Flyout Menus Using Visual Basic v2.0

Overview

I am beginning a programming project in which I need to implement a simple menuing system that is conducive to alternate input devices such as touch screens. Typical Windows menus can be used with touch screens but, at higher resolutions, can be difficult for "fat-fingered users". Therefore, I decided that I would need a flyout menuing system similar to CorelDRAW and DrafixCAD For Windows.

A flyout menu system uses a toolbox window which provides access to the main menu choices. These are analogous to the top level menus in a typical pull-down menuing system. These main menu choices are typically represented by icons depicting the functions contained in the flyout menu. When the user selects one of the icons in the toolbox, a flyout menu is displayed that contains icons for specific commands. The user can then select the desired command from the flyout. Once a flyout menu item is selected, the flyout menu closes and the selected command is executed.

Usually the icons in the toolbox and flyout contain a pictorial representation of their function, though they can contain text, or anything the programmer desires. The real advantage to flyout menus is that the icon can be any size and color the designer requires and, the icons can be sized dynamically at program run-time using any of the Window's BitBlt API functions.

There are a couple other advantages to flyout menus. Once you have the core routines in place to implement flyout menus and you have made the menu definition generic enough, it is simple to define multiple menus that are used as needed. As a point of comparison, if you tried implement multiple menus using Visual Basic, you would be required to: define all menus under one menuing system, set and reset the visibility property of these menu items as required so that only the menus you wanted were visible at any time, and put up with a lot of screen flickering as you set and reset the visibility property of menu item. Moreover, your multiple menuing system would reside in memory at one time consuming precious system resources. Using the flyout menuing system described here, changing menus is very simple and does not suffer from screen flickering. More important, however, the use of system resources is fixed and can be controlled by the programmer.

The second advantage of flyout menus is they provide quick access to common program functions and commands, and do so in an intuitive manner. The whole idea behind flyout menus is to supplement, not replace, a program's menus. A well thought out flyout menuing system will contain those commands and function most often used. Since the user can place the toolbox window anywhere he desires, he can have quick access to the commands he needs most often. To facilitate this concept, the program designer must use icons that clearly indicate the commands they represent. Therefore, the success, or failure, of any iconic menuing system lies wholly in the hands of the program designer.

There is one disadvantage to the flyout menuing system described here: it provides for only one level in the menu tree. That is to say, a flyout menu cannot invoke a second, lower level, flyout menu. I have not seen a software package implement multi-level flyout menus. That doesn't mean you could not implement them if you wanted to, but you be sacrificing one of the advantages to flyout menus: simplicity. This subject will be discussed later in the *Improvements* section of this document.

Implementation

Design Requirements

The flyout menuing system I needed had to meet a few requirements. Since my program will use the Multiple Document Interface (MDI) inherent to VB, the toolbox window had to be an MDI child window. This allows the user to position the toolbox anywhere within the MDI parent's form, but not outside the parent form.

The toolbox contains a series of bitmap icons that are used to indicate the program functions and commands contained in the flyout menu associated with the toolbox icon (tool). As the user moves the mouse over the tools in the toolbox, a brief help message is displayed in the status line of the MDI parent form that indicates the function of the tool.

When the user presses the left mouse button on a tool, the tool icon is changed to a depressed state and a flyout menu is displayed which contains icons for specific program functions and commands. As the user moves the mouse over the icons in the flyout menu (with the left mouse button depressed), the icon under the mouse changes to the depressed state and the icon the mouse was over previously changes back to its normal state. A help message is displayed in the MDI parent's status line that indicated the command that will be invoked by the flyout menu item the mouse is over. The flyout menu is displayed until the user releases the left mouse button.

If the mouse is over one of the icons in the flyout menu when the left button is released, the flyout menu is closed and the command associated with the flyout menu item is executed. If the mouse is not within the flyout menu when the left button is released, the flyout menu is closed and the menu selection process is aborted.

