General Description

Document description: Overview over the *Functions, Events* and *Properties* of the EasyX control.

Version: 1.2

Date: 01-10-98

Modifications from version 1.1

Added the following function descriptions:

DrawLine
DrawCircle
Plot
GetDisplayModes
GetStaticFrequency
SetStaticFrequency
GetStreamingFrequency
GetStaticByteLenght
SetCurrentStaticPosition
GetCurrentStaticPosition

Added the following Event description

DisplayModes

Modifications from version 1.0:
Added the following function descriptions:

IsPaletteMode
IWCreateFakeBackBuffer
IWDrawSprite
IWDrawToBackBuffer
IWFillSurface
IWInitDirectDraw
IWResetSurface
IWDrawFakeToPrimary

Function reference

InitDirectDraw (Width as Integer, Height As Integer, BitDepth As Integer) As Long

Description
Initializes DirectDraw® and sets up the screen properties.

Parameters
Width: The width of the screen in pixels. Must be one of the supported modes.

Height: The height of the screen in pixels. Must be the one of the supported modes.

BitDepth: The Bit depth of the screen in pixels. This is the number of bits which are used to describe the color of each pixel.

Remarks
This function initializes direct draw and sets the screen to passed values. The *Window* property of the control must be set before this function is called. This function will do some unusual things to the window, which are set in the *Window* property. It is therefore not recommended to use this window for anything else.

Return values

EX_OK: Everything was set and initialized
EX_UNSUPPORTED: The display card does not support direct draw
EX_UNSUPPORTEDMODE: The display card does not support the selected mode
EX_OUTOFMEMORY: There is not enough memory available to initialized direct draw
EX_UNDEFINED: An unknown or unrecoverable error occurred

ReleaseSurfaces() As Long

Description

This function release all the secondary surfaces created from the LoadBitmapFile function.

Parameters

None…

Remarks

This function will release all the memory consumed by the created surfaces. Use this function to restore used resources. The surfaces cannot be recovered when this function has been called.

Return values

EX_OK: all the surfaces was released.
EX_UNDEFINED: The Surfaces could not be released.

LoadBitmapFile(FileName As String, ColorKey As Long,) As Long

Description

This function creates a secondary surface from the bitmap file supplied.

Parameters

FileName: The filename of the bitmap.

ColorKey: The color key which will be used as the transparent color of the surface. If 0 is passed the pixel in the upper left corner of the bitmap will be the transparent color.

Remarks

This function will load the bitmap into a surface, which can be used to define sprites or used to blit into the primary surface. The bitmap should be a bitmap with a format equal to the primary surface (bit depth), if not some color variation will be apparent.
Bitmaps which are larger than the primary surface can not be loaded into a surface.
It is always a good idea to moderate the size of bitmap surfaces i.e. do not create a surface for each single sprite needed, collect the sprites into bigger bitmaps and create surfaces from these. Do not create very big bitmaps and load them into surfaces, since the function will first try to load the bitmap into video memory, and then into system memory which is a great deal slower. So if the bitmap is just a tiny bit to big to load into video memory, it will load into system memory, and hence will be slower to use, while there is still some space left in video memory.

Return values

>= 0: The function succeeded, and an index to the created surface is returned.
EX_SURFACECREATIONFAILED: There was error in creating the surface.
EX_NOBITMAP: There was no bitmap file supplied
EX_COLORSETFAILED: The color key could not be set.
EX_UNDEFINED: An unknown or unrecoverable error occurred

FlipSurface() As Long

Description

This function flips the back buffer and primary surface.

Parameters

None…

Remarks

This function will flip whatever is blitted or drawn into the back buffer onto the primary surface and thereby showing it on the screen. This function must always be called before any color fills or blit operations are shown on the screen.

Return values

EX_OK: The flip was performed
EX_UNDEFINED: The flip could be made.

MakeSprite(
UpperX As Integer,
UpperY As Integer,
LowerX As Integer,
LowerY As Integer,
Surface As Long
) As Long

Description

This function defines a sprite, which can be used with the DrawSprite function.

Parameters

UpperX: The upper x coordinate of the sprite on the surface.

UpperY: The upper Y coordinate of the sprite on the surface.

LowerX: The lower x coordinate of the sprite on the surface.

LowerY: The lower Y coordinate of the sprite on the surface.

Surface: The surface where the sprite is loaded. This must be a valued returned from the LoadBitmapFile function.

Remarks

This function defines and creates a sprite, which can be used with the DrawSprite function. If all the coordinates are 0, then the whole surface will be defined as a sprite. You can create sprites which overlap or a equal to each other.

Return values

>= 0: The sprite was created and the value is the index to the sprite.

EX_BADRECT: The Coordinates were not acceptable
EX_NOSURFACE: The surface index, which was supplied, did not match a created surface.
EX_UNDEFINED: An unknown error occurred

DrawSprite(
UpperX As Integer,
UpperY As Integer,
Width As Integer,
Height As Integer,
SpriteNumber As Integer
) As Long

Description

Draws a sprite into the back buffer.

