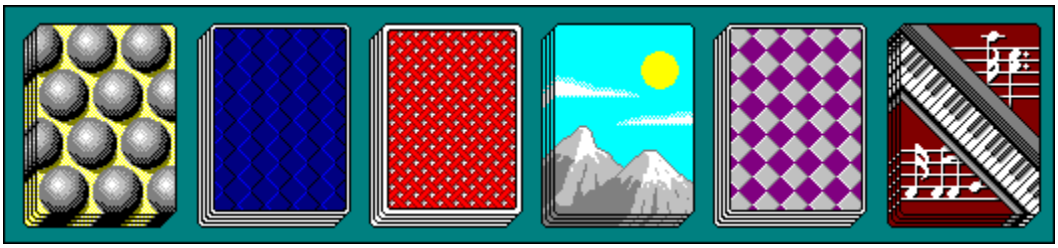
Welcome to QCard.DLL 2.0



Glad you could come and play...

cards for fun and profit!

Subroutines & Functions

Are things starting to drag...?

Credit where credit is due...

ABOUT QCard.DLL

What is QCard.DLL?

QCard.DLL is a dynamic link library that simplifies the creation of card games for the Windows 3.x environment. It grew out of the author's desire to write his own card games which were as good in quality as the Windows solitaire game. There were some other card DLL's available, but they proved to be somewhat slow, and the final result was not always that pretty.

QCard.DLL is fully compatible with the Visual Basic environment as well as C/C++. If you are working in VB you might encounter slightly slower execution speed, but this is the nature of VB. **QCard.DLL** gives you easy access to 40 card-game specific functions and procedures. Creating a Windows card game has never been easier!

I do this for fun, not for money! Therefore **QCard.DLL** is released to an unsuspecting world as FREEWARE and offers no ties or binds to its author. *The other benefit of releasing this as FREEWARE is being able to avoid the thirteen paragraphs of legal mumbo-jumbo that would normally go right here.*

Contents

Cards Up Your Sleeves

Writing Card Games

Writing a card game can be a frustrating process even when using tools like this. **QCard.DLL** will not write your card game for you! You will need to do lots of work behind the scenes to ensure that you are passing **Qard.DLL** correct information in the correct way. The process is eased by the amount of fore-thought you bring to it. Like any other programming project, the more time you spend designing and thinking about what you want to do, the easier it is once you actually sit down and start coding it. I am as guilty as the next person when it comes to just sitting down and banging out wonderfully creative code with no regard to where it is heading. I usually end up paying the price, which is involves lengthy rewrites. A solid plan is always a good idea before you start.

Cards Up Your Sleeves

Using VB and QCard.DLL

You will need to declare all of **QCard.DLL's** functions in your Global Module. You can copy them from the VB demo program which is included here. Without these declarations, your program cannot access the routines in the DLL. When you are building your program, you will need a copy of **QCard.DLL** in your VB directory so VB can access it. Once you have built your program, and you are running it from its own EXE, you will need a copy of **QCard.DLL** in that EXE's current directory. As an alternative, you can put a copy in your Windows directory, in which case Windows can locate it at design time and run time. If your program can't find **QCard.DLL**, make sure you have it located in the right directory. Once you have finished your project, and you are ready to release it to the world, you will need to ship with it a copy of **QCard.DLL**.

In many of the sub and function calls to **QCard.DLL**, you will need to reference the handle of the window in which you are working. In VB, this is the hWnd property of the form you are drawing in, (for example, Form1.hWnd). When drawing and locating items, **QCard.DLL** uses the MM_TEXT mapping mode. This is a pixel-based co-ordinate system where the top-left corner of your window is at x, y point 0, 0. Values increase as you move to the right and down. You should set the ScaleMode property of the form you will be working with to Pixel (3), and leave its ScaleTop and ScaleLeft properties at their defaults of 0. This will ensure that both **QCard.DLL** and your application will be talking in the same terms when making reference to locations.

QCard.DLL gives you two decks of cards to work with. If you need any more decks than that in one game, then you will have to improvise your own work-around. The cards are referenced by their number. Numbers range from 1 to 113, and are arranged as follows:

1 to 52	**Deck 1**
53 to 104	**Deck 2**
105 to 109	**Cardback cards**
110 to 113	**Jokers**

If you just need one deck in your game, you can ignore numbers 53 to 104. The easiest way to keep track of what you are doing is to create a Deck array of your own, and fill it with the numbers 1 through 52 (or 104 if you are using 2 decks).

```
Global Deck (1 To 52) As Integer
Dim i As Integer
For i = 1 to 52
        Deck(i) =   i
Next i
```

Then you can go ahead and shuffle the array, if thats what you want.

```
Randomize Timer
For i = 1 To 10
    For j = 1 To 52
        k = (Int(Rnd * 52) + 1)
        Temp = Deck(j)
        Deck(j) = Deck(k)
```

```
        Deck(k) = Temp
    Next j
Next i
```

You can now make calls to **QCard.DLL** by referencing your array. For example,

```
DealCard    Form1.hWnd, Deck(1), xLoc, yLoc
```

This will deal the first card in your array on Form1 at location xLoc, yLoc in pixel coordinates relative to the top-left corner of your form.

