CardLib Reference

CardLib User Guide

The following document describes the various C++ classes which make up the CardLib game library.

class Card

Function Prototype	Description
`int Suit()`	Returns the suit of the card [0..3]
`int LoVal()`	Returns the value of the card, Aces Low, [1..13]
`int HiVal()`	Returns the value of the card, Aces High, [2..14]
`int Idx()`	Returns the unique card index, [0..51]
`bool FaceUp()`	Returns if card is face-up
`bool FaceDown()`	Returns if card is face-down
`void SetFaceUp(bool)`	Sets the face direction
`bool IsBlack()`	Returns if card is Clubs or Spades
`bool IsRed()`	Returns if card is Diamonds or Hearts

class CardStack

Function Prototype	Description
`void NewDeck()`	Allocates a new 52-card deck
`int NumCards()`	Returns number of cards in stack
`void Shuffle()`	Shuffles the cards in the stack
`void Clear()`	Removes all cards from the stack
`void Reverse()`	Reverses the order of cards
`void Push(Card card)`	Pushes the specified card

`void Push(CardStack &cardstack)`	Pushes the specified CardStack
`Card Pop()`	Pops the top-most card
`CardStack Pop(int items)`	Pops the top-most n-cards
`Card Top()`	Peeks the top-most card, without removing it
`CardStack Top(int items)`	Peeks the top-most n-cards, without removing them
`Card RemoveCard(size_t index)`	Removes the card at specified index
`void InsertCard(size_t index, Card card)`	Inserts a card at specified index

class CardButton

Function Prototype	Description
`void SetStyle(UINT uStyle)`	Sets the button style (push-button, label etc).
`UINT GetStyle()`	Returns the button style.
`void SetText(TCHAR *fmt, ...)`	Set the button text.
`void SetFont(HFONT hFont)`	Set the button type-face.
`void SetPlacement( ... )`	* Aligns the button within it's parent CardWindow.
`void SetForeColor(COLORREF col)`	Set the button text colour.
`void SetBackColor(COLORREF col)`	Set the button background colour.
`void Move(int x, int y, int w, int h)`	Re-position the button.
`void Show(bool fShow)`	Show / Hide the button.
`void Redraw()`	Force the button to repaint itself.
`void Id()`	Return the button ID.

* See `CardRegion::SetPlacement` for more details.

{short description of image}

class CardRegion

Function Prototype	Description		
`void SetBackColor(COLORREF cr)`	Set a new colour-scheme for the cardgame.		
`void SetCardStack(const CardStack &cs)`	Assign a new CardStack.		
`const CardStack & GetCardStack()`	Retrieve the current CardStack.		
`bool SetThreedCount(int count)`	Control how high a card-stack can appear by grouping cards together. (Useful for decks with large numbers of cards).	`void SetOffsets(int x, int y)`	Control the direction that stacked cards take.
`void SetPos(int x, int y)`	Position the card region.		
`void Show(bool fShow)`	Show / Hide the card region.		
`bool IsVisible()`	Is the card region visible?		
`void SetEmptyImage(UINT uImage)`	Set the image used to represent an empty card stack. Use `CS_EI_NONE` or `CS_EI_SUNK`.		
`void SetBackCardIdx(UINT uBackIdx)`	Set the card bitmap used for the card-backs.		
`void SetPlacement( ... )`	Align the card region within its parent CardWindow.		
`void Redraw()`	Force the card region to repaint itself.		
`void SetFaceDirection( ... )`	Control how cards are placed on the stack (face-down, face-up, or a combination of the two).		
`UINT GetFaceDirection( ... )`	Return the current face-direction rule.		
`void Flash(int count, int timeout)`	Make the stack flash a number of times.		
`void StopFlash()`	Stop the stack flashing.		
`int Id()`	Return the ID.		
`bool PlayCard( ... )`	Move the specified number of cards with index [0..51] to the specified CardRegion.		

`bool MoveCard( ... )`	Move the specified number of cards from the top of the stack.
CardStack wrappers (for the internal stack).	
`int NumCards()`	Returns the number of cards in the internal CardStack.
`int NewDeck()`	Allocates a new 52-card deck.
`int Shuffle()`	Shuffles the cards in the stack.
`int Clear()`	Removes all cards from the stack.
Event Callbacks (see user guide for details).	
`SetDragRule`	Set the dragging rule for the CardStack.
`SetDropRule`	Set the drop rule.
`SetClickProc`	Mouse-button click event.
`SetDblClickProc`	Mouse-button double-click event.
`SetAddCardProc`	Post-add event callback.
`SetRemoveCardProc`	Post-remove event callback.

The following list describes a few of the functions in the above table which need more careful explanation.

