DFL Software

Light Lib Images 1.0

DLL Functions

We strongly suggest that you use the extensive support classes and functions provided for the individual languages instead of calling the DLLs directly. The following DLL functions are provided for reference and should not be called directly unless you have a thorough understanding of how to use them.

Constants

Device Constants

LLI_DISK
LLI_MEMORY
LLI_SCANNER
LLI_SCREEN

System Constants

LLI_CLASS_APPLICATION
LLI_CLASS_IMAGE

LLI_APPLICATION_VERSION	`Get Light Lib Images version`
LLI_APPLICATION_IDLE	`Get/Set Udf idle function`
LLI_APPLICATION_IDLE_REPEAT	`Get repeat count of Udf`

General Constants

LLI_FULL_SIZE	Maximun size for any image
LLI_IMG_IS_IMAGE	Image signature
LLI_PALETTE_SHARED	Use a shared palette
LLI_PALETTE_EXCLUSIVE	Use an exclusive color palette
LLI_VOID_PARAM	Void parameter

Disk Constants

Device

LLI_DISK

File Types

LLI_DISK_AUTO	File Type determined from extension
LLI_DISK_BMP	BMP file format
LLI_DISK_GIF	GIF file format
LLI_DISK_JPG	JPG file format
LLI_DISK_PCX	PCX file format
LLI_DISK_TIF	TIF file format

Disk Compression Types

LLI_DISK_COMPRESS_AUTO	Use the best compression automatically
LLI_DISK_COMPRESS_CCITT1D	Modified Huffman for TIF
LLI_DISK_COMPRESS_CCITTG3	CCITT Group 3 for TIF
LLI_DISK_COMPRESS_CCITTG4	CCITT Group 4 for TIF
LLI_DISK_COMPRESS_LZW	Lempel for TIF and GIF
LLI_DISK_COMPRESS_RLE	Run Length Encoding TIF/PCX

Memory Constants

Device

LLI_MEMORY

Colors

LLI_MEMORY_16	`4 bits / pixel`
LLI_MEMORY_256	`8 bits / pixel`
LLI_MEMORY_16M	`24 bits / pixel`
LLI_MEMORY_BW	`1 bit / pixel`

Scanner Constants

Device
LLI_SCANNER

LLI_SCANNER_TWAIN	`TWAIN`
LLI_SCANNER_TWAIN_DIALOG	`TWAIN Dialog`

Screen Constants

Device

LLI_SCREEN

Type

LLI_SCREEN_DEVICE_CONTEXT	Screen using a device context
LLI_SCREEN_WINDOW_HANDLE	Screen using aWindow handle

Transforming

LLI_COPY_CLONE	iCopy() Clone message
LLI_COPY_TURN	iCopy() Turn message
LLI_COPY_ZOOM	iCopy() Zoom message

Turn formats

LLI_TURN_180	iCopy(LLI_COPY_TURN) 180°
LLI_TURN_270	iCopy(LLI_COPY_TURN) 270°
LLI_TURN_90	iCopy(LLI_COPY_TURN) 90°

Zoom Constants

LLI_ZOOM_FIT_HEIGHT	All the image height must fit in the window
LLI_ZOOM_FIT_NONE	The zoom factor is driven buy the scale factors
LLI_ZOOM_FIT_REFRESH	Refresh the fit request on a window resize
LLI_ZOOM_FIT_WIDTH	All the image width must fit in the window

Fit Constants (CA-Visual Objects)

LLI_FIT_HEIGHT	Fit the image height in the window
LLI_FIT_NONE	Zoom factor is determined by the scale factors
LLI_FIT_REFRESH	Refresh the requested Fit mode on a window resize
LLI_FIT_WIDTH	Fit theimage width in the window
LLI_FIT_WINDOW	Fit the entire image in the window

User Defined Constants

LLI_UDF_ABORT	Udf Abort return value
LLI_UDF_CONT	Udf Continue return value
LLI_UDF_ERROR	Error append during a LLI function execution
LLI_UDF_EXIT	Exit phase for a LLI function execution
LLI_UDF_IDLE	Idle phase for a LLI function execution
LLI_UDF_INIT	Init phase for a LLI function execution