Tools Required To Implement Design

Visual Basic, in its "out-of-the-box" form did not contain all the tools needed to implement the flyout menu design. The design requires the ability to monitor a series of system messages to determine the location of the mouse and the state of the mouse buttons. These messages are generated by Windows itself in response to certain system events such as moving the mouse and clicking the mouse buttons. Though VB allows access to these messages via event procedures, by the time the event procedure receives the message, VB has filtered it and, more important, does not report all messages. Simply stated, VB does allow the programmer access to all the mouse-related messages needed to implement a flyout menu system.

To get around this limitation, the menuing system had to intercept the needed messages generated by Windows prior to VB getting its hands on them. This type of message interception is call sub-classing. Sub-classing means nothing more than hooking a procedure in the message processing chain at a certain point and monitoring the stream of messages generated by Windows in order to detect and process certain messages. Since VB does not provide the mechanism to sub-class a form or control, I had to use a third-part add-on for VB. The one I chose (in fact, the only one currently available that provides sub-classing capabilities) is SpyWorks-VB by Desaware.

The icons displayed in the toolbox and in the flyout menus are contained in Image controls placed on the toolbox and flyout forms. Since the contents of the toolbox can be dynamic (defined at program run-time), it would be impractical to store all required icons bitmaps in Image controls (remember, each icon

requires one icon for the normal state and one for the depressed state). The preferred method is to store a single bitmap that contains all icons in both their normal and depressed state and then extract the desired bitmap as needed.

The Professional version of VB comes with a custom control called Picture Clip that provides this capability. Using the Picture Clip control, the programmer can load a single bitmap image that contains the icons required for all toolboxes and flyout menus required by the program. By setting the Rows and Cols properties of the Picture Clip control, the number of icons in each row and column of the bitmap can be defined. This, effectively, divides the bitmap into smaller bitmaps that can be accessed individually by an index. Now, extracting an icon from the bitmap is as easy as using the GraphicCell method of the Picture Clip control and specifying the index of the bitmap to be retrieved.

Though not required to implement the flyout menu system, I have also used the 3D Panel control contained in the ThreeD custom control to create the status line at the bottom of the MDI parent's window. The ThreeD custom control is part of the VB Professional Toolkit.

Description Of Data Structures Used

As was mentioned before, one of the design goals was to devise a method to generically define a toolbox and its flyout menus. This scheme allows the core menuing routines to be application independent and allows support for multiple toolboxes in a program. The data structures used are defined below.

The following data structure is used to describe individual icons in the toolbox and in the flyout menus:

```
Type tagIcons
  icon_index              As Integer
  help_str              As String
End Type
```

`icon_index`
The index in the Picture Clip custom control's bitmap for the icon to be used for the toolbox or flyout menu item. Note, this index is for the normal state of the icon. The depressed state of the icon is accessed by adding 1 to this index.

`help_str`
The text string that will be displayed in the status line when the mouse is over this icon.

The following data structure is used to define a flyout menu:

```
Type tagFlyoutData
  num_icons               As Integer
  num_columns           As Integer
  num_rows              As Integer
End Type
```

`num_icons`
The number of icons contained in a flyout's menu. This field must be set correctly prior to using the flyout menu.

num_columns
The number of columns in the flyout menu. This field must be set correctly prior to using the flyout menu.

num_rows
The number of rows in the flyout menu. This field is calculated just prior to displaying the flyout menu. The calculation is based on the num_icons and num_columns fields. You can set this field to anything you want since it calculated at run-time. The reason this field is calculated instead of being preset is so you can change the shape of the flyout menu at run-time. To do so, simply set the num_icons (if needed) and the num_columns fields. The rest is taken care of by the flyout menu routines.

