. This button will be come the “Cancel” button. Users can use this button to cancel the editing of an existing record or the adding of a new value.

Properties for the “cmdCancel” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdCancel
Control Tip Text	Click to undo any changes to this item

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	Yes

Format Tab

<u>Property</u>	<u>Value</u>
Caption	&Cancel (include the ampersand)
Visible	No
Left	3.7917"
Top	1.4583"
Width	0.5"
Height	0.2083"

Save your work.

Add a new command button to the form. This button will become the “Up” button and, when available, will be used to cause the currently selected item from the List Values list box to be moved up in sort order.

For this button, we will now add a “Picture” to identify the button to users. Please refer to the “Add a Picture to a Button” topic below and use the instructions for the “cmdUp” button to add the picture to this control.

Properties for the “cmdUp” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdUp
Control Tip Text	Click to move selected item up in list

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	No

Format Tab

<u>Property</u>	<u>Value</u>
Caption	(leave blank)
Picture	(bitmap) see “Adding Picture to Button” below
Visible	No

Left	1.9583"
Top	1.7917"
Width	0.2542"
Height	0.3375"

Save your work.

Add a new command button to the form. This button will become the “Down” button and, when available, will be used to cause the currently selected item from the List Values list box to be moved down in sort order.

Again, we will add a “Picture” to this button to identify it to users. Please refer to the “Add a Picture to a Button” topic below and use the instructions for the “cmdDown” button to add the picture to this control.

Properties for the “cmdDown” command button:

Other Tab

<u>Property</u>	<u>Value</u>
Name	cmdDown
Control Tip Text	Click to move selected item down in list

Data Tab

<u>Property</u>	<u>Value</u>
Enabled	No

Format Tab

<u>Property</u>	<u>Value</u>
Caption	(leave blank)
Picture	(bitmap) see “Add a Picture to a Button” below
Visible	No
Left	1.9583"
Top	2.2083"
Width	0.2542"
Height	0.3375"

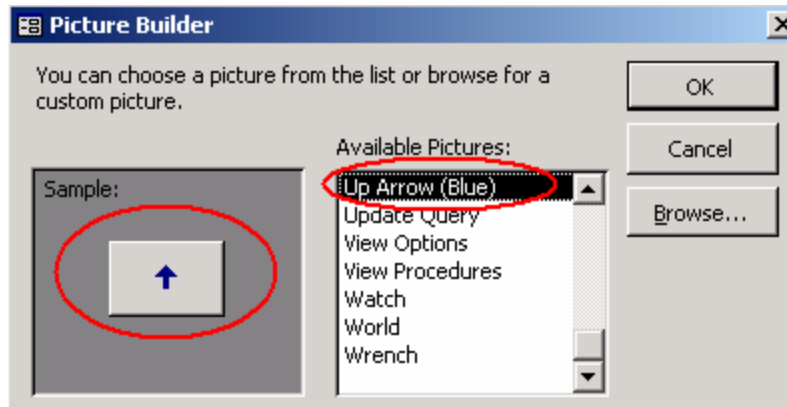
Save your work.

Add a Picture to a Button

We now need to add the up or down arrows respectively to each of our last two buttons.

Adding the picture to the “cmdUp” button

To accomplish this, first select the “cmdUp” button. With the “Properties” dialog box displayed, locate the “Picture” property on the “Format” tab and click in this property row. A small button with 3 dots will be displayed at the right end of this property row. Click this button to display “Picture Builder” dialog box. See Graphic below.




Picture Builder with “Up Arrow (Blue)” selected

Scroll down in the “Available Pictures” list box until you see the “Up Arrow (Blue)” item. Click to select this picture. It will be displayed in the “Sample” area to the right of the list. Click the “OK” button to add this picture to the selected button.

Adding the picture to the “cmdDown” button

Next, select the “cmdDown” button and repeat the process for selecting a picture. This time, scroll down until you see the “Down Arrow (Blue)” item in the list. Click to select this picture. It will be displayed in the “Sample” area to the right of the list. Click the “OK” button to add this picture to the selected button.