Callback Constants

```
LLI function names passed to the callback function
```

LLI_CALLER_COPY
LLI_CALLER_GET
LLI_CALLER_PUT

Image Information Constants

LLI_IMAGE_BITS	`Read Only`
LLI_IMAGE_CARGO	`Assignable`
LLI_IMAGE_COLORS	`Read Only`
LLI_IMAGE_DPI	`Read Only`
LLI_IMAGE_HEIGHT	`Read Only`
LLI_IMAGE_WIDTH	`Read Only`

Introduction

Welcome to Light Lib Images !

Light Lib Images was designed to be the easiest image management library available for Windows application developers. Light Lib Images enables you to easily implement document and image support for existing or new applications. Light Lib Images is highly optimized for very fast image processing. The product provides full support for .BMP, .PCX, .TIF, .GIF, .JPG file formats and uses state-of-the-art compression RLE, HUFFMAN, LZW(ZIP), CCITT Group 3 & 4 (fax) and JPEG.

Light Lib Images' amazing speed is based on excellent internal image processing. All images are divided into smaller images or "strips" before beingprocessed. This gives Light Lib Images the ability to display very large images efficiently. Other imaging libraries do not do this. They rely on Windows to provide the necessary memory management in the hope that an image will fit neatly into availablememory.

Light Lib Images was developed with the following goals in mind:

Ease of Use	It is very easy to integrate Light Lib Images into existing applications. There exists lessthan a dozen core functions which provide the power needed to manage images.
Execution speed	Execution speed is excellent. Our internal processing algorithms have been optimized to let you efficiently manage very large images.
Language Support	Providing DLLs is not enogh. All Light Lib products for Windows come with extensive language support which make it effortless to introduce Light Lib libraries using established and familiar syntax. Light Lib products are the best solution if you need to develop applications with various Windows languages. There is no need to learn different libraries because Light Lib products provide support for the following WIndows application development languages, Borland C++ Microsoft C/C++ Microsoft Visual Basic CA-Visual Objects

Light Lib Images is comprised of a small set of core functions which provide all of the necessary services Above the core is the language support layer which provides simple interface to the core functions in the desired language. Language support is provided MS-Visual Basc, CA_Viaual Objects, and C/C++ Windows Development Systems..

All trademarks are the property of their registered owners.

Available devices are Screen, Printer, Scanner, Disk and Memory.

No help available for this section.

Using Light Lib Images

The easiest way to learn any new concept is by example. Each installed language has the source code to its own set of demonstration programs. Please reference them to gain a good understanding of how to use Light Lib Images.

If you are familiar with object-oriented programming, you will find it useful to implement OOP concepts.

How to use this Help

This help system was designed to provide quick access to information. Help is provided for the extensive language support and for the supplied Light Lib Images and Light Lib Objects DLLs.

We strongly suggest that you use the individual language support with your applications!

When a language is selected, you will be prompted with an overview of all support classes and/or functions. There is also a "How Do I?" section which provides step-by-step instructions on various common tasks.

A secondary window will open containing details and descriptions when any of these items are selected. This window is set to always stay on top. That way once a help topic is selected, you can continue working without losing focus on this window. To close it, simply select the window's system menu and select Close.

Quick Start

See the sample application provided for each supported language. You should execute thesample application and experiment with the image features in order to gain a good understanding of how Light Lib Images works. Once you understand how to load and save an image, you will be able to easily modify the samples to fit your needs.

See also How Do I?

Overview

Light Lib Images is a powerful yet easy to implement image management library. It is comprised of a few core DLL functions. Each of these functions take several parameters which provide important programming flexibility. For example, iPut() is used to dothree of the following different things depending on the device parameter passed.