The following data structure is used to define a toolbox and its flyout menus. The previous data structures are incorporated into this master data structure:

```
Type tagToolBox
  title                     As String
  num_items                 As Integer
  num_columns            As Integer
  num_rows               As Integer
  tool_selected             As Integer
  icons(0 To 15, 0 To 16)   As tagIcons
  flyout_data(0 To 15)      As tagFlyoutData
  flyout_item_selected      As Integer
End Type
```

title
The caption for the toolbox window. Remember that the toolbox window will be only as wide as needed to show the number of columns. If you make the title too long, it will be truncated in the toolbox window.

num_items
The number of icon in the toolbox. This field must be set correctly prior to using the toolbox.

num_columns
The number of columns in the toolbox. This field must be set correctly prior to using the toolbox.

num_rows
The number of rows in the toolbox. This field is calculated just prior to displaying the toolbox window. The calculation is based on the num_items and num_columns fields. You can set this field to anything you want since it calculated at run-time. The reason this field is calculated instead of being preset is so you can change the contents of the toolbox at run-time. To do so, simply set the num_items (if needed) and the num_columns fields. The rest is taken care of by the toolbox routines.

tool_selected
Contains the index of the last tool selected from the toolbox. Initially, this field is set to -1. This field is used by the flyout menu routines but there is no reason you can't use the contents of this field for your own purpose.

icons(0 To 15, 0 To 16)
This array contains the definition for all icons in the toolbox and the flyout menus associated with each

tool. The first index of this array refers to the tools in the toolbox. The second dimension of the array refers to individual menu items in the flyout menu associated with the tool. For example: icons(2, 0) refers to the icon definition for the third tool in the toolbox whereas icon(2, 3) refers to the icon definition for the fourth item in the flyout menu associated with the third tool in the toolbox.

This array is hard-coded for 16 toolbox tools each having a flyout menu with up to 16 items.. You can change the size of this array if you remember three things: change the index of the flyout_data array to match the first dimension of this array, add enough Image controls to the toolbox form so there is one Image control for the first dimension of this array, and add enough Image controls to the flyout form so that there is one Image control for the second dimension of this array. This first dimension of this array must have a lower bound of 0 since the index is also used to address a corresponding Image control array element on the toolbox form. Likewise, the lower bound of the second dimension of the array must be 0 since it corresponds to the Image control array on the flyout form.

`flyout_data(0 To 15)`
This array contains the definitions for all flyout menus in a toolbox: one definition for each tool in the toolbox. When the user selects a tool from the toolbox, the corresponding element of this array is used to determine the size and shape of the flyout menu that will be displayed.

Like the icons array described above, this array is hard-coded for 16 toolbox tools. You can change the size of this array if you remember two things: change the first index of the icons array to match the size of this array, and, add enough Image controls to the toolbox form so there is one Image control for each element of this array. This array must have a lower bound of 0 since the index is also used to address a corresponding Image control array element on the toolbox form.

`flyout_item_selected`
Contains the index of the item selected from the last flyout menu displayed. Initially, this field is set to -1 and, if the user aborts the flyout menu selection process, this field is set to -1. This is the field your program uses to determine what flyout menu selection was made.

Forms Used And Their Contents

The flyout menuing system uses two forms: the toolbox form and the flyout form. The toolbox form is a child window of the MDI parent form (its MDIChild property is set to TRUE). You could make this form a popup window by setting its MDIChild property to FALSE but, by doing so, the user will be able to move the toolbox outside the program's main form which may not be desirable. This form contains 16 Image controls in a control array indexed from 0 to 15; one Image control for each element in the first dimension of the Toolbox.icons() array, and the Toolbox.flyout_data() array.

The flyout form contains one Image control, stored in a control array (0..15). There is one Image control for each element in the second dimension of the Toolbox.icons() array (not counting the 0th element in the second dimension).

These two forms also contain an instance of Desaware's Sub-classing custom control. These controls are used to process various Window's messages.

System Logic Flow

To use the flyout menu system, invoke the procedures as follows:

1) Load the toolbox form. Use the standard VB Load command to load the form.

2) Initialize the toolbox. Call the InitializeToolbox() routine to define the contents of the toolbox and its flyouts. If you want to support multiple toolboxes, define all of them in this routine.