You might want to declare a few Global Constants which will make your job easier. These include the width of the cards, the height of the cards, the number of cards, etc. These just make the numbers easier to remember:

```
Global Const CARDWIDTH = 71
Global Const CARDHEIGHT = 96
Global Const NUMCARDS = 52
```

Cards Up Your Sleeves
Contents

QCard.DLL Card Properties

You can think of each card as having a set of properties, some of which you can change and reference as your program executes. The properties you have access to are as follows:

Related Topics:

1) Number

You reference a card by its number. The cards are numbered 1 through 113. 1 to 52 is the first deck. 53 to 104 is the second deck. 105 to 109 are cards that just display the current cardback but act like other cards and are useful for building piles of undealt cards, and whatnot. 110 to 113 are jokers, a black and a red one for each deck. To reference a card in a fuction call to **QCARD.DLL**, just call on its number. In the text that follows , nCard refers to the number of the card you are referencing, ie)

GetCardSuit(nCard).

Other Properties

2) Color

You can determine a card's color by calling the function GetCardColor(nCard). Black cards return a value of 1, Red cards return a value of 2.

Other Properties

3) Suit

You can determine a card's suit by calling the function GetCardSuit(nCard), passing it the number of the card you are interested in. Clubs return 1, Diamonds return 2, Hearts return 3, Spades return 4.

Other Properties

4) Value

You can determine a card's value by calling the function GetCardValue(nCard). An ace returns 1, and a King returns 13. The values in between are as you might expect.

Other Properties

5) X

You can determine a card's x location by calling GetCardX(nCard). This will return its current location from 0 on the left side of your Form. This value is originally set at -72. When a card is dealt using the DealCard sub, this value is updated automatically. This value can be set manually by calling SetCardX(nCard, nLoc), giving the sub routine the card number and new x location value.

Other Properties

6) Y

You can determine a card's y location by calling GetCardY(nCard). The value is originally set at -97. When a card is dealt using the DealCard sub, this value is updated automatically. It can be set manually by calling SetCardY(nCard, nLoc).

Other Properties

7) Status

This is a Boolean value (TRUE or FALSE) which is initially set to TRUE. When a card is dealt or drawn, it will be faceup by default. You can determine a card's status with the Function GetCardStatus(nCard) which returns either TRUE or FALSE. You can change a card's status between faceup and facedown by using the Sub SetCardStaus(nCard, bValue) where bValue is TRUE for faceup, and FALSE for facedown.

To deal a card facedown, simply set its status to FALSE, and deal it:

```
SetCardStatus 1, FALSE
DealCard Form1.hWnd, 1, 10, 10
```

To flip it over (say, in response to a user clicking on it), change its status to TRUE and redraw it at its current location:

```
SetCardStatus 1, TRUE
nX = GetCardX(1)
nY = GetCardY(1)
DrawCard Form1.hWnd, 1, nX, nY
```

The cardback used for the facedown image will be that set by the SetCurrentBack Sub.

Other Properties

8) IsBlocked

The IsBlocked property is used when determining if a card is free for dragging or not. When a card is dealt, and another is moved on top of it by you or by the player, this value should be set to TRUE for the lower card by calling AdjustCardBlocked (nCard, TRUE). To put it simply, if you have a pile of cards, only the topmost card should have an IsBlocked value of FALSE. All the cards below it should have an IsBlocked value of TRUE. This makes it possible for **QCARD.DLL** to carry out its hit-testing when initializing a drag event. The IsBlocked value of a card is returned by a call to GetCardBlocked(nCard).

Other Properties

9) Disabled

If you want to remove a card from play, you can set its Disabled property to TRUE. Doing this will make it impossible for the user to select the card for dragging with the mouse. This can be done by calling SetCardDisabled(nCard, TRUE). The card can be enabled again by calling SetCardDisabled(nCard, FALSE).

Other Properties

10) BOOL User1

This is a Boolean (TRUE or FALSE) user defined property which you can use as you see fit. You may want to use it to monitor some particular attribute in your game. You can set this value by calling SetUser1(nCard, bValue) where bValue is either TRUE or FALSE. You can obtain its present value by calling the GetUser1(nCard) function.