Action	*Device Constant*
Display an image on Screen	LLI_SCREEN
Send an image to a Printer	LLI_PRINTER
Save an image on Disk	LLI_TWAIN

Light Lib uses a set of pre-defined device constant which allow the use of the same Light Lib functions for multiple target devices. For example, changing the value of the iPut() device constant parameter allows the image to be sent to the screen, sent to a printer, or saved to disk. Other device constants are used to specify image and file formats, compression techniqes, processing techniqes, scaling and so on. For example, when saving an image to disk, you will need to provide the file name and file type in addition to the disk device constant.

If your operation does not require the use of a particular parameter, simply substitute the unused parameter with the predefined value LLI_VOID_PARAM.

Readme

Compatibility

Windows Screen Drivers

Light Lib Images is compatible with all installed Windows screen drivers.

Windows Printer Drivers

Light Lib Images is compatible with all installed Windows printer drivers.

TWAIN Scanner

Light Lib Images is fully TWAIN compliant

Callback Functions

Light Lib images uses "Callback" functions toprovide your applications with the ability to do something while an image is being prcocessed. For example, displaying a gauge while an image is being scanned. This allows you to keep users updated during various image manipulation. See also Callback Constants

Referencing an object from a Callback Function

In general, images are attached to objects (for example a window), but when Light Lib Images executes a callback function which is not a method, the reference to "self" is lost. In this case, we suggest that "self" be passed to one of the user defined parameters. This gives your callback function a reference to "self" which provides access to all themethods and instance variables. This is very useful object oriented programming. So, if your callback function needs to display a message inside the window which owns the image, you can !

BLOBs

Light Lib Images supports BLOB (Binary Large OBjects) data formats in that you are able to convertan image into a BLOB and vice versa. This allows you to store images to files which support BLOBs.

BLOB's are supported internally at this time. In the near future, the API will be published.

You can convert image data to a BLOB using iImg2Blob(*ptrImg*) and store thisBLOB in a database's BLOB field. Here is a simple example:

```
Function BlobSample( cFile )

Local ptrImg  // Pointer to an image structure containing the image

// Load .TIF file image from disk
ptrImg = iGet(LLI_DISK,
            LLI_DISK_TIF,
            0,
            0,
            LLI_FULL_SIZE ,
            LLI_FULL_SIZE ,
            "MyImage.TIF",            // Image File Name
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM )

// Open a DBF file containing a BLOB type field
use IMAGE.DBF

// Add a new record to the DBFfile
append blank

// Convert the pointer the structure containing the
// image, into a BLOB and save the converted data
// to field called BLOBDATA
IMAGE->BLOBDATA =iImg2Blob( ptrImg )


// Load the image from a record in a database and display it.
ptrImg = iBlob2Img( IMAGE->BLOBDATA )

iPut(   ptrImg,
        0,
        0,
        LLI_FULL_SIZE,
        LLI_FULL_SIZE,
        LLI_SCREEN,
        LLI_WINDOW,
        0,
        0,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM )

// Erase the image from memory
prtImg = iErase( ptrImg )

return( NIL )
```

What's New...

Appendices

CA-Visual Objects Functions and Classes

The CA-Visual Objects support AEF supplied with Light Lib Images should not be modified directly since this support layer calls the Light Lib Objects DLL directly. Functions which should never be modified are explicitly labeled in the AEF itself.

How Do I?

Classes

ImageInWindow
ImageWindow
ImageWindowControl

Functions

dwLightLibApp()
dwLightLibAppRegister()
dwLightLibAppUnRegister()
ImageIdle()
lImageOperationComplex()