3) Arrange the toolbox. Call the ArrangeToolbox routine which performs the following actions:
   a) Calculate the number of rows in the toolbox.
   b) Determine the size of the toolbox window.
   c) Load the appropriate icons from the Picture Clip bitmap into the corresponding Image controls.
   d) Set the Visible property for all Image controls used to TRUE.
   e) Set the Visible property for all Image controls not used to FALSE.
   f) Move and size the toolbox window on the screen. Note this routine is hard-coded so that the upper left corner of the toolbox window is 10 pixels from the top of, and 10 pixels from the left edge of the MDI parent window. You can change this position to anything you want.

4) Load the flyout form. Use the standard VB Load command to load the form.

Now, the flyout menuing system is available for use. The following describes the logic flow for the system.

1) As the user moves the mouse over the icons in the toolbox window, the Sub-class control on the toolbox form intercepts the mouse messages, determines which icon the mouse is over, and displays the help string associated with the icon in the MDI parent's status line.

2) When the user presses the left mouse button over one of the tools in the toolbox, the Image control's MouseDown event procedure:
   a) Sets the contents of the Toolbox.tool_selected field to the index of the tool selected.
   b) Changes the icon for the tool selected to its depressed state.
   c) Calls the ArrangeFlyout routine if needed (this routine is not called if the last tool selected is the same as the currently selected tool). The top edge of the flyout menu window is aligned with the bottom edge of its tool's icon in the toolbox, and the left edge of the flyout is aligned with the center of the tool's icon.
   d) Starts sub-classing the flyout form.
   e) Captures all mouse input to the flyout form.
   e) Displays the flyout form as a modal form.

3) The flyout menu is now displayed. The Sub-class control on the flyout form processes a set of the Windows mouse messages (WM_MOUSEMOVE, WM_LBUTTONDOWN, and WM_LBUTTONUP). Remember, the user still has the left mouse button depressed. As the mouse is moved, the icon the mouse was over previously (if any) is changed to its normal state, the icon the mouse is currently over (if any) is changed to its depressed state, and the help string associated with the icon the mouse is over is displayed in the MDI parent's status line (if the mouse is outside the flyout menu, the status line is cleared). This processing is continued until the user releases the left mouse button at which point: the icon the mouse is over is restored to its normal state, the status line is cleared, the capture of all mouse input is released, sub-classing for the flyout form is ended, and a Window message is posted to the toolbox form.

4) The Sub-class control on the toolbox form receives the message posted by the flyout form. One of

the parameters of the message is the index of the flyout menu item selected by the user. The value of this parameter is stored in the Toolbox.flyout_item_selected field. If the left mouse button was released outside the flyout menu (the menu selection process was aborted), this parameter contains -1. The flyout form is now hidden, the icon for the tool selected in the toolbox is restored to its normal state and you can now add the logic required to act on the menu choice which is stored in the Toolbox.flyout_item_selected field.

To stop using the flyout menuing system, unload the toolbox form. All cleanup functions will be performed in the toolbox form's Forn_Unload event procedure.

Improvements

Improving The Initial Display Of The Toolbox Window

You will notice when you run the demo program that the toolbox window is initially displayed in one place with the wrong size, and then is correctly positioned and sized. This results from the fact that the toolbox form is an MDI child window which means that it can't be hidden. The initial (incorrect) display of the window is a result of loading the form. Immediately after loading the form, the InitializeToolbox and Arrange Toolbox procedures are called which position and size the toolbox correctly.

One way to avoid the initial, incorrect display of the toolbox window is to make it a popup window (its MDIChild property is set to FALSE). That way the form will not be displayed when it is loaded and the form can be correctly positioned and sized before ever displaying it. The downside to this approach is that the user will be able to move the toolbox anywhere on the screen, even outside the program's main form. This may be acceptable to you but there are two things to bear in mind if you take this approach: you will have to manually unload the form when your application terminates, and, you will have to manually hide the toolbox window when the user minimizes your program's main window.