CardRegion::SetPlacement

```
void SetPlacement(UINT xJustify, UINT yJustify,
                  int xAdjust, int yAdjust)
```

Controls the alignment of the CardRegion within it’s CardWindow.

xJustify specifies the horizontal alignment. Possible values are:

`CS_XJUST_NONE`	Default alignment (left-aligned).
`CS_XJUST_CENTER`	Aligned to the center of the CardWindow.
`CS_XJUST_RIGHT`	Aligned to the right side of the CardWindow.

yJustify specifies the vertical alignment. Possible values are:

`CS_YJUST_NONE`	Default alignment (aligned to the top of the window).
`CS_YJUST_CENTER`	Aligned to the middle of the CardWindow.
`CS_YJUST_RIGHT`	Aligned to the bottom of the CardWindow.

xAdjust specifies a horizontal offset (or border) to use. This offset is applied when the card-region is aligned to the center or right of the window. A positive value offsets the card-region to the right, a negative value offsets to the left.

yAdjust specifies a vertical offset (or border) to use. This offset is applied when the card-region is aligned to the middle or bottom of the window. A positive value offsets the card-region downwards, a negative value offsets the card-region upwards.

CardRegion::SetFaceDirection

```
void SetFaceDirection(UINT uDirType, int nOption)
```

Controls how cards are placed onto the card-stack. Cards can be either face-up, face-down, or a combination of the two.

`uDirType` specifies what the face-direction is. Possible values are:

`CS_FACE_UP`	All cards will be face-up when added to the stack.
`CS_FACE_DOWN`	Cards will be face-down.
`CS_FACE_DOWNUP`	The bottom most `nOption` cards will be face-down, all other cards on top of these will be face-up.
`CS_FACE_UPDOWN`	The bottom most `nOption` cards will be face-up, all other cards on top of these will be face-down.
`CS_FACE_ANY`	Cards can be either face-up or face-down.

nOption is used in conjunction with `CS_FACE_DOWNUP` and `CS_FACE_UPDOWN` . This value controls the face-direction of the bottom-most nOption cards.

For example, assume that `uDirType` is `CS_FACE_DOWNUP` and `nOption` is 3. This means that the bottom 3 cards will always be face-down, and any cards above these 3 will always be face-up.

CardRegion::GetFaceDirection

```
UINT GetFaceDirection(int *pnOption)
```

Returns the current face-direction rules.

This function returns the direction-type variable, and stores the option in the address specified by `pnOption` .

CardRegion::Flash

```
void Flash(int count, int timeout)
```

Causes the card-region to flash on and off. This is achieved by hiding and then showing the card-region multiple times.

count specifies how many times the card-region will be hidden and shown again.

timeout specifies the duration of each stage of animation, in milliseconds.

CardRegion::StopFlash

```
void StopFlash()
```

Stops immediately any stack-flashing that might be active.

CardRegion::PlayCard

```
bool PlayCard(CardRegion *pDest, int val, int num)
```

Moves **num** cards of face-value **val** to the specified card-region.

val specifies the face-value of the card, using “Aces High” numbering [2..14].

num identifies the number of these cards to move.

PlayCard returns **true** on success, **false** on failure.

CardRegion::MoveCard

```
bool MoveCard(CardRegion *pDest, int nNumCards, bool fAnimate)
```

Moves the specified number of cards from the top of the current stack, to the specified destination stack.

nNumCards specifies how many cards are to be moved.

fAnimate is a boolean value, which specifies if the cards are to be animated when they are moved, or (when fAnimate is false) moved without any animation.

MoveCard returns **true** on success, **false** on failure.

{short description of image}

class CardWindow

Function Name	Description
`Create`	Create a physical card window.
`CreateButton`	Create a button.
`CreateRegion`	Create a card-region.
`CardButtonFromId`	Return the card-button with specified ID.
`CardRegionFromId`	Return the card-region with specified ID.
`DeleteButton`	Deletes the specified button.
`DeleteRegion`	Deletes the specifies card-region.
`DeleteAll`	Deletes ALL regions, buttons and drop-zones
`EmptyStacks`	Empty all card-stacks in the card-window.
`Redraw`	Force the card-window to repaint itself.
`Update`	Update the position of all card-stacks and buttons.
`DistributeStacks`	Position a range of card-stacks in an aligned manner.
`SetResizeProc`	Specify a callback function for window resizes.
`GetWidth`	Return the width of the card-window.
`GetHeight`	Return the height of the card-window.
`SetBackColor`	Set the current background colour.
`GetBackColor`	Return the current background colour.
`SetBackCardIdx`	Specify which bitmap is to be used as the card-backs.
`SetBackImage`	Set the bitmap to be used as window background.
`RegisterDropZone`	Creates a new DropZone area.
`DeleteDropZone`	Deletes the specified DropZone.

The CardWindow functions are not as obvious as the previous classes, so following section describes each CardWindow member function separately.

CardWindow::Create

```
BOOL Create(HWND hwndParent, DWORD dwExStyle, DWORD dwStyle,
                int x, int y, int width, int height)
```

Creates a physical window which will form the basis for a card-game.

CardWindow::CreateButton

```
CardButton *CreateButton(int id, TCHAR *szText, UINT uStyle, bool fVisible,
                             int x, int y, int width, int height)
```

Create a button within the window.

id specifies a unique integer ID for the button.

szText is the address of a null-terminated string, which contains the button text.

uStyle specifies the button styles. The following styles are supported.