Samples

Simple Image In Window

How Do I? CA-Visual Objects

Display an image in an MDI

1. Create a ShellWindow
2. Register the application with the DLL
3. Show the ShellWindow
4. Create an ImageWindow object
5. Show the window
6. Add the oImageWindow object to the ShellWindow's array of ChildWindows
7. At the end of execution, unregister the application from the DLL

```
METHOD Start() CLASS App

    LOCAL oWindow AS StandardShellWindow
    LOCAL oImageWindow AS ImageWindow
    LOCAL sFile := "YOUR.PCX"
    LOCAL lScanImage := FALSE

    // Create a ShellWindow
    oWindow := StandardShellWindow{ self }

    // Register this application with the DLL
    dwLightLibAppRegister( self, oWindow )

    // Show the ShellWindow
    oWindow:Show()

    // Create the ImageWindow object
    oImageWindow := ImageWindow{ oWindow, sFile, lScanImage }

    // Show the image window
    oImageWindow:Show()

    // Add the oImageWindow object to the MDI's array of ChildWindows
    aadd( oWindow:aChildWindows, oImageWindow )

    // At the end of execution, unregister the application from the DLL
    dwLightLibAppUnRegister()

RETURN NIL
```

Scan a document

1. Create a ShellWindow
2. Register the application with the DLL
3. Show the ShellWindow
4. Create an ImageWindow object
5. Show the window
6. Add the oImageWindow object to the ShellWindow's array of ChildWindows
7. At the end of execution, unregister the application from the DLL

```
METHOD Start() CLASS App

    LOCAL oWindow AS StandardShellWindow
    LOCAL oImageWindow AS ImageWindow
    LOCAL sFile := "YOUR.PCX"
    LOCAL lScanImage := FALSE

    // Create a ShellWindow
    oWindow := StandardShellWindow{ self }

    // Register this application with the DLL
    dwLightLibAppRegister( self, oWindow )

    // Show the ShellWindow
    oWindow:Show()

    // Create the ImageWindow object
    oImageWindow := ImageWindow{ oWindow, sFile, lScanImage }

    // Show the image window
    oImageWindow:Show()

    // Add the oImageWindow object to the MDI's array of ChildWindows
    aadd( oWindow:aChildWindows, oImageWindow )

    // At the end of execution, unregister the application from the DLL
    dwLightLibAppUnRegister()

RETURN NIL
```

How Do I? CA-Visual Objects

Add an image to a data window

Please note that for this explanation, you will need to have a DBServer created.

1. Optionally, create a new module or edit an existing one
2. Select the Window Editor to create a new window
3. Select **LightLibDataWindow** as the the window type
4. Provide a name for this new window (eg. Test)
5. Select **AutoLayout** from the Menu
6. Select a DBServer(See note above)
7. Select **LightLibImage** Control from the Toolbar and drag the control onto the data window *or* from the Menu, select **Edit**, then **Palette** and then **ImageWindowControl.**
8. Drag this new image control onto the data window

Clicking on this new image control will allow you to edit the properties. Light Lib Images will generate all the needed source code to take full advantage of the image.

Optimize processing speed

Optimize image quality

To obtain the best image quality, use 24 bit image formats. Currently, the only 24 bit file format supported is .JPG

To display multiple 256 images on the screen, (8 bit or 24bit) your hardware will need to support more than 32k colors, (ie 32K , 65K or 16M). Not that the difference between a 65K and 16M color image is almost indistinguishable to the naked eye and that any setting greater than 256 colors is referred to as "true color".

Color palettes are not used when the video setting is 65K colors and multiple images, regardless of their definition, are able to be displayed very clearly. Keep in mind that using 24 bit images require fast processing machines.

ImageInWindow Class

Purpose

Images which will be implemented inside a window.

Properties

GaugeVisible Access/Assign
Bits Access
Colors Access
Density Access
DisHeight Access
DisWidth Access
Dpi Access
OriHeight Access
OriWidth Access
PaletteShared Access/Assign

Methods

Clear()
ColorOperations()
Crop()
Display()
Fit()
FitInWindow()
FitRelease()
FitRescale()
FitToHeight()
FitToWidth()
Grab()
IdleOff()
IdleOn()
Information()
Init()
IsLoaded()
Load()
MemoryImage()
MemoryImage16()
MemoryImage16M()
MemoryImage256()
MemoryImageBW()
Print()
Rotate()
SaveAs()
Scan()
SwapSharedExclusive()
Zoom()

System Properties

These properties are used internally. The are provided as reference only and should ***NEVER*** *be accessed directly in your applications.*
fScaleX Export
fScaleY Export
lPaletteShared Export

oWindowOwner Export

Inherits From

(No ancestors)

Inherited By

(No descendants)

ImageInWindow:fScaleX Export

Description

The X scale factor to use when zooming an image. Do not assign directly.