<u>Parameters</u>

UpperX: The top X coordinate of the sprite.

UpperY: The top Y coordinate of the sprite.

Width: The width of the sprite being drawn.

Height: The height of the sprite being drawn.

SpriteNumber: The sprite being drawn.

<u>Remarks</u>

This function draws a sprite created with the *MakeSprite* function. If the coordinates are 0, the whole sprite will be drawn and stretched to fit the entire back buffer. If the w*idth* and *height* arguments are not the same size as the original created sprite, the sprite will be stretched or shrink to fit the coordinates.

Note: This function is very vulnerable to errors, and will create a run-time error "Automation Error", if something is wrong with the sprite and back buffer.

<u>Return values</u>

EX_OK: The sprite was drawn and the specified coordinates.
EX_UNDEFINED: An unknown error occurred.

ReleaseSprites() As Long

<u>Description</u>

Releases all the created sprites.

<u>Parameters</u>

None…

<u>Remarks</u>

This function will release all the sprites created with the *MakeSprite* function. All the sprites are unrecoverable after this operation.

<u>Return values</u>

EX_OK: The sprites were released.
EX_UNDEFINED: An unknown error occurred.

FillSurface(Color As Long, Surface As Long) As Long

<u>Description</u>

This function fills a surface with a specified color.

Parameters

Color: The Color value or index into a palette.

Surface: The surface to fill with the specified color.

Remarks

If the current screen mode is a palette mode then the color value will be a index in the surface's palette. If the screen mode is a non-palette mode then the value will be a direct color value of the form BGR (NOT RGB). You can use the RGB() function in Visual Basic to make such a color value, just keep in mind that the Red and Blue values are reversed.

Return values
EX_OK: The operation completed with no errors
EX_NOSURFACE: The surface specified does not exits
EX_UNDEFINED: An unknown error occurred

GetDeviceContext(Surface As Long) As Long

Description

Retrieves the device context of the specified surface.

Parameters

Surface: The surface which a device context should be retrieved.

Remarks

This function returns the device context of the specified surface. Use this function only when absolutely necessary, and release it immediately after use with the *ReleaseDeviceContext()* function.

Return values
> 0: The device context
EX_NOSURFACE: The surface specified was not valid.
EX_UNDEFINED: An unknown error occurred.

ReleaseDeviceContext(hdc As Long, Surface As Long) As Long

Description

Releases a device context obtained from *GetDeviceContext()* function.

Parameters

Hdc: The device context to be released.

Surface: The surface which owns the device context

Remarks

Use this function to release device contexts, obtained from the *GetDeviceContext() function.*

Return values

EX_OK: The device context was released
EX_NOSURFACE: The surface specified was not valid.
EX_UNDEFINED: An unknown error occurred.

SetText(
Surface As Long,
BackColor As Long,
Color As Long,
Height As Long,
FontFace As String,
) As Long

Description

Sets a surface's text attributes.

Parameters

Surface: The surface which text attributes will be set

BackColor: The back color of the text

Color: The color of the text

Height: The height of the text in pixels

FontFace: The font face of the text. ("Verdana", "Times New Roman" etc.)

Remarks

Use this function to set the text attributes of a given surface. If this function is not called before the *PrintText()* function, the system default attributes will be used.

Return values

EX_OK: The attributes were set

PrintText(
Surface As Long,
Text As String,
X As Long,
Y As Long,
Width As Long,
Height As Long,
) As Long;

Description

This function prints a text into to the specified surface.

Parameters

Surface: The surface to print the text.

Text: The text to print

X: The upper X coordinate of the text starting point

Y: The upper Y coordinate of the text starting point

Width: The width of the text printing area

Height: The Height of the text printing area

Remarks

This function prints the >Text< parameter into the specified surface. The coordinates and the width and height parameters are used to specify the text area where the text will be printed. The text will be automatically wrapped to fit the width of this text box. Use the function *SetText()* to set the attributes of the text.

Return values
EX_OK: The text was printed
EX_NOSURFACE: The surface specified was not valid
EX_UNDEFINED: An unknown error occurred

InitializeSound() As Long
Description

Initializes DirectSound®

Parameters

None…

Remarks

This function initializes DirectSound® for use. This function must be called before any other sound related functions are called. The *Window* property of the control must be set before this function is called.

Return values

EX_OK: The sound was correctly initialized
EX_NOHWND: The *Window* property was not correctly set.
EX_SOUNDINITFAILED: DirectSound® could not be initialized

CreateStreamingSound(FileName As String,) As Long

Description

Creates a streaming sound.

Parameters

FileName: The filename of the sound wav file

Remarks

This function creates a streaming sound from a file. The format of the file must be PCM WAV format. DirectSound® currently supports no other formats.

Return values

>= 0: The sound was created and the index to it was returned.
EX_UNDEFINED: An unknown error occurred.