At last, we only need to add one more control to our new form. We will add a new

“Label” control. Locate the “Label” tool, , from the “Toolbox” and click to activate it. With your form in “Design View”, move your cursor over your form to an area just to the right of the up and down arrows and below the “Save” and “Cancel” button and click to add the new label control. (Note: that if you do not provide any text for the caption of a new label control, that new label control will disappear so you need to immediately add at least a space or some character to the label to hold it on the form.) Immediately type the following statement into the new Label control:

To Change Sort Order: Select an item form the "Values List" and then use these buttons to move that item to the desired position in the list.

Change or set the following properties for this new Label control.

Properties for the “lblExplainRtMgr” label control:

Other Tab

Property

Name

Value

lblExplainRtMgr

Format Tab

Property

Caption

Visible

Value

See statement provided above

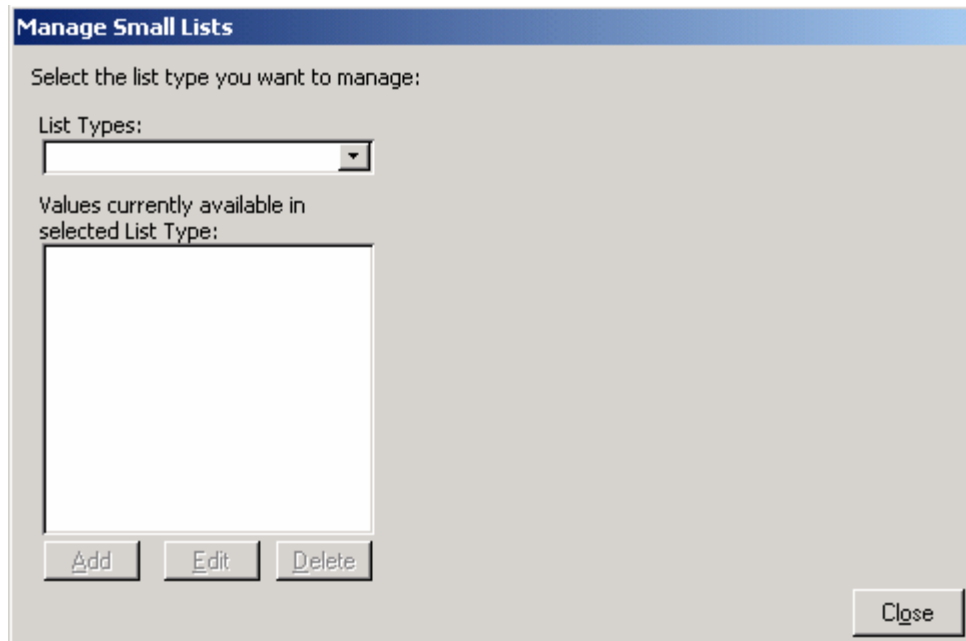
No

Left	2.2917"
Top	1.7083"
Width	1.4167"
Height	0.8333"

We have now completed the process of adding controls to our form. Below is a depiction of how the form should now look when viewed in “Design View” when all controls have been added.

From in “Design View” with all controls added

Below is a depiction of how the form should now look when viewed in “Form View”



Form in "Form View" after all controls have been added

Adding Functionality Using VBA Code

Well, if you have stuck with it and gotten this far with this demo, you are a trooper. Seriously, just building the form, adding all of the controls and positioning them can turn into quite a task. However, a well designed form that provides users with a consistent and easy to understand user interface will certainly provide users with a much more pleasurable experience than one that does not have any continuity.

One of the things that any developer should always address is how are my users going to interface with my application and what path does the application need to take to allow them to accomplish the data management requirements. Remember, in our case, our users are attempting to "manage multiple small lists" that will be used to allow users to make selections for various fields for data entry in the application.