Other Properties

11) int User2, int User3 & int User4

User2, User3 and User4 are all integer types. You can set these according to your needs. You set them by calling SetUser2(nCard, nValue), SetUser3(nCard, nValue) and SetUser4(nCard, nValue), where nValue is any integer. You can obtain their current values by calling the GetUser2(nCard), GetUser3(nCard) and GetUser4(nCard) functions.

For example, your game might award a player 8 points for placing an Ace in a certain pile. Before shuffling, you could call:

```
For i = 1 To 40 Step 13
        SetUser3(Deck(i), 8)
Next i
```

Then when the card is played and you are determining how many points to award, you can call:

```
        nNumPoints = GetUser3(nCurrentCard)
```

and retrieve the value.

Other Properties

NOT ALL SMOKE AND MIRRORS

QCard.DLL Subs and Functions

Once you have included the sub and function declarations in your Global Module, you can call **QCARD.DLL** functions just as if you were calling any other function or sub in VB. The following sections list each function and sub and describes what it does. In this documentation, any parameter which begins with n, such as nCard, indicates that that value should be an integer. Any parameter which begins with b, such as bValue, indicates that that value should be a Boolean, TRUE or FALSE. CALLING THESE SUBS AND FUNCTIONS WITH VALUE TYPES OTHER THAN THOSE INDICATED IS A GOOD WAY TO LOCK UP YOUR SYSTEM! Always save your work before you run a routine to test it, especially the Block Dragging routines. If you pass these routines an improper value, you will be dumped out of VB and any of your unsaved work will be lost. Not a big deal, that's all part of the Windows programming experience!

Just remember to save your work first.

Contents

Initialization Functions

BOOL InitializeDeck(hWnd)

Only call this function once in your application. Did you catch that? I said "ONLY CALL THIS FUNCTION ONCE IN YOUR APPLICATION!!". One call to this function is all that's required for **QCARD.DLL** to operate; any more calls than that, and you will get screwy things happening in your game. This sets up all the card pictures and values within the DLL. You can make this call in your Form.Load event, passing it the handle of the window in which you will be working. This function returns TRUE if it is successful and FALSE if it fails. You should always check the return value of this function and act accordingly if it fails. For example:

```
nReturnValue = InitializeDeck(Form1.hWnd)
If nReturnValue = FALSE Then
        MsgBox "Sorry. Another application is currently using
                QCard.DLL"
        ' bail out now!
        End
End If
```

Generally, the InitializeDeck function will only return a FALSE value if the DLL is already in use by another application. Only one application can use **QCARD.DLL** at any given time. This is due to the fact that **QCard.DLL** (unlike other DLLs which don't maintain data), contains lots of data. Namely, where all of your cards are located, their current background bitmaps, and other such wonderful things. This should not pose a problem unless a user tries to run two copies of your game at the same time. It may be a good idea to mention this fact in your game's documentation. When your application ends, **QCARD.DLL** is released and unloaded by Windows. You may find that if your application dies and ends prematurely when you are designing and running it, due to a General Protection Fault on your part, Windows might not properly unload the DLL. You will have to restart Windows again to clear the DLL out of memory before you can continue debugging your game. Under normal conditions, the DLL will be unloaded automatically by Windows.

Other Initialization Subs and Functions

SetCurrentBack(nIndex)

In addition to the DrawBack Sub, which draws one of six card back designs in your window, you can also use the regular card drawing and dealing subs with card numbers 105 through 109 to draw card backs. Cards numbered 105 through 109 act just like other cards, but their picture is a card back design rather than a card front design. This allows you to manipulate 5 facedown cards in the same way you can the other regular cards. The picture is the same for all five cards and is initially set at cardback design number 1. Call SetCurrentBack(nIndex), where (nIndex) is a number between 1 and 6, to change these cards to a different design. You may call SetCurrentBack(nIndex) any number of times to change card backs during your game, but remember to re-draw any previously dealt face-down cards to reflect the new choice.

You can also use cards 105 through 109 to display a pile of cards that dwindles as as the user clicks on the pile, as in Windows Solitaire. To achieve this effect, first draw the "O" symbol on your form. Then deal card 105 directly on top of it. Then deal cards 106 through 108, each time offsetting their x and y by 2. This creates a nice 3-D stack effect. You will need to block all the cards except the top one (108 in this example). As an example, part of your MouseDown Event might look like this:

```
Dim Shared nTopCard As Integer
nTopCard = 108
nSourceCard = InitDrag(Form1.hWnd, x, y)
If nsourceCard = nTopCard Then
        RemoveCard nTopCard
        SetCardDisabled nTopCard, TRUE
        AdjustCardBlocked nTopCard - 1, FALSE
        nTopCard = nTopCard - 1
End If
AbortDrag
```

This partial code sample is just to get you started. To create a fully developed card pile, you will need to add much more functionality to the routine.