PlayStreamingSound(Index As Long, Loop As Long) As Long

Description

Plays a streaming sound

Parameters

Index: The index of the streaming sound file.

Loop: Loop option:
0 = No Loop
1 = Loop

Remarks

This function plays a streaming sound created by the *CreateStreamingSound()* function.

Return values

EX_OK: The playback has started
EX_INVALID: An invalid index was provided

StopStreamingSound(Index As Long) As Long

Description

Stops the playback of a streaming sound.

Parameters

Index: the index of the streaming sound being played

Remarks

This function will stop a streaming sound currently in playback mode.

Return values

EX_OK: The sound was stopped
EX_INVALID: the index supplied was invalid

CreateStaticSound(FileName As String) As Long

Description

Creates a static sound.

Parameters

FileName: The file name of the sound file.

Remarks

This function creates a static sound from a file. The format of the file must be PCM WAV format. DirectSound® currently supports no other formats.

Return values
>= 0: The creation succeeded and an index to the sound was returned.
EX_BADSOUNDFILE: The format of the file was invalid
EX_UNDEFINED: An unknown error occurred

DuplicateStaticSound(Index As Long) As Long

Description

Duplicates a previous created static sound.

Parameters

Index: The index of the static sound to be duplicated

Remarks

This function duplicates a static sound previously created. By doing so, the same sound can be played in an overlapping mode. Do use this function instead of creating a new sound, in order to play the same sound, since this function will only create new pointers to the same sound buffer, and a new sound buffer. Use the *PlayStaticSound()* to play the duplicated sound. There is no limit (other than the usual system resource limit) in the number of times a sound buffer can be copied.

Return values
>= 1: The index of the new sound.
EX_SOUNDNOTINITIALIZED: DirectSound® was not initialized
EX_UNDEFINED: An unknown error occurred.

PlayStaticSound(Index As Long, Loop As Long) As Long

Description

Plays a static sound

Parameters

Index: The index of the static sound

Loop: Loop option:
0 = No Loop
1 = Loop

Remarks

This function plays a static sound created with the *CreateStaticSound()* function.

Return values
EX_OK: The sound was played.
EX_UNDEFINED: An unknown error occurred

StopStaticSound(Index As Long) As Long

Description

Stops a static sound in playback-mode
Parameters

Index: The index of the static sound in playback.

Remarks

This functions stops a static sound in playback-mode.

Return values
EX_OK: The sound was stopped
EX_INVALID: An invalid index was supplied

SetStaticPan(Index As Long, PanValue As Long) As Long

Description

Sets the pan value of a static sound

Parameters

Index: The index of the sound file

PanValue: The pan value

Remarks

This function sets the pan value of a static sound. The value ranges between 10,000 and –10,000. A value of zero means that both left and right are played at full volume, whereas a value of –10,000 means that the right channel is silenced (or attenuated by 100 dB), and vice versa.

Return values
EX_OK: The pan value was set.
EX_INVALID: An invalid index was specified
EX_INVALIDVALUE: The pan valued specified was less than -10,000 or more than 10,000.

SetStreamingPan(Index As Long, PanValue As Long) As Long

Description

Sets the pan value of a streaming sound

Index: The index of the sound file

PanValue: The pan value

Remarks

This function sets the pan value of a streaming sound. The value ranges between 10,000 and –10,000. A value of zero means that both left and right are played at full volume, whereas a value of –10,000 means that the right channel is silenced (or attenuated by 100 dB), and vice versa.

Return values
EX_OK: The pan value was set.
EX_INVALID: An invalid index was specified
EX_INVALIDVALUE: The pan valued specified was less than -10,000 or more than 10,000.

SetStaticVolume(Index As Long, Volume As Long) As Long

Description

Sets the volume of a static sound

Parameters

Index: The index of the sound

Volume: The volume of the static sound

Remarks

This function sets the volume of a static sound. The volume value ranges from 0 (no attenuation) to –10,000 (silence – 100 dB attenuation). The default volume will always be 0 (full volume), so this function can not amplify sounds.

Return values
EX_OK: The volume was set.
EX_INVALIDVOLUME: The volume value specified was invalid
EX_INVALID: The index specified was invalid

SetStreamingVolume(Index As Long, Volume As Long) As Long
Description

Sets the volume of a streaming sound

Parameters

Index: The index of the sound

Volume: The volume of the streaming sound

Remarks

This function sets the volume of a streaming sound. The volume value ranges from 0 (no attenuation) to –10,000 (silence – 100 dB attenuation). The default volume will always be 0 (full volume), so this function can not amplify sounds.