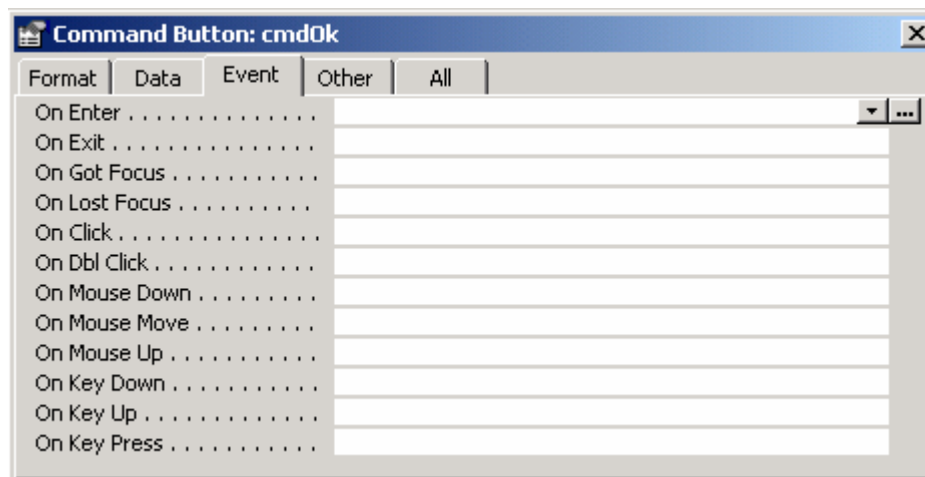
This "list management" tool is only a small piece of the actual application and as such should not be the entire focus of the application. We are making this process the focus right now only because we are using it in a demonstration environment. When it is actually implemented it would be integrated into a much larger database application. In this demo database, I have included another table, the "tblContact" table, which holds basic contact data and another form, the "frmContacts", which actually makes use of the "small lists" that our process manages. You should review the "tblContact" table and the "frmContacts" form as there are additional things that can be learned about how these objects are designed, how they are used and what VBA code has been used to provide the user interface options. The "frmMainMenu" form also provides a basic demonstration of the options that can be used for creating your own menu system rather than use the "Switchboard" method.

Now that I have you thoroughly confused (I hope not) lets get started adding some VBA code to our new form. Just one note before we start; Although there are several things that we will do with VBA coding that could just as easily be accomplished using Macros, I wanted to introduce VBA coding in this demo for those who want to move past macros and really experience the true power and capabilities of Microsoft Access as a development environment.

Let's start off by doing something that you will be required to do in almost any application that you ever build. Let's use VBA code to close our form once it has been opened. (If, when you have completed this demo, you would like to see just how to close one form and open another, you can look at the code behind the "cmdOk" button on the "frmContacts" form.)

To actually add VBA code to an object, be it a form, a control or a report, you must first select the object to which you wish to add code and then you must also locate and select the appropriate "Event" that you will use to cause the code to be executed. There are multiple "Events" for almost every object. Some objects have more "Events" than others. "Events" are directly related to one object and can be described as occurrences that happen when the object is being used. For example, you could use the "On Click" event of a command button to execute VBA code or run a macro. This "Event" would happen when the user "clicks" the command button, thus it is referred to as the "On Click" event because that is exactly what it is and when it occurs.

For our first adventure into VBA coding, select the "cmdClose" button, located at the lower right of our form and display the "Properties" dialog box. This time click on the "Event" tab of the "Properties" dialog box to view the entire list of events that can be utilized for interacting with this and any other command button control. The list of "Events" available for use with a command button is shown in the graphic below.