Type

FLOAT

ImageInWindow:fScaleY Export

Description

The Y scale factor to use when zooming an image. Do not assign directly.

Type

FLOAT

ImageInWindow:lPaletteShared Export

Description

If the color palette is being used in Shared mode. Do not assign directly.Use ImageInWindow:PaletteShared instead.

Type

LOGICAL

ImageInWindow:oWindowOwner Export

Description

The image's owner Window. Do not assign directly.

Type

OBJECT

ImageInWindow:GaugeVisible Access/Assign

Description

If the image gauge is being used.

Type

LOGICAL

ImageInWindow:PaletteShared Access/Assign

Description

If the color palette is being used in Shared mode.

Type

LOGICAL

ImageInWindow:Bits Access/Assign

Description

The number of bits in an image. siBits contains the number of bits per pixel. The following are the possible values:

2	for B&W
4	for 16 colors
8	for 256 colors
24	for 16M colors

Type

SHORTINT

ImageInWindow:Colors Access/Assign

Description

The number of colors in an image.

Type

SHORTINT

ImageInWindow:Density Access/Assign

Description

The density of an image.

Type

SHORTINT

ImageInWindow:DisHeight Access/Assign

Description

The displayable height of an image.

Type

SHORTINT

ImageInWindow:DisWidth Access/Assign

Description

The displayable width of an image.

Type

SHORTINT

ImageInWindow:Dpi Access/Assign

Description

The dots per inch (DPI) of an image. The following are the possible values are 150 and 300.

Type

SHORTINT

ImageInWindow:OriHeight Access/Assign

Description

The original height of an image.

Type

SHORTINT

ImageInWindow:OriWidth Access/Assign

Description

The original width of an image.

Type

SHORTINT

ImageInWindow:Clear() Method

Purpose

Prepare the window area for the image to be painted by clearing all variables and removing images from memory.

Syntax

```
<oImageInWindow>:Clear() ---> NIL
```

Arguments

None

Returns

NIL

ImageInWindow:ColorOperations() Method

Purpose

Provide a dialog which allows for performing various operations on an image (eg. dithering, quantising, grayscaling etc.)

Syntax

```
<oImageInWindow>:ColorOperations() ---> NIL
```

Arguments

None

Returns

NIL

ImageInWindow:Crop() Method

Purpose

Crop an image.

Syntax

<oImageInWindow>:Crop(*<oStart>, <oEnd>*) ---> NIL

Arguments

<oStart>	The start point of the cropping rectangle.
<oEnd>	The end point of the cropping rectangle.

Returns

NIL

Description

Allows cropping to be performed on an image.

ImageInWindow:Display() Method

Purpose

Display an image.

Syntax

<oImageInWindow>:Display(*<hDC>*) ---> NIL

Arguments

<hDC>	Handle to the Device Context

Returns

NIL

ImageInWindow:Fit() Method

Purpose

Adjust the way an image is displayed in a window.

Syntax

<oImageInWindow>:Fit(<*siFitMode*>) ---> NIL

Arguments

<*siFitMode*>	The Fit method to use in adjusting the image. See Screen Constants for the available modes.

Returns

NIL

Description

This will properly fit an image to the coordinates of its bounding window.

ImageInWindow:FitInWindow() Method

Purpose

Fits an image or document completely in a window.

Syntax

<oImageInWindow>:FitInWindow() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:FitRelease() Method

Purpose

Releases an image that was previously fit in a Window using <oImageInWindow>:Fit()

Syntax

<oImageInWindow>:FitRelease() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:FitReScale() Method

Purpose

Re-applies aFit attribute after a window has been resized.

Syntax

<oImageInWindow>:FitRescale() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:FitToHeight() Method

Purpose

Fit the entire height of an image in a window.