Other Initialization Subs and Functions

SetDefaultValues()

Use this sub to reset all card properties back to their default values. A good time to use this is right before setting up a fresh deal, so you can be sure all previous values are flushed out. It has no parameters. Call this as many times as you want during the course of your game.

Other Initialization Subs and Functions

SetCardStatus(nCard, bValue)

This Sub allows you to change any cards from their default setting of faceup to facedown. If you set a card's status to facedown, ie, SetCardStatus (1, FALSE), it will be treated as facedown by the DLL when it is drawn or dragged. The image used for the facedown image will be that set by the SetCurrentBack(nIndex) Sub. Simply set the Status of any cards in your game which will be dealt facedown to FALSE, and then deal them as normal. They will be drawn facedown. To flip them faceup, set their Status to TRUE with SetCardStatus(nCard, TRUE), and use DrawCard to draw them faceup at their current location.

Other Initialization Subs and Functions

SetOffSet(nValue)

If you want to use **QCard.DLL** for dragging blocks of cards in your game, you must deal cards in vertical columns as in Windows Solitaire. By default, cards in each column should be offset 16 pixels down from each other. If you wish to use a different vertical spacing in your game (other than 16 pixels), inform the DLL of the new spacing value with this Sub. nValue is the new value which will be used by the DLL to carry out block dragging.

Other Initialization Subs and Functions

Card drawing subs

DrawCard (hWnd, nCard, nxLoc, nyLoc)

This is the quickest and easiest way to draw a card onto your window. Simply pass it your window handle, the number of the card you want drawn, and the x, y location you want the card to appear at. The DrawCard sub does not update any of the card's data members, such as its x or y location. If your application does not require any of this other information, this may be the only drawing sub you need to use. You cannot implement dragging operations if this is the only sub you use to draw your cards, however. Again, this does not update any of the card's data members. It just draws the card on the screen. It is fast and simple. (This makes it good for redrawing items for screen updates, as well).

Other Card Drawing Subs

DealCard (hWnd, nCard, nx, ny)

The DealCard sub does many important things over and above the DrawCard sub. It updates the card's X and Y properties to the location you deal the card. Most importantly, it grabs from the video display that portion of the screen your card will be covering over. That is, it keeps a copy of the image which lies behind that card. This is very crucial when you go to drag a card, because whatever used to be behind the card has to be replaced on the video display. If you are going to be doing any dragging, you must place your cards on the screen using the DealCard sub. If you try to drag a card that was originally drawn using the DrawCard sub alone, you will end up with a video mess.

Please note: the cards in **QCARD.DLL** adapt to any background color your window may have. You can feel free to include an option for the user to change window colors in your game knowing that funny colored corners will not appear on the cards if the background color changes. One warning, however. As has been mentioned, each card carries with it a copy of the screen image which lies behind the card, for dragging purposes. If you deal the cards on a green background, and the user changes color to a red background, your card's background images will still reflect a green background. Not a pretty sight when he starts dragging! When changing screen colors in midstream, you should:

- Remove your cards from the screen
- Repaint the window to the new color
- Use the DealCard sub to replace your active cards at their proper location

Doing this will ensure that their background images correspond to the present background color.

Other Card Drawing Subs

DrawSymbol (hWnd, nValue, nx, ny)

This sub draws the basic X, O and place holder symbols. It requires the hWnd of your form, a symbol value and an x and y position where you want the symbol drawn. Valid values are 1 for an X, 2 for an O, and 3 for the place-holder. These symbols feature a gray background rather than the usual black. They will show up on any background color, including very dark colors.

Other Card Drawing Subs

DrawBack (hWnd, nValue, nx, ny)

This sub draws one of the six included cardback designs at the location x, y. Valid values are 1 through 6 inclusive. Use Card numbers 105 to 109 if you want actual cards that show the current cardback as their image. The DrawBack Sub only draws a picture of the selected cardback on the screen.

Other Card Drawing Subs

RemoveCard (hWnd, nCard)

This sub removes the card from the window display assuming two things are true: First, the card must have been originally placed on the window using the DealCard sub. Second, the card must not be overlapped from above in any way. This sub actually repaints the card's background image where it used to be.

Other Card Drawing Subs

Information Functions

GetCardColor(nCard)

This function returns the color of the card specified. It returns 1 for a black card, 2 for a red card.