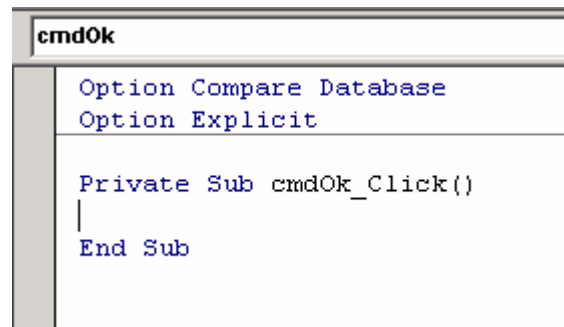


Events tab of the properties for the "Ok" command button

Locate the "On Click" event row and click in the row to select it. Two buttons will appear at the right end of the selected row. We now need to add an "Event Procedure" to this event. Click on the button with the down arrow on it to reveal the list options and

select the “[Event Procedure]” option. Note that in the future when you need to assign the “[Event Procedure]” option to an Event, all you have to do is to double click in the desired Event row and this option will be added as the default value. (Please note that there are other options for using an event but we will not be dealing with those in this demo.)

Once you have the “[Event Procedure]” identified in the Event you wish to use, click the second button at the right end of the selected row to actually create the Event Procedure. You will be taken to what is commonly referred to as the “code window” where the event procedure has now been created. You should see a screen that looks very much like the graphic depiction below.



VBA code window

There could be one major difference between what you are seeing on your screen and the depiction above and that is the “Option Explicit” statement that appears as the second line in the code window. This line appears in the code window when the Access option called “Require Variable Declaration”, available by selecting the Options option from the Tools menu of the VBA Code Window, has been checked. By default this option is not selected (checked) when you start a new database unless you have previously set this option. Once you set the option for one database, all subsequent new databases will also maintain this option. The reason that we want this option turned on is that we want to insure that we always declare and define every variable that we use in our VBA projects.

Now, finally, it is time to actually add a line of code. When you are looking at the VBA code window your cursor should be flashing in between the two lines that identify the Event Procedure. The “Private Sub cmdOk_click()” statement is the identification of the start of the event procedure and the “End Sub” statement is the identification of the actual end of the event procedure.

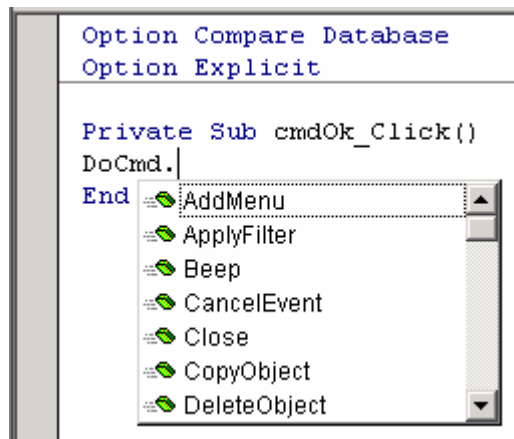
With your cursor between these two statements, type the following code:

DoCmd.Close acForm, "frmManageListInfo"

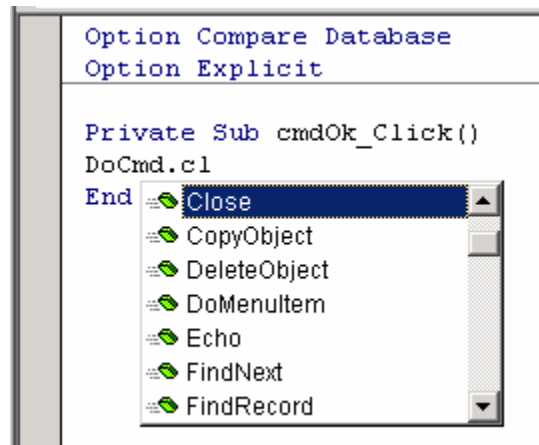
exactly as presented above.

By the way, you can type the statement using all small case letters and the VBA code window will provide you with the proper case for each of your statements if you have enter the syntax of the statement correctly. You should notice that as you type the “docmd” and the period that you will experience something called “intellisense”. This is

a tool that provides the developer with on-screen assistance by presenting a list of the currently available options when building VBA code statements. An example is depicted below.