Code Reduction

You will notice that the demo program includes a routine called InitializeToolbox which defines the toolbox and its flyouts. In a production environment you will probably want to write a small program that will create these definitions and store them in a file. Then, the Initialize toolbox routine would simply read in the toolbox definitions from the file at program run-time.

Support For Multiple Toolboxes

The global data structure used to define and process flyout menus is called gToolbox. Obviously you should only have one toolbox active at one time but you could have multiple toolboxes defined in your program.

One method for implementing multiple toolboxes is to create an array of tagToolbox data structures to hold the definitions for all your toolboxes. Then load the one you want to use in the gToolbox structure and proceed normally. When you want to switch toolboxes, unload the current toolbox, load the new definition from the array into gToolbox and start processing the toolbox as usual.

Support For Multi-level Flyouts

If desired, you could add additional depths to the flyout menu structure. In other words, an item on a flyout menu could display yet another flyout menu. To implement this scheme, you would have to add a additional dimensions to the Toolbox.icons() and Toolbox.flyout_data() arrays to contain the definitions for these lower-level flyouts. You will also have to add some additional routines to process these lower-level flyout menus. These additional routines would duplicate the functionality of the ArrangeFlyout() and Sub-class procedures. On the other hand, you could make the ArrangeFlyout() and Sub-class procedures more generic by passing them the data structures needed to process a particular flyout menu.

Dynamically Allocating The Image Controls

The toolbox and flyout forms contain 16 Image controls each that are stored in a control array. You could optimize the use of system resources by only having one Image control on each form (the 0th element of a control array) and then expanding the control arrays as needed in the ArrangeToolbox and ArrangeFlyout routines. That way you would only have the number of Image controls needed at any given time.

Though I haven't tried implementing this, I suspect it would impose a delay between the time the user clicked on a tool in the toolbox and the time the flyout is displayed. Maybe not.

Toolbox Tools With No Associated Flyout

Though not supported in this version of the flyout menuing system, it would be a trivial task to add support for a toolbox tool that does not have a flyout menu associated with it. When the user selects such a tool, simply post a WM_USER message to the toolbox form passing some dummy parameter.

Technical Notes

Sub-classing: Pre-default And Post-default

As I mentioned before, the flyout menuing system uses the SpyWorks-VB custom control from Desaware which hooks itself into the Window's message processing chain. This control allows you to specify where in the chain to hook itself: before or after the default message processing routine. Hooking before the default message processing chain allows you to get at all the messages being sent to your program. Hooking after the default message processing routine lets you see the messages after the default message processing routine has performed whatever filtering it would normally perform.

The Sub-classing control in the flyout form hooks itself before the default message processing routine. This is done because it needs to see all messages to determine the state of the mouse inside and outside the flyout window and to prevent certain message from being passed on to the default message processing routine. On the other hand, the Sub-classing control on the toolbox form hooks itself after the default message processing routine because it only needs mouse messages while the mouse is within the toolbox window (to display the help strings): any mouse activity outside the toolbox window is of no interest.

Fast And Firm No-Nos

One rule of thumb when using a Sub-classing control: NEVER unload the form the control is sub-classing within the Sub-class control's event routines. If you do you'll generate a hard-GPF, guaranteed!

The Demonstration Program

The enclosed demonstration program doesn't do much else than demonstrate the flyout menuing system described in this document. I wasn't too creative with the icons used: I used the sample icons shipped with VB.

The archive contains all the files you need to run the demonstration program except for VBRUN200.DLL. You should have this file. If not, there are copies available on almost all BBS systems.

The archive also contains the source code for the demonstration program. Feel free to use and abuse this code anyway you see fit. In order to load this code into VB, you must have both the VB Professional Toolkit (for the Picture Clip and ThreeD custom controls) as well as Desaware's SpyWorks-VB custom control.

Peter Koch
74007,2450
MSBASIC Forum