Syntax

<oImageInWindow>:FitToHeight() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:FitToWidth() Method

Purpose

Fit the entire width of an image in a window.

Syntax

<oImageInWindow>:FitToWidth() ---> NIL

Arguments

None.

Returns

NIL

ImageInWindow:Grab() Method

Purpose

Captures an image on the screen.

Syntax

<oImageInWindow>:Grab(*<liScreenGrabMode>*) ---> NIL

Arguments

<liScreenGrabMode>	Screen area to capture. Valid values are
	LLI_SCREEN_CLIENT_AREA Everything inside the window not including menus and borders.
	LLI_SCREEN_DESKTOP Entire desktop
	LLI_SCREEN_WINDOW Everything inside the window including menus and borders.

Returns

NIL

Description

This method is useful in capturing various portions of the screen such as dialog windows or forms.

ImageInWindow:IdleOff() Method

Purpose

Disable the Idle callback function

Syntax

<oImageInWindow>:IdleOff() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:IdleOn() Method

Purpose

Enable the Idle callback function

Syntax

<oImageInWindow>:IdleOn(*<iRepeat>*) ---> NIL

Arguments

<iRepeat>	The number of times to call the Idle callback function

Returns

NIL

ImageInWindow:Information() Method

Purpose

Display image information in a window.

Syntax

ImageInWindow:Information() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:Init() Method

Purpose

Creates a new ImageInWindow object.

Syntax

ImageInWindow{ *<oWindow>, <sFileName>*, *<lGetFromScanner>* }   ---> SELF

Arguments

<oWindow>	Window to use when displaying the image.
<sFileName>	Image to open.
<lGetFromScanner>	Logical flag if the image is to be retrieved from the scanner.

Returns

SELF

Description

This will also setup the image to use the proper color palette.

ImageInWindow:IsLoaded() Method

Purpose

Checks if an image is loaded in the ImageInWindow.

Syntax

<oImageInWindow>:IsLoaded()   ---> <*lLoaded*>

Arguments

None

Returns

<*lLoaded*>	If the image is loaded

ImageInWindow:Load() Method

Purpose

Load an image file.

Syntax

<oImageInWindow>:Load(<*sFileName*>, <*liFileFormat*>) ---> NIL

Arguments

<*sFileName*>	Image file name to load.
<*liFileFormat*>	File format. See Editions for supported file formats. If this is not specified, LLI_DISK_AUTO will be used

Returns

NIL

Description

This method calls iGet() with LLI_DISK and stores the image in a protected class instance *dwDisImage*

ImageInWindow:MemoryImage() Method

Purpose

Syntax

<oImageInWindow>:MemoryImage() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:MemoryImage16() Method

Purpose

Syntax

<oImageInWindow>:MemoryImage16() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:MemoryImage16M() Method

Purpose

Syntax

<oImageInWindow>:MemoryImage16M() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:MemoryImage256() Method

Purpose

Syntax

<oImageInWindow>:MemoryImage256() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:MemoryImageBW() Method

Purpose

Syntax

<oImageInWindow>:MemoryImageBW() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:Print() Method

Purpose

Syntax

<oImageInWindow>:Print() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:Original() Method

Purpose

Reset an image to its original state.

Syntax

<oImageInWindow>:Original() ---> NIL

Arguments

None

Returns

NIL

ImageInWindow:Rotate() Method

Purpose

Rotate an image.

Syntax

<oImageInWindow>:Rotate(*<siTurnAngle>*) ---> NIL

Arguments

<siTurnAngle>	Angle to turn the image or document.

Returns

NIL

ImageInWindow:SaveImageAs() Method

Purpose

Display a dialog and save a loaded image to disk using the selected file format.

Syntax

<oImageInWindow>:SaveImageAs() ---> NIL

Arguments

None

Returns

NIL

Description

This will prompt a user with a dialog containing several file formats to select as options. It will then proceed to save the loaded image using the selected file format.

ImageInWindow:Scan() Method

Purpose

Scan an image.