Return values
EX_OK: The volume was set.
EX_INVALIDVOLUME: The volume value specified was invalid
EX_INVALID: The index specified was invalid

GetStaticVolume(Index As Long) As Long

Description

Return the volume of a static sound

Parameters

Index: The sound to read the volume from

Remarks

This function retrieves the volume of an indexed sound. The value will be in the range of 0 t0 –10,000 (where 0 is max volume)

Return values
=>0 : The volume of the sound
101: An invalid index was supplied

GetStreamingVolume(Index As Long) As Long
Description

Return the volume of a streaming sound

Parameters

Index: The sound to read the volume from

Remarks

This function retrieves the volume of an indexed sound. The value will be in the range of 0 t0 –10,000 (where 0 is max volume)

Return values
=>0: The volume of the sound
101: An invalid index was supplied

GetStaticPan(Index As Long) As Long

Description

Return the pan value of an indexed sound

Parameters

Index: The index of the sound

Remarks

This function retrieves the pan value of a given sound. The value will be in the range of 10,000 to –10,000. *See SetStaticPan() function.*

Return values
The pan value of the sound

-10,001: An invalid index was supplied.

GetStreamingPan(Index As Long) As Long

Description

Return the pan value of an indexed sound

Parameters

Index: The index of the sound

Remarks

This function retrieves the pan value of a given sound. The value will be in the range of 10,000 to –10,000. *See SetStreamingPan() function.*

Return values
The pan value of the sound

-10,001: An invalid index was supplied.

InitDirectInput() As Long

Description

This function initializes DirectInput®.

Parameters

None…

<u>Remarks</u>

This function initializes DirectInput®. It must be called before any other DirectInput® input functions are called.

<u>Return values</u>
EX_OK: The initialization succeeded.
EX_DIRECTINPUTFAILED: The initialization failed.

CreateKeyboard() As Long

<u>Description</u>

This function creates the keyboard for use with the DirectInput® functions.

<u>Parameters</u>

None…

<u>Remarks</u>

This function creates the keyboard for use with the keyboard functions. This function must be called after the *InitDirectInput()* and before other keyboard functions are called.

<u>Return values</u>
EX_OK: The keyboard was successfully created.
EX_DIRECTINPUTDEVICEFAILED: The device could not be created.

GetKeyState(KeyCode As Long) As Long

<u>Description</u>

Returns the state of a specified key.

<u>Parameters</u>

KeyCode: A specified key.

The key values are as following (taken from the supplied module in the Visual Basic® examples)

EX_ESCAPE	= &H1
EX_1	= &H2
EX_2	= &H3
EX_3	= &H4
EX_4	= &H5
EX_5	= &H6
EX_6	= &H7
EX_7	= &H8
EX_8	= &H9
EX_9	= &HA
EX_0	= &HB

```
EX_MINUS       = &HC          ' - on main keyboard */
EX_EQUALS      = &HD
EX_BACK        = &HE          ' backspace */
EX_TAB         = &HF
EX_Q           = &H10
EX_W           = &H11
EX_E           = &H12
EX_R           = &H13
EX_T           = &H14
EX_Y           = &H15
EX_U           = &H16
EX_I           = &H17
EX_O           = &H18
EX_P           = &H19
EX_LBRACKET    = &H1A
 EX_RBRACKET   = &H1B
 EX_RETURN     = &H1C         ' Enter on main keyboard */
 EX_LCONTROL        = &H1D
 EX_A           = &H1E
 EX_S           = &H1F
 EX_D           = &H20
 EX_F           = &H21
 EX_G           = &H22
 EX_H           = &H23
 EX_J           = &H24
 EX_K           = &H25
 EX_L           = &H26
 EX_SEMICOLON        = &H27
 EX_APOSTROPHE       = &H28
 EX_GRAVE           = &H29     ' accent grave */
 EX_LSHIFT          = &H2A
 EX_BACKSLASH        = &H2B
 EX_Z           = &H2C
 EX_X           = &H2D
 EX_C           = &H2E
 EX_V           = &H2F
 EX_B           = &H30
 EX_N           = &H31
 EX_M           = &H32
 EX_COMMA            = &H33
 EX_PERIOD          = &H34     ' . on main keyboard */
 EX_SLASH           = &H35        ' / on main keyboard */
 EX_RSHIFT          = &H36
 EX_MULTIPLY         = &H37      ' * on numeric keypad */
 EX_LMENU           = &H38       ' left Alt */
 EX_SPACE           = &H39
 EX_CAPITAL         = &H3A
 EX_F1          = &H3B
 EX_F2          = &H3C
 EX_F3          = &H3D
 EX_F4          = &H3E
 EX_F5          = &H3F
 EX_F6          = &H40
 EX_F7          = &H41
 EX_F8          = &H42
```

```
EX_F9            = &H43
EX_F10           = &H44
EX_NUMLO            = &H45
EX_SCROLL           = &H46          ' Scroll Lock */
EX_NUMPAD7          = &H47
EX_NUMPAD8          = &H48
EX_NUMPAD9          = &H49
EX_SUBTRACT         = &H4A        ' - on numeric keypad */
EX_NUMPAD4          = &H4B
EX_NUMPAD5          = &H4C
EX_NUMPAD6          = &H4D
EX_ADD            = &H4E              ' + on numeric keypad */
EX_NUMPAD1          = &H4F
EX_NUMPAD2          = &H50
EX_NUMPAD3          = &H51
EX_NUMPAD0          = &H52
EX_DECIMAL          = &H53        ' . on numeric keypad */
EX_F11           = &H57
EX_F12           = &H58

EX_F13           = &H64         '                    (NEC PC98) */
EX_F14           = &H65           '                    (NEC PC98) */
EX_F15           = &H66             '                    (NEC PC98) */

EX_KANA            = &H70           ' (Japanese keyboard)           */
EX_CONVERT           = &H79        ' (Japanese keyboard)           */
EX_NOCONVERT          = &H7B      ' (Japanese keyboard)           */
EX_YEN             = &H7D       ' (Japanese keyboard)           */
EX_NUMPADEQUALS       = &H8D    ' = on numeric keypad (NEC PC98) */
EX_CIRCUMFLEX        = &H90      ' (Japanese keyboard)           */
EX_AT            = &H91         '                    (NEC PC98) */
EX_COLON           = &H92      '                    (NEC PC98) */
EX_UNDERLINE        = &H93         '                    (NEC PC98) */
EX_KANJI           = &H94         ' (Japanese keyboard)           */
EX_STOP            = &H95       '                    (NEC PC98) */
EX_AX            = &H96            '                    (Japan AX) */
EX_UNLABELED        = &H97      '                     (J3100) */
EX_NUMPADENTER       = &H9C      ' Enter on numeric keypad */
EX_RCONTROL         = &H9D
EX_NUMPADCOMMA       = &HB3     ' , on numeric keypad (NEC PC98) */
EX_DIVIDE           = &HB5    ' / on numeric keypad */
EX_SYSRQ            = &HB7
EX_RMENU            = &HB8    ' right Alt */
EX_HOME             = &HC7    ' Home on arrow keypad */
EX_UP            = &HC8    ' Up Arrow on arrow keypad */
EX_PRIOR            = &HC9    ' PgUp on arrow keypad */
EX_LEFT            = &HCB    ' Left Arrow on arrow keypad */
EX_RIGHT            = &HCD    ' Right Arrow on arrow keypad */
EX_END            = &HCF    ' End on arrow keypad */
EX_DOWN             = &HD0    ' Down Arrow on arrow keypad */
EX_NEXT            = &HD1    ' PgDn on arrow keypad */
EX_INSERT           = &HD2    ' Insert on arrow keypad */
EX_DELETE           = &HD3    ' Delete on arrow keypad */
EX_LWIN            = &HDB    ' Left Windows key */
EX_RWIN             = &HDC    ' Right Windows key */
```

```
 EX_APPS           = &HDD    ' AppMenu key */
```