Other Information Functions

GetCardSuit(nCard)

This function returns the suit of the card specified. It returns 1 for Clubs, 2 for Diamonds, 3 for Hearts, 4 for Spades.

Other Information Functions

GetCardValue(nCard)

This function returns the value of the requested card. Aces have a value of 1, Twos have a value of 2, right up to King which has a value of 13. For example:

```
nReturnValue = GetCardValue(nCard)
If nReturnValue = 11 Then
        Text$ = "Jack"
End If
```

Other Information Functions

GetCardStatus(nCard) AND SetCardStatus(nCard, bStatus)

This Function and Sub pair can be used to get the current faceup/facedown status of a card, and also to change the current faceup/facedown status of a card. All cards originally have a Status of TRUE, or faceup. If you want some cards to be facedown, set their Status to FALSE with SetCardStatus(nCard, FALSE). Those cards will be handled as facedown by the DLL when they are subsequently drawn, dealt or dragged.

Other Information Functions

GetCardBlocked(nCard) AND AdjustCardBlocked(nCard, bValue)

These get and set the IsBlocked value of a card. You will have to pay particular attention to this property if you will be doing any dragging. Imagine Windows Solitaire. When the user presses the mouse button down over a card, the cursor may actually be over many cards in a pile. A technique is required which allows the application to determine which of those cards should actually be selected. **QCARD.DLL** uses a method of putting a block on all cards covered over by another card. This is your responsibility to maintain.

When initializing a drag event, **QCARD.DLL** first checks for any unblocked cards under the mouse cursor. If it finds one, it will return the number of that card. This initiates a Single drag operation. If it doesn't find one, it then determines if it is in the top 16 (or user defined OffSet) pixels of any other card, including blocked cards. If it is, it returns the number of that card. This initiates a Block drag operation. If you maintain your cards in proper blocked and unblocked fashion, you will have no trouble dragging single or groups of cards in the same way as Windows Solitaire. When creating a row or pile of cards, only the topmost card should have an IsBlocked value of FALSE. Remove a block by calling AdjustCardBlocked(nCard, FALSE). Block a card by calling AdjustCardBlocked(nCard, TRUE).

Other Information Functions

IsCardDisabled(nCard), SetCardDisabled(nCard, bValue)

You can set a card's Disabled property to TRUE so it can no longer be selected by the mouse for drag operations. Again, imagine Windows Solitaire. Even when cards are played to their top, final locations, they can be dragged down again and replaced on the lower piles. If you do not want "finished" cards to be replayed like that in your game, you can set their Disabled property to TRUE with SetCardDisabled(nCard, TRUE). Then the user will no longer be able to drag them back down into play.

Other Information Functions

GetCardX(nCard) AND GetCardY(nCard)

Use these functions to get the x and y location properties for a card. These are pixel coordinates based on 0, 0 in the top left corner of the window you are working in. These, along with SetCardX and SetCardY, are very useful for drawing and relocating cards around your window. For example, when the user drags a card over onto a new pile and lets it go, you will want to relocate it and snug it up below the previous card:

```
nDestCard = EndDrag Form1.hWnd, x, y
nNewX = GetCardX(nDestCard)
nNewY = GetCardY(nDestCard)
RemoveCard Form1.hWnd, nSourceCard
' using the default value of 16 for OffSet
DealCard Form1.hWnd, nSourceCard, nNewX, nNewY + 16
```

Other Information Functions

SetCardX(nCard) AND SetCardY(nCard) Subs

Use these two Subs to change the current X or Y setting of a card. Although this changes the value of X or Y, it does not move the card to the new location.

Other Information Functions

GetUsern(nCard) & SetUsern(nCard, nValue)

Use these subs and functions to get and set values of your choosing which you can associate with any of your cards. User1 is a Boolean TRUE and FALSE value. User2, User3 and User4 are all Integer types. You can use these any way your application requires. For example, you can call SetUser4 12, 1000 to set a User value of 1000 to card 12, and retrieve that value later by calling nMyValue = GetUser4(12).

These can come in handy in a variety of ways. In the demo program, they are used them to keep track of which array each card belongs to and its position in the array. This makes it easy to move cards from one pile (array) to another as the game executes.

Other Information Functions

Dragging Functions

InitDrag(hWnd, nx, ny)

Use this function in a MouseDown event to start a drag operation. The function searches through all cards to determine if the mouse cursor is over any card whose IsBlocked property is FALSE. If it finds one, it returns the number of that card. If it does not find one, it searches through all the cards in the deck to see if the mouse lies in the top 16 (or User-Defined OffSet) pixels of any card, blocked or not. If it does, it returns the number of that card.