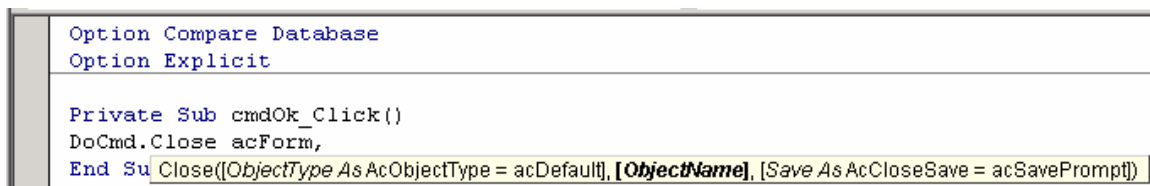


"Intelligence" helping to write VBA Code



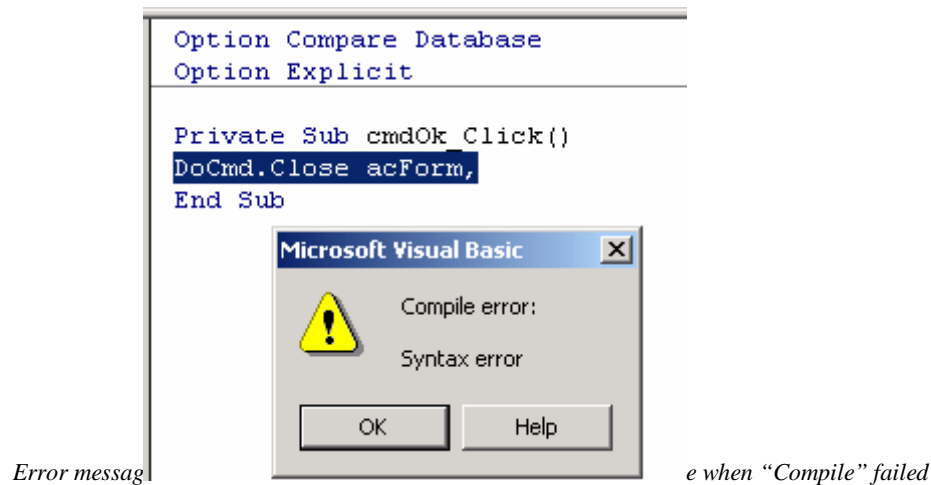
More use of "Intelligence" when writing VBA code

Please note also that as you type, context sensitive "tooltip" type prompts are also provided to assist with the format of commands and the required information that is needed in the context in which you are working. These will prompt the developer for needed content. See below.



When you have completed typing in the line of code for this event, you can check to see if there is any problem with what you have typed by locating and clicking the "Debug" option on the menu and selecting the "Compile MultipleLists" option. In just a

second or so the code will either be compiled successfully (no error messages presented) or you will get a response like the one shown in the graphic below.



In this case, we simply did not provide the required information to complete the command. You can assure yourself that the code did compile correctly, even when you do not get an error message by again selecting the “Compile ...” option from the “Debug” menu and if the option is now grayed-out then you can know that the code compiled successfully. This option will become available again when you make any changes to the VBA code.


Now it is time to test our code by actually using it. Someone has said “code a little; test a lot.” This is truly a good rule to live by. As you add more and more VBA code, be sure to always thoroughly test your code to be sure that it accomplishes the things you expect it to do.

To test your code, first, save your changes to your form and then open the form in “Form View”. Now, click the “Close” button. If you created your VBA code correctly, your form should immediately close.

When you have this working as designed, give yourself a big “thumbs up” for a great job, well done. You are now a VBA coder and have now written your first, working VBA code statement. Congratulations!

Now, let’s add some functionality that was promised way back near the beginning of this demo, just after we added the “List Values” list box and tested our form. If you recall, we could make a selection from our “List Types” combo box, but no values associated with the selected “List type” would be displayed in our list box. We are now going to fix this problem.