Remarks

Use this function to test the state of a given key. The state can be either EX_KEYDOWN or EX_KEYNOTDOWN, which refers the key being presses or not pressed. This function makes it possible to check for whether multiple keys are pressed or not.

Return values
EX_KEYNOTDOWN: The specified key is not pressed
EX_KEYDOWN: The specified key is pressed
EX_KEYBOARDLOST: The keyboard was lost or not acquired.
EX_UNDEFINED: An unknown error occurred.

CreateMouse() As Long

Description

This function creates the mouse for input.
Parameters

None…

Remarks

This function creates the mouse for use with the mouse functions. This function must be called after the *InitDirectInput()* and before other mouse functions are called.

Return values
EX_OK: The mouse was successfully created.
EX_DIRECTINPUTDEVICEFAILED: the creation failed

SetMouseEvents(StartEvents As Long,) As Long

Description

Sets the mouse up for event notification.

Parameters

StartEvents: This parameter determines whether the event notification should be started or stopped. A value of 1 will start the event notification and a value of 0 will stop the event notification.

Remarks

This function will set the control up for event notification when a state of the mouse changes. The event which will be notified is the *EasyX_MouseEvent(X As Long, Y As Long, Button As Long)*
Use this function instead of the polling function *GetMouseState()*.

Return values

EX_OK: The specified event operation succeeded.
EX_EVENTALLREADYSTARTED: An attempt to start the events was made, but the event was already started.

EX_OUTOFMEMORY: there is not enough memory to complete this operation.
EX_EVENTSNOTSTARTED: An attempt to stop the event notification was made, but the event was not started.

GetMouseState(X As Long, Y As Long, ButtonState As Long) As Long

Description

This function returns the mouse's state into the specified parameters.

Parameters

X: The movement of the X coordinate since the last poll.

Y: The movement of the Y coordinate since the last poll.

ButtonState: The state of the mouse buttons.

Remarks

This function polls the mouse for its state. The values are returned in the parameters passed to the function. The coordinate (X, Y) values are relative values, meaning that they represent the movement since the last call to this function, and not the (X, Y) screen coordinate.

Return values
EX_OK: The values are returned into the passed parameters
EX_DEVICENOTACQUIRED: The device is not acquired.
EX_DEVICENOTCREATED: The device is not created.

AcquireMouse() As Long

Description

This function acquires the mouse.