By checking the IsBlocked property of this returned card, you can tell if the user wants to carry out a single drag or a block drag. If InitDrag returns a value of 0, the mouse is not currently located over any card. The InitDrag function should always be followed by a corresponding AbortDrag, an EndDrag or an EndBlockDrag call. This is due to that fact that InitDrag "captures" all mouse input, and requires one of these corresponding calls to release the capture. The values nx and ny are the current mouse coordinates.

Other Dragging Functions

AbortDrag() Sub

This sub ends any drag operation started by an InitDrag call. AbortDrag releases the mouse which is captured by InitDrag. Abort drag takes no other action.

Other Dragging Functions

DoDrag(hWnd, nx, ny) Sub

Carries out the drag operation which was initiated by InitDrag call. DoDrag moves the current Source Card to its new location. Values nx and ny are the current mouse coordinates.

Other Dragging Functions

BlockDrag(hWnd, CardList(0), nNumCards, nx, ny)

BlockDrag carries out a block drag operation which was initiated by an InitDrag call. It requires a list of cards to be dragged in the form of an array. The array can be passed to the sub using the array's first element (0). The sub also requires the number of cards to be dragged as well as the current mouse coordinates.

Other Dragging Functions

EndDrag(hWnd, nx, ny)

EndDrag ends a single drag operation and returns the number of the Destination card (that is, the card it is being dropped on), if any. It searches the deck for any card which overlaps the Source Card and whose IsBlocked property is FALSE. If it finds one, it returns the number of that card. The function also releases the mouse which was captured by the InitDrag call.

Other Dragging Functions

EndBlockDrag(hWnd, CardList(0), nNumCards, nx, ny) Function

EndBlockDrag ends a block drag operation which was initiated by an InitDrag call. The function searches through the deck for any card which overlaps the Source Card and whose IsBlocked property is FALSE. If it finds one, it returns the number of that card. The function also releases the mouse which was captured by the InitDrag call. The function requires a list of the cards being dragged in the form of an array. The array can be passed to the function using the array's first element. The function also requires the number of cards being dragged and the current mouse position.

Other Dragging Functions

ReturnDrag(hWnd, nCard, nxLoc, nyLoc)

This sub drags the card nCard to the location nxLoc, nyLoc along a straight line from its current location. Return drag can be used for returning cards to their original location after an invalid drag operation. (That is, drag operations that your game determines to be invalid).

Other Dragging Functions

ReturnBlockDrag(hWnd, CardList(0), nNumCards, nxLoc, nyLoc)

This sub drags a block of cards to the location nxLoc, nyLoc along a straight line from their current location. It can be used to return a block of cards to their original location if your game determines the user has dragged them somewhere they shouldn't be.

Other Dragging Functions

DRAGGING

Although dragging is made easier using **QCARD.DLL**, it is still a complex operation and one that will prove a little tough when you first try to implement it. Still, if your application is well organized and thought out, you will be able to include dragging operations with no problems.

Related Topics:

A Simple Single Drag Example
Avoiding Problems

Contents

A Simple Single Drag Example

When dragging cards, you will need to provide code for three events in relation to the Form you are working with. These are the MouseDown, MouseMove and MouseUp events. In the MouseDown event, you will initialize the drag operation. In the MouseMove event, you will carry out the drag operation. In the MouseUp event, you will end the drag operation. Of course, these events are happening all the time as your application is running, so you will need a switch to indicate whether a drag is in progress or not. For this purpose, create a Shared Integer variable in your General section which you can set to the Boolean values of TRUE and FALSE as your application runs:

```
Dim Shared bDragging
```

Initially, in your Form's Load procedure, this should be set to FALSE. In this simple example, begin by dealing a single card on your form:

```
DealCard Form1.hWnd, 1, 10, 10
```

This will deal the Ace of Clubs at location 10, 10 on your form. In response to the MouseDown event, we need to determine if the mouse is currently on the card or not. If it is, we can set the bDragging switch to TRUE. If not, we need to cancel the drag operation by calling the AbortDrag sub. The following code handles the MouseDown event:

```
Dim nSourceCard as Integer
nSourceCard = InitDrag(Form1.hWnd, x, y)
if nSourceCard = 0 Then
        AbortDrag
Else
        bDragging = TRUE
End If
```

The InitDrag function does several things. First, it takes the x, y mouse coordinates you pass it, and it looks through all the cards in the deck to see if any card lies underneath that location. If there is a card at that location and its IsBlocked and Disabled properties are both FALSE, it returns the number of that card. We can assign this value to the nSourceCard variable. In this case, this value will be 1, since that is the only card we have dealt on our form. If the InitDrag function cannot find a card at that mouse location, it will return a value of 0. In this case, we will need to abort the drag operation. Always follow up an InitDrag call with either an AbortDrag call or an EndDrag call. One of the things that InitDrag does is capture all mouse movements. You need to release the mouse capture by calling either AbortDrag or EndDrag, otherwise your application will effectively lock up your system since it is collecting all mouse information and directing it only to itself.