`CB_STATIC`	Creates a static text label (default)
`CB_PUSHBUTTON`	Creates a normal push-button
`CB_ALIGN_CENTER`	Centers the button text (default)
`CB_ALIGN_LEFT`	Left-aligns the button text
`CB_ALIGN_RIGHT`	Right-aligns the button text

fVisible specifies whether or not the button is visible.

x, y specifies the position of the button.

width, height specifies the dimensions of the button.

CardWindow::CreateRegion

```
CardRegion *CreateRegion(int id, bool fVisible,
                             int x, int y, int xoffset, int yoffset)
```

Create a card-stack region within the window.

id specifies a unique integer ID for the region.

fVisible specifies whether or not the region is visible.

x, y specifies the position of the region.

xoffset, yoffset specifies the direction that cards take when placed on the stack.

CardWindow::CardButtonFromId

```
CardButton *CardButtonFromId(int id)
```

Returns a pointer to the CardButton object with the specified ID.

CardWindow::CardRegionFromId

```
CardRegion *CardRegionFromId(int id)
```

Returns a pointer to the CardRegion object with the specified ID.

CardWindow::DeleteButton

```
bool DeleteButton(CardButton *pButton)
```

Destroy the specified button. Do not use the `delete` operator to delete a button - use this function instead. Note that the CardWindow will automatically delete all buttons when it is deleted itself.

CardWindow::DeleteRegion

```
bool DeleteRegion(CardRegion *pRegion)
```

Destroy the specified region. Do not use the `delete` operator to delete a region - use this function instead. Note that the CardWindow will automatically delete all region when it is deleted itself.

CardWindow::DeleteAll

```
bool DeleteAll()
```

Destroys every CardRegion, CardButton and drop-zone. All pointers to these objects become invalid.

CardWindow::EmptyStacks

```
void EmptyStacks()
```

Empty ALL card-stacks currently in the card-window.

CardWindow::Redraw

```
void Redraw()
```

Force the card-window to repaint itself.

CardWindow::Update

```
void Update()
```

Force the card-window to reposition all stacks and buttons.

CardWindow::DistributeStacks

```
bool DistributeStacks(int nIdFrom, int nNumStacks,
                      UINT xJustify, int xSpacing, int nStartX)
```

Physically distribute the specified card stacks. The stack with the specified id (`nIdFrom`) is used as the starting point., and the `nNumStacks` created after this stack will have their positions modified.

xJustify specifies the type of alignment to use:

`CS_XJUST_NONE`	No alignment (left-aligned, default)
`CS_XJUST_CENTER`	Center alignment
`CS_XJUST_RIGHT`	Right-aligned

xSpacing specifies how far apart each stack is to be placed.

nStartX specifies the starting x-coordinate to place the series of card-stacks.

CardWindow::SetResizeProc

```
void SetResizeProc(pResizeWndProc proc)
```

Specify a callback function, which is executed whenever the CardWindow's size is altered. The callback function must have the following prototype:

```
void CARDLIBPROC ResizeWndProc(int width, int height)
```

CardWindow::GetWidth

```
int GetWidth()
```

Returns the width (in pixels) of the card-window.

CardWindow::GetHeight

```
int GetHeight()
```

Returns the height (in pixels) of the card-window.

CardWindow::SetBackColor

```
void SetBackColor(COLORREF cr)
```

Set the current background colour.

CardWindow::GetBackColor

```
COLORREF GetBackColor()
```

Returns the current background colour.

CardWindow::SetBackCardIdx

```
void SetBackCardIdx(UINT uBackIdx)
```

Specify which card-bitmap is to be used as the card-back.

uBackIdx must be in the range [53..68] inclusive. Use one of the ecbxxx constants defined in CardLib.h

CardWindow::SetBackImage

```
void SetBackImage(HBITMAP hBitmap)
```

Specifies a bitmap to be used as the card-window background. The CardWindow does not assume ownership of the bitmap.

hBitmap must be a valid handle to a bitmap object, or **NULL** to remove the background bitmap.

CardWindow::RegisterDropZone

```
bool RegisterDropZone(int id, RECT *rect, pDropZoneProc proc)
```

Creates a DropZone within the card-window.

id specifies a unique ID for the DropZone. This id must NEVER be -1.

rect is the address of a RECT structure, which specifies the coordinates of the DropZone.

proc is the address of a callback function, which is executed whenever a card is dropped inside the rectangular area.

A DropZone is a special area within a CardWindow which cards can be dropped on. Even if a card-stack is within the area, the cards will not be dropped on it.

The specified function callback is called whenever one or more cards are dropped within the zone. This function must have the following prototype:

```
int CARDLIBPROC DropZoneProc(int dzid, const CardStack &cards)
```

The callback function returns the ID of the stack to drop the specified cards onto. Using this mechanism provides complete control over which stack cards are dropped.

The function must return a valid ID of a CardRegion to send the cards to, or use a value of `CS_DROPZONE_NODROP` to prevent the cards from being dropped.

CardWindow::DeleteDropZone

```
bool DeleteDropZone(int id)
```

Deletes the DropZone with the specified ID.

Updated: December 04, 2001