Parameters

None…

Remarks

This function acquires the mouse after creating or loosing it. This function must be called after *InitDirectInput()* and *CreateMouse()* respectively, and before any other mouse functions are called.

Return values
EX_OK: The device was acquired.
EX_DEVICENOTCREATED: The device is not created
EX_UNDEFINED: An unknown error occurred

AcquireKeyboard() As Long

Description

This function acquires the keyboard

Parameters

None…

Remarks

This function acquires the keyboard after creating or loosing it. This function must be called after *InitDirectInput()* and *CreateKeyboard()* respectively, and before any other keyboard functions are called.

Return values
EX_OK: The device was acquired.
EX_DEVICENOTCREATED: The device is not created
EX_UNDEFINED: An unknown error occurred

EndDirectX() As Long

Description

This function destroys all the created DirectX® objects created and returns the screen back in normal mode.

Parameters

None…

Remarks

Use this function to end a DirectX® session. After calling this function all objects (sprites, surfaces etc.) created are destroyed.

Return values
EX_OK: No errors the objects were destroyed.

SetPalette(BitmapFile As String, SurfaceIndex As Long) As Long

Description

This function sets a palette to a surface.

Parameters

BitmapFile: The bitmap from which the palette will be created.

SurfaceIndex: The surface to attach the palette to.
Remarks

This function will set a palette to a surface. The palette must be from a bitmap.

<u>Return values</u>
EX_OK: The palette was set
EX_UNDEFINED: An unknown error occurred.

IsPaletteMode() As Long

<u>Description</u>

Tests whether the screen mode is palettized.

<u>Parameters</u>

None…

<u>Remarks</u>

Use this function to test if the screen mode is a palettized mode. This function especially applies to Windowed DirectDraw®.

<u>Return values</u>
EX_PALETTEMODE: The mode is palettized
EX_NOPALETTEMODE: The mode is not palettized

IWCreateFakeBackBuffer(
Width As Integer,
Height As Integer,
ColorKey As Long
) As long

<u>Description</u>

Creates a fake back buffer in a Windowed DirectDraw® mode.

<u>Parameters</u>

Width: The width of the fake back buffer

Height: The height of the fake back buffer

ColorKey: The color key of the fake back buffer

<u>Remarks</u>

Use this function to create a fake back buffer. This special function is designed to work as a fake back buffer for the primary buffer. This is necessary in Windowed DirectDraw®, since there is no flipping buffer to work with. Because of this missing flipping (back buffer) buffer, tearing will be apparent if blitting is done directly to the primary surface, which is allowed in Windowed DirectDraw® (not in full screen mode where blitting is done to the back buffer, which is then flipped to the primary surface). By creating this fake back buffer, an empty surface is created, with the specified height and width. This surface can then be blitted to, with the *IWDrawToBackBuffer*, and when all blittings are done, the *IWDrawFakeToPrimary* function can be used to blit the entire fake back buffer onto the primary surface.

Return values

>= 0: The index of the back buffer
EX_SURFACEWRONGSIZE: The specified *Width* and *Height* values were not valid
EX_COLORSETFAILED: The Color key could not be set.
EX_UNDEFINED: An unknown error occurred.

```
IWDrawSprite(UpperX As Integer,
             UpperY As Integer,
             Width As Integer,
             Height As Integer,
             SpriteNumber As Integer
            ) As Long
```

Description

Draws a sprite onto the primary surface.

Parameters

UpperX: The upper x coordinate of the sprite on the primary surface

UpperY: The upper y coordinate of the sprite on the primary surface

Width: The width of the sprite

Height: The height of the sprite

SpriteNumber: The sprite index of the sprite to draw.

Remarks

This function draws a sprite to the primary surface, at the specified location. The parts of the sprite that are not located on the window will be clipped away. Drawing with this function might result in tearing on the sprites.

Return values
EX_OK: The sprite was drawn
EX_UNDEFINED: An unknown error occurred

```
IWDrawToBackBuffer(UpperX As Integer,
                   UpperY As Integer,
                   Width As Integer,
                   Height As Integer,
                   SpriteNumber As Integer,
                    BackBuffer As Integer) As Long
```

Description

This function draws a sprite to a fake created back buffer.

Parameters

UpperX: The upper x coordinate of the sprite.

UpperY: The upper y coordinate of the sprite

Width: The width of the sprite

Height: The height of the sprite

SpriteNumber: The sprite index of the sprite to draw

BackBuffer: The back buffer index

Remarks

Use this function to draw sprites to the fake back buffer. All parts of a sprite that is not within the borders of the fake back buffer will be clipped away. Use this function and the *IWCreateFakeBackBuffer* and *IWDrawFakeToPrimary* to implement a tear free environment.

Return values
EX_OK: The sprite was onto the fake back buffer
EX_UNDEFINED: An unknown error occurred

IWFillSurface(Color As Long, Surface As Long) As Long

Description

Fills a surface with a specified color.

Parameters

Color: The color value, either in BGR for non-palettized mode or and index for palettized mode.