The problem here arises from the fact that when we make the selection from the combo box, the list box has no way of knowing that a new value has been selected. We will use

VBA code to force the list box to show us any record related to our selection from our combo box. Select the “cboListTypes” combo box. With the “Properties” dialog box displayed, select the “Event” tab and locate the “After Update” event of our selected combo box. Double-click in this event row and the “[Event Procedure]” statement will be added to this event row. Click on the button with the three dots, , to create the Event Procedure in the VBA code window. Type the following line of code:

Me.lstListValues.Requery

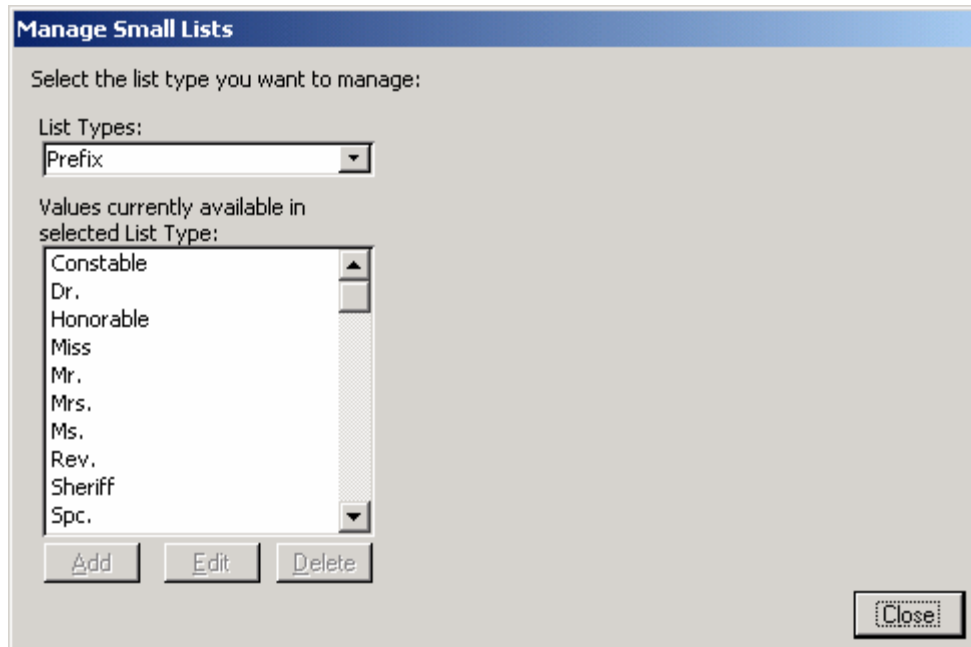
This line of code causes the list box to be “re-queried”. In other words, when the combo box has been updated (after we make a selection from its list), the query that provides the values for the list is run again, causing it to use the new value selected from the combo box to return the “List Values” that match the “List type” value selected in the combo box.

Note: Please understand that although these simple lines of code are sufficient to cause the desired actions to be carried out, additional coding is required to complete a totally designed set of VBA commands. This would include checking for required values before executing commands and providing error trapping code to find and react to unexpected errors.

It is now time to try to compile your code, save your changes when the compile is successful and then test the new VBA code. Open your form in “Form View” and select a value from the “List Types” list box. This time, the “List Values” list box should immediately be populated with the correct list of values.

The previous two very simple and rather short lines of VBA code may not appear to be all that difficult, and they are not. However, can you now see just how easily we can add functionality and automated power to our form with just a couple of simple lines of VBA code?

Below is the screen capture of our form after we have added and tested our VBA code, opened the form in “Form View” and selected “Prefix” from the “List Types” combo box.



Form with only two lines of VBA code and selection made from combo box

The next issue that we will address with our VBA code relates to the “Option Group” that we added to our form. That control is intended to allow our users to modify the “sort order” of the records that are displayed in the “List Value” list box. As of right now, that “Option Group” control and the options it provides is totally not visible. As designed, this control should not be visible until a list is available in the list box. Once a list is displayed, only then would it be appropriate for the option group to be available. We now need to use some VBA code to make this option available to our users. Again we will return to the “After Update” event of the “cboListTypes” combo box. This time we will simply add more code to the existing event.