Syntax

<oImageInWindow>:Scan(*<sFileName>*) ---> NIL

Arguments

<sFileName>	File name to save the scanned image to.

Returns

NIL

Description

You can use oImageInWindow>:IsLoaded() to determine if an image was successfully loaded or scanned.

ImageInWindow:SwapSharedExclusive() Method

Purpose

Toggles the mode of a color palette between Shared and Exclusive.

Syntax

ImageInWindow:SwapSharedExclusive() ---> NIL

Arguments

None

Returns

NIL

Description

An Exclusive color palette provides for best results. However, if you are viewing several images simultaneously you may require setting the color palette to Shared. A Shared color palette is comprised of an optimized selection of colors based on available video colors and the colors required by the images themselves. Light Lib Images uses sophisticated calculations when establishing the a Shared palette.

ImageInWindow:Zoom() Method

Purpose

Zoom an image by a passed value.

Syntax

ImageInWindow:Zoom(*<fZoomFactorX>*, *<fZoomFactorY>*) ---> NIL

Arguments

<fZoomFactorX>	Scaling factor for the X axis
<fZoomFactorY>	Scaling factor for the Y axis

Returns

NIL

ImageWindow Class

Purpose

Provide a window capable of displaying an image.

Properties
None

Methods
Close()
ColorOperations()
Crop()
Destroy()
Expose()
FileExit()
FitInWindow()
FitRelease()
FitToHeight()
FitToWidth()
GrabClientArea()
GrabDeskTop()
GrabWindow()
HorizontalScroll()
Information()
Init()
Load()
MouseButtonDown()
MouseButtonUp()
MouseDrag()
Open()
Print()
PrinterSetup()
Resize()
RotateInvert()
RotateLeft()
RotateRight()
SaveAs()
Scan()
ScanAndSave()
SwapSharedExclusive()
VerticalScroll()
Zoom10In()
Zoom10Out()
Zoom25In()
Zoom25Out()

System Properties
*These properties are used internally. The are provided as reference only and should **NEVER** be accessed directly in your applications.*
oImageInWindow Export
oScrollBarHor Export
oScrollBarVer Export

Inherits From

(No ancestors)

Inherited By

(No descendants)

ImageWindow:oImageInWindow Export

Description

Reference to the ImageInWindow object.

Type

OBJECT

ImageWindow:oScrollBarHor Export

Description

Reference to a Horizontal Scroll Bar object.

Type

OBJECT WindowHorizontalScrollBar

ImageWindow:oScrollBarVer Export

Description

Reference to a Horizontal Scroll Bar object.

Type

OBJECT WindowVerticalScrollBar

ImageWindow:Close() Method

Purpose

Close the window containing an image.

Syntax

<oImageWindow>:Close() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:ColorOperations() Method

Purpose

Provide a dialog which allows for performing various operations on an image (eg. dithering, quantising, grayscaling etc.)

Syntax

<oImageWindow>:ColorOperations() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:Crop() Method

Purpose

Crop a selected part of an image.

Syntax

<oImageWindow>:Crop() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:Destroy() Method

Purpose

Destroy the ImageWindow object.

Syntax

<oImageWindow>:Destroy() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:Expose() Method

Purpose

Display an image in a window.

Syntax

<oImageWindow>:Expose(*<oEvent>*) ---> NIL

Arguments

<oEvent>

Returns

NIL

ImageWindow:FileExit() Method

Purpose

Post a message to have the window closed.

Syntax

<oImageWindow>:FileExit() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:FitInWindow() Method

Purpose

Fits an image or document completely in a window.

Syntax

<oImageWindow>:FitInWindow() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:FitRelease() Method

Purpose

Releases an image that was previously fit in a Window.

Syntax

<oImageWindow>:FitRelease() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:FitToHeight() Method

Purpose

Fit the entire height of an image in a window.

Syntax

<oImageWindow>:FitToHeight() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:FitToWidth() Method

Purpose

Fit the entire width of an image in a window.