Surface: The index of the surface to be filled

Remarks

This function is similar to the 'normal' *FillSurface* function, except that this one fills a surface in a windowed DirectDraw® session.

Return values
EX_OK: The surface was filled with the specified color.
EX_NOSURFACE: The surface specified was invalid.
EX_UNDEFINED: An unknown error occurred.

IWInitDirectDraw(PaletteFile As Long) As Long

Description

This function initializes DirectDraw® in a windowed session.

Parameters

PaletteFile: The bitmap file of a palette.
Remarks

This function initializes DirectDraw® in a windowed mode. This function should be called instead of the InitDirectDraw, when the application will be working like a normal window. The *PaletteFile* parameter should be used if a specific palette is to be associated with the primary surface. The parameter is of no use in non-palettized modes.

Return values
EX_OK: The initialization succeeded.
EX_CLIPPERFAILED: The clipper creation failed
EX_PRIMARYCREATIONFAILED: The primary surface could not be created
EX_UNDEFINED: An unknown error occurred

IWResetSurface() As Long

Description

Resets and reloads all surfaces and palettes.

Parameters

None…

Remarks

This function resets and reloads all surfaces, and the palette attached to the primary surface. This function is especially useful in Windowed DirectDraw®, where an application does not have exclusive rights to the primary palette (which is actually the system palette). Because of this the palette might change when an other application gets the focus, this will immediately be reflected in any drawings done in the DirectDraw® application, which may still be visible. Nothing can be done about this, and nothing should. But when the DirectDraw® application receives the focus again, the palette is still the same one that was used by the other application. This is where this function comes in handy, since a call to it, will reload the palette and make the appropriate color conversions on all surfaces, so everything again looks like it should. This is of course only relevant in palettized modes.

Return values
EX_OK: The resetting and reloading was successful
EX_NOPALETTE: The screen mode is not palettized
EX_UNDEFINED: An unknown error occurred

IWDrawFakeToPrimary(Backbuffer As Long) As Long

Description:

This functions draws everything in the back buffer onto the primary surface

Parameters:

Backbuffer: The index of the backbuffer

Remarks:

Use this function to draw everything from the back buffer to the primary surface. This function acts like a fake flipping function, and should be worked into a double buffering scheme, with the *IWCreateFakeBackBuffer* and the *IWDrawToBackBuffer* functions.

Return values:
EX_OK: The back buffer was drawn successfully
EX_UNDEFINED: An unknown error occurred

```
DrawCircle( CenterX As Long,
            CenterY As Long,
            Radius As Long,
            Color As Long,
           SurfaceIndex As Long
           ) As Long;
```

Description

This function draws a circle on the specified surface, with the specified dimensions. The circle will always be 1 pixel wide.

Parameters

CenterX: The X-coordinate of the center of the circle

CenterY: The Y-coordinate of the center of the circle

Radius: The Radius of the circle in pixels

Color: The color of the circle

SurfaceIndex: The surface to draw the circle on

Remarks

NOTE: This function will only work in 8, 16 and 32 bit color mode, and not in 24 bit color mode

This function draws a circle on the specified surface. The circle will automatically be clipped if it is out of bounds (to wide or too high compared to the screen size). The circle is not drawn with the Win32 GDI functions, but is instead drawn directly onto the surface memory. This makes the function very fast and efficient.

Return values:

EX_OK: The circle was drawn.
EX_NOSURFACE: The surface specified was invalid
EX_UNDEFINED: The circle could not be drawn

DrawLine(X1 As Long,
Y1 As Long,
X2 As Long,
Y2 As Long,
Color As Long,
SurfaceIndex As Long
) As Long

Description

This function draws a line, with the specified dimensions and color.

Parameters

X1: The X-coordinate of the first point defining the line

Y1: The Y-coordinate of the first point defining the line

X2: The X-coordinate of the second point defining the line

Y2: The Y-coordinate of the second point defining the line

Color: The color of the line

SurfaceIndex: The surface to draw the line on.

Remarks

NOTE: This function will only work in 8, 16 and 32 bit color mode, and not in 24 bit color mode

This function draws a line on the specified surface. The line is dimensioned by two points (X1, Y1) and (X2, Y2). The line will automatically be clipped if it goes out of bounds. The width is always 1 pixel. As with the *DrawCircle* function, then this function is a Win32 GDI call, but instead draws the line directly onto the surface, thus making it very fast and efficient.

Return values:
EX_OK: The line was drawn correctly
EX_NOSURFACE: The surface specified was invalid
EX_UNDEFINED: The line could not be drawn

Plot(X As Long,
Y As Long,
ColorKey As Long,
SurfaceIndex As Long,
) As Long;

Description

This function plots a pixel on the specified surface.

Parameters

X: The X-coordinate of the pixel to be drawn

Y: The Y-coordinate of the pixel to be drawn

ColorKey: The color of the pixel

SurfaceIndex: The surface to draw the pixel on

Remarks

NOTE: This function will only work in 8, 16 and 32 bit color mode, and not in 24 bit color mode

This function plots a pixel on the specified surface. The pixel will be drawn with specified color, at the specified coordinate. As with the other drawing functions, then this function draws directly onto the surface memory, without going through the Win32 GDI system.