With your form in “Design View”, click on the “cboListTypes” combo box to select it. Display the “Properties” dialog box if it is not already available. Click on the “Event” tab and then click in the “After Update” event row. Click the far right button on the same row to return to the VBA code window and the selected event. Our original line of code is present and, although this line of code is working just fine at the moment, there is something that we really need to take into consideration before we start to make other options available to our users. The “After Update” event will occur any time there is a selection made from the “cboListTypes” control. We cannot just assume the event occurred because that they picked a value from the list. In fact, they could have just highlighted the content of the combo box, deleted it and then exited the control. This would also have caused the “After Update” event to occur. This would not be a problem for our statement that will requery the list box, but it would not be appropriate if we simply had a statement that would make another control visible. If there was no value left in the combo box control, we would not want the other controls to be available even if there had been a value selected in the combo box but that value was then removed. How do we deal with this type of situation?

Let's add some new lines of VBA code. See the graphic depiction of the new code for the "After Update" event of our combo box.

```
Option Compare Database
Option Explicit

Private Sub cboListTypes_AfterUpdate()
    'request the list box every time the
    'value in the combo box changes
    Me.lstListValues.Requery
    'if there is any content of the
    'combo box is not nothing
    If Me.cboListTypes > "" Then
        'make the option group visible
        Me.frmListOpts.Visible = True
    Else
        'if there is no content in the combo box
        'make the option group not visible
        Me.frmListOpts.Visible = False
    End If
End Sub

Private Sub cmdOk_Click()
    DoCmd.Close acForm, "frmManageListInfo"
End Sub
```

Code window with new code in the "After Update" event for the "cboListTypes" combo box

The first thing that you will see in our new lines of code is that there are several lines that have a single quote (') at the start of each line and that these lines are in a different color. These lines are known as "comments" and should be placed in the code to document the actions and expectations for the code.

This time we have designed our VBA code to evaluate the content of the combo box and, based on that content, determine if other controls should be made visible or not visible. The "If" statement does just that. This statement actually has two parts. One part tells the code what to do in the event that the condition (the combo box has a value), and the other part to tell the code what to do if the condition is not met (the combo box does not have a value).

Now that we actually have something happening when we make a selection from the List Types combo box, we can modify or add to our code to manage the status of the "Add" button located just below the Values List box, again based on the existence of a selected value in the List Types combo box. Due to the fact that when a new value is to be added, a List Type must be designated before any value could be added, we never want the "Add" button to be available to users unless a value has already been selected from the List Types combo box.

Note: When developing any application, you always want to develop from a "pro-active" position, never after the fact. In other words, never let the user have

access to something until all criteria required to complete the task had been met.

Below is the screen capture of our code for the “After Update” event of our List Types combo box after the addition of these two new lines of code.

```
Private Sub cboListTypes_AfterUpdate()  
    'request the list box every time the  
    'value in the combo box changes  
    Me.lstListValues.Requery  
    'if there is any content in the combo box  
    If Me.cboListTypes > "" Then  
        'make the option group visible  
        Me.frmListOpts.Visible = True  
        'make the "Add" command button to be enabled  
        Me.cmdAdd.Enabled = True  
    Else  
        'if there is no content in the combo box  
        'make the option group not visible  
        Me.frmListOpts.Visible = False  
        'make the "Add" command button to be Not enabled  
        Me.cmdAdd.Enabled = False  
    End If  
End Sub
```

Code window with new code in the “After Update” event for the “cboListTypes” combo box

You can now save and again compile your code. Then you can test your form again.

At this point I would like to encourage you to open my original “frmManageListInfo” form in design view and start examining the VBA code. I have tried to place comments in the code to make it more understandable even for those who might not be very familiar with VBA code.

I hope that this demo of the “Manage Multiple Lists” database has been of some benefit to you.