If no card is selected, we can abort the drag right here. If one is selected, we will release the mouse in the MouseUp event procedure. In the MouseMove event procedure, we need to test whether or not a drag is in progress. If it is, we will carry out the drag, if not, we don't need to do anything. Here is the MouseMove procedure:

```
If bDragging = TRUE Then
        DoDrag Form1.hWnd, x, y
```

```
End If
```

The DoDrag sub moves the card to its new location and updates its X and Y properties accordingly. The card which moves is the one selected by the InitDrag function. If bDragging = FALSE, nothing happens.

In the MouseUp event procedure, we need to end the drag if one is in progress. Here is where we will call the Endrag function, thereby releasing the mouse capture:

```
Dim nDestCard As Integer
If bDragging   = TRUE Then
        nDestCard = EndDrag(Form1.hWnd, x, y)
        bDragging = FALSE
End If
```

The EndDrag function does several things. Most importantly, as mentioned, it releases the mouse capture so mouse information can once again go to other applications other than this one. It also relocates the card to its new location and updates its X and Y properties accordingly. Finally, it checks to see if any other card lies beneath the card that has just been dragged and dropped. If there is a card that you have just covered over and that card's IsBlocked property is FALSE, the EndDrag function will return the number of that card. We assign that value here to the nDestCard variable. If there is no such card beneath the just dragged card, the function returns 0.

In this example, we declared the two variables nSourceCard and nDestCard as local to their respective Subs. In a real application, you would declare them as Shared or Global so you could carry out some comparison and testing on them. Since we now know the number of the Source card and the number of the Destination card, we could test whether or not this is a valid drag. In a game like Windows Solitaire, for example, where you can place a card of one less value on top of a card of the opposite color, some of the testing might look like this:

```
Dim nSourceColor As Integer
Dim nSourceValue As Integer
Dim nDestColor As Integer
Dim nDestValue As Integer
Dim bValidDrag As Integer
nSourceColor = GetCardColor(nSourceCard)
nDestColor = GetCardColor(nDestCard)
nSourceValue = GetCardValue(nSourceCard)
nDestValue = GetCardValue(nDestCard)
If nSourceColor <> nDestColor And nSourceValue = nDestValue - 1 Then
        bValidDrag = TRUE
Else
        bValidDrag = FALSE
End If
```

QCARD.DLL provides a nice little routine for handling invalid drags. In your MouseDown event procedure, you can save the original location of the Source Card before dragging it. Declare two Shared

or Global variables called OldX, and OldY, and assign them as follows in your MouseDown procedure:

OldX = GetCardX(nSourceCard)
OldY = GetCardY(nSourceCard)

Then, in your MouseUp event procedure, if you determine through comparison that this Source Card does not really belong on this Destination Card, then you can send it back to where it came from by calling:

ReturnDrag Form1.hWnd, nSourceCard, OldX, OldY

The ReturnDrag sub will automatically drag a card from its current location to the x, y location specified. It drags the card along a nice straight line. You may also find the ReturnDrag sub useful in other situations.

More On Dragging

Avoiding Problems

To avoid problems, just think of the cards in your game as being real cards in a three dimensional sense. At any given time, some of the cards in your game will be standing alone in the open while others will be blocked in a pile with other cards. Problems arise when you allow the user to do a Single Drag operation on a card which is covered over by another card. This produces some unsightly video results. Instead, you must think in terms of "Last Card On, First Card Off". In doing this, you must dynamically maintain the IsBlocked status of all the cards in your game. Only free standing cards in the clear should have an IsBlocked status of FALSE. All other cards should have an IsBlocked status of TRUE.

Although you do not have to use arrays in your game, arrays representing piles of cards makes things much easier to maintain. As cards are dragged from one pile to another, you can update the IsBlocked properties for the affected cards in each array. For example, if dragging a single card from the bottom of one row of cards to the bottom of another row, you would unblock the last card which was freed up in the original row, block the last card in the destination row which is now covered by the new card, remove the card from the original row's array of members, and add the new card to the destination row's array of members. If you think the problem out carefully, you can arrange the data in your game so this process is handled automatically as your game executes. See the demo program for an example of one way to do this.