Syntax

<oImageWindow>:FitToWidth() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:GrabClientArea() Method

Purpose

Captures the window contents.

Syntax

<oImageWindow>:GrabClientArea() ---> NIL

Arguments

None

Returns

NIL

Description

This method is useful in capturing various portions of the screen such as dialog windows or forms. This method does not capture the window's menus and borders.

See also: ImageWindow:GrabDeskTop(), ImageWindow:GrabWindow()

ImageWindow:GrabDeskTop() Method

Purpose

Captures the entire desktop.

Syntax

<oImageWindow>:GrabDeskTop() ---> NIL

Arguments

None

Returns

NIL

Description

This method is useful in capturing the desktop.

See also: ImageWindow:GrabClientArea(), ImageWindow:GrabWindow()

ImageWindow:GrabWindow() Method

Purpose

Captures everything in a window.

Syntax

<oImageWindow>:GrabWindow() ---> NIL

Arguments

None

Returns

NIL

Description

This method is useful in capturing various portions of the screen such as dialog windows or forms. This method captures the window and its menus.

See also: ImageWindow:GrabClientArea(), ImageWindow:GrabDeskTop()

ImageWindow:HorizontalScroll() Method

Purpose

Allows the thumb position on the Horizontal Scroll bar to position an image in a window.

Syntax

<oImageWindow>:HorizontalScroll() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:Information() Method

Purpose

Display image information in a window.

Syntax

<oImageWindow>:Information() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:Init() Method

Purpose

Creates a new ImageWindow object.

Syntax

ImageWindow{ <*oParentWindow*>, <*sFileName*>, <*lGetFromScanner*> } ---> SELF

Arguments

<*oParentWindow*>	Window to use when displaying the image
<*sFileName*>	Image to open.
<*lGetFromScanner*>	Logical flag if the image is to be retrieved from the scanner.

Returns

SELF	Reference to the image window

ImageWindow:Load() Method

Purpose

Load an image file

Syntax

<oImageWindow>:Load(*<sFileName>, <liFormat>*) ---> NIL

Arguments

<sFileName>	Image file name to load.
<liFileFormat>	File format. See Editions for supported file formats. If this is not specified, LLI_DISK_AUTO will be used.

Returns

NIL

ImageWindow:MouseButtonDown() Method

Purpose

Syntax

<oImageWindow>:MouseButtonDown(<*oMouseEvent*>) ---> NIL

Arguments

<*oMouseEvent*>	Mouse event.

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindow:MouseButtonUp() Method

Purpose

Syntax

<oImageWindow>:MouseButtonUp(*<oMouseEvent>*) ---> NIL

Arguments

<oMouseEvent>	Mouse event object

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindow:MouseDrag() Method

Purpose

Allow a selected region on an image to be dragged.

Syntax

<oImageWindow>:MouseDrag(<*oMouseEvent*>) ---> NIL

Arguments

<*oMouseEvent*>	Mouse event object

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindow:Open() Method

Purpose

Display an Open File dialog.

Syntax

<oImageWindow>:Open() ---> NIL

Arguments

None

Returns

NIL

Description

Allows the selection of image or document files from the standard windows file open dialog.

ImageWindow:Print() Method

Purpose

Print an image

Syntax

<oImageWindow>:Print() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:PrinterSetup() Method

Purpose

Display the Printer Setup dialog.

Syntax

<oImageWindow>:PrinterSetup() ---> NIL

Arguments

None

Returns

NIL

Description

Calls the standard windows printer setup dialog.

ImageWindow:Resize() Method

Purpose

Resize an ImageWindow object.

Syntax

<oImageWindow>:Resize(*<oResizeEvent>*) ---> NIL

Arguments

<oResizeEvent>	Resizes a window.

Returns

NIL

ImageWindow:RotateInvert() Method

Purpose

Rotate an image 180 degrees.

Syntax

<oImageWindow>:RotateInvert() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:RotateLeft() Method

Purpose

Rotate an image 270 degrees.

Syntax

<oImageWindow>:RotateLeft() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:RotateRight() Method