Return values:
EX_OK: The Pixel was plotted onto the surface
EX_NOSURFACE: The specified surface was invalid
EX_INVALID: The pixel could not be plotted.

GetStaticByteLength(SoundIndex As Long) As Long

Description

This function returns the length of the given sound buffer in bytes.

Parameters

SoundIndex: The buffer to return the length of.

Remarks

Use this function to determine the length of a given sound buffer. It is especially useful in conjunction with the *GetCurrentStaticPosition* and the *SetCurrentStaticPosition*.

Return values:
Return > 1: The length of the sound buffer in bytes
EX_BADSOUNDFILE: The sound index was invalid

SetCurrentStaticPosition(SoundIndex As Long, NewPosition As Long) As Long

Description

This function sets the current play position in the specified position.

Parameters

SoundIndex: The sound buffer to edit

NewPosition: The new play position in the buffer.

Remarks

Use this function to set the play position of the specified sound buffer. This will make the sound buffer start playing from this position instead of position 0. The *NewPosition* value must not be greater or smaller than the length of the sound buffer. Use the GetStaticByteLength to obtain the length of the sound buffer.

Return values:
EX_OK: The new position was set
EX_BADSOUNDFILE: The sound file specified was invalid.
EX_INVALID: The value specified in the *NewPosition* variable was invalid

GetCurrentStaticPosition(SoundIndex As Long) As Long

Description

This function returns the current play position of the sound buffer

Parameters

SoundIndex: The sound index of the buffer to return the position of.

Remarks

Use this function to get the play position of the sound buffer in bytes. This function is useful in conjunction with the *SetCurrentStaticPosition* and *GetStaticByteLength*.

Return values:

Return >= 0: The current play position of the sound buffer.
EX_BADSOUNDFILE: The sound index was invalid
EX_INVALID: An unknown error occurred.

GetStreamingFrequency(SoundIndex As Long) As Long

Description

This function returns frequency of a streaming sound.

Parameters

SoundIndex: The index of the sound buffer to return the frequency from

Remarks

Use this function to obtain the frequency of the specified sound buffer

Return values:

Return > 0: The frequency of the sound buffer
EX_BADSOUNDFILE: The sound index specified was invalid

SetStaticFrequency(SoundIndex As Long, NewFrequency As Long) As Long

Description

This function sets the frequency of the specified sound buffer.

Parameters

SoundIndex: The sound buffer to set the frequency on

NewFrequency: The new frequency of the sound buffer.

Remarks

Use this function to set the frequency of a static sound buffer. The minimum value is 100 and the maximum value is 100.000. Remember that changing the frequency also changes the time it takes to play the sound, i.e. if the frequency is made greater than the original, the sound buffer will play faster and vice versa.

Return values:
EX_OK: The frequency was set successfully.
EX_BADFREQUENCY: The value of the *NewFrequency* variable was invalid
EX_BADSOUNDFILE: The index to the sound buffer was invalid

GetStaticFrequency(SoundIndex As Long) As Long

Description

This function returns the frequency of the specified sound buffer.

Parameters

SoundIndex: The index of the sound buffer to return information from

Remarks

Use this function to get the frequency of the specified sound buffer. This function is especially useful with the *SetStaticFrequency* function

Return values:

Return >1: The frequency of the sound buffer.
EX_BADSOUNDFILE: The sound index was invalid

GetDisplayModes() As Long

Description

This function triggers the DisplayModes event.

Parameters

None…

Remarks

Use this function to get all the possible display modes on the operating computer. Calling this function will trick the DisplyModes event to fire as many times as there are different display settings.
NOTE: This function must be called before any initialization is done, i.e. before any display mode is created.
See the *DisplayModes* event for more information

Return values:
EX_OK: The function fired the event properly
EX_UNDEFINED: An unknown error occurred
EX_WRONGDIRECTXVERSION: The version of DirectX installed on the computer is wrong

Properties:

*<Control>.***Window**

This property sets the hWnd or window of the DirectX® session. This property *must be set before* any functions are called.

To set this property simply use a forms hWnd property:

```
EasyX.Window = Me.hWnd
```

Events:

EasyX_MouseEvent(X As Long, Y As Long, Button As Long)

This event is fired on mouse movements or events. To use this event call the *SetMouseEvents()* function.

The coordinate values (X, Y) passed to this function are relative values, meaning that they represent the movement since the last time the event was fired.

DisplayModes(Width As Long, Height As Long, PixelFormat As Long)

This event is fired when the *GetDisplayModes* function is called. The event will usually be fired several times as it goes through the different settings available.

Width: This value contains the Width of the screen in a given mode

Height: This variable contains the height of the screen in a given mode

PixelFormat: This varaible holds the pixel format of the given mode.