More On Dragging

Dragging a Block of Cards

QCARD.DLL allows your application to drag a block of cards from one location to another. To carry out this operation, several guidlines must be followed. First, the cards being block dragged must all have the same x location. That is, their left sides must be aligned in column fashion. Second, their y locations must increase by the same amount. That is, they must all be evenly spaced vertically. The default setting for this OffSet is 16. You can change that value with the SetOffSet sub if you wish. Third, the last card in block drag operation is assumed to be the bottom card in a row. If you try to pull the middle three cards from a row and drag them, you will not get the result you want. **QCARD.DLL** drags blocks of cards the same way Windows Solitaire does. When the user presses the mouse button down on the top of a card, that card becomes the first card in the block drag. The last card in the block drag in always the last card in that row.

Related Topics:

Block Dragging Example
Avoiding Problems

Contents

Block Dragging Example

Like Single Dragging, Block Dragging requires you to provide code for three event procedures for your form: MouseDown, MouseMove and MouseUp. In the MouseDown event procedure, you initialize the drag operation. In the MouseMove event procedure, you carry out the drag operation. In the MouseUp event procedure, you end the drag operation.

Related Topics:

The MouseDown event procedure
The MouseMove event procedure
The MouseUp event procedure

More On Dragging a Block of Cards

The MouseDown Event Procedure

Begin the drag procedure by calling the function InitDrag. This function searches through the deck to determine if any card whose IsBlocked property is FALSE is currently located at the mouse's x, y location. If it finds one, it returns the number of that card. This would successfully initialize a single drag operation. If it cannot find an unblocked card at that location, it will determine if the mouse is in the top 16 (or user defined) pixels of any other card, blocked or not. If it finds one, it returns the number of that card. This would initialize a block drag.

In the MouseDown event procedure then, a little testing of the selected card's IsBlocked status will tell us if we are about to do a block drag or a single drag:

```
Dim nStatus As Integer
nSourceCard = InitDrag(Form1.hWnd)
If nSourceCard = 0 Then
        ' no card selected
        AbortDrag
Else
        nStatus = GetCardBlocked(nSourceCard)
        If nStatus = FALSE Then
                bSingleDragging = TRUE
        Else
                bBlockDragging = TRUE
        End If
End If
```

The Other Mouse Event Prodecures

The MouseMove event procedure

To drag the block of cards in the MouseMove event you use the sub BlockDrag. You must give this sub a list of the cards you want to drag and the total number of cards in the list. The list itself should be in the form of an array, and you can pass the array to the DLL by referencing its first element:

BlockDrag Form1.hWnd, CardList(0), nNumCards, x, y

By giving the DLL the first element in the array, the DLL will have access to all the elements in the array. If you declare a shared array for this purpose, you can use the Redim statement to resize the array to the correct size when you need it. You can then just fill the array with the numbers of the cards you want to drag, beginning with the top card in the block (the Source Card) and ending with the bottom card of the block. See the demo program for a better idea of how this is done.

The Other Mouse Event Prodecures

The MouseUp event procedure

The block drag operation is completed in the MouseUp event procedure by calling the EndBlockDrag function. Again, you must pass this function an array containing the numbers of the cards being dragged and the total number of cards in the list. Since this information does not change between the MouseMove event and the MouseUp event, you can reuse the same array for both, provided the array was declared as Shared:

nDestCard = EndBlockDrag(Form1.hWnd, CardList(0), nNumCards, x,y)

The function returns the number of any card whose IsBlocked property is FALSE which lies beneath the first (Source) card being dragged in the block. You now know the SourceCard and the Destination card and you can go ahead and make comparisons to determine if the drag is valid or not. If it is, you will have to relocate all the cards in your CardList array to their new pile. See the demo program for a better idea of how this might be done.

The Other Mouse Event Prodecures

Avoiding Problems

Problems in block dragging can usually be traced to the array you are trying to pass the BlockDrag sub. If the array is not properly filled beginning with its first element (0), you will end up with strange results. If you pass the sub the wrong number of cards to drag you will also get strange results. Ensure that you are using Redim and that you are filling the array properly if you run into problems. The example program shows a how to do this correctly. Steal some code from there if you can't get things to work just right.

More On Dragging a Block of Cards

About QCard.DLL

QCard.DLL was written by Stephen Murphy. If you have any comments about the project or would like to contact the author for any reason, I can be reached at: CompuServe 70661,2461.

Special Thanks:

Special thankyou to Daniel Di Bacco for his great cardbacks and jokers. Look for his own projects on a service near you. He does excellent work!

Thanks to David Shank who did the Help file for the first version of **QCard.DLL**. His super work inspired me to write the Help file for this version.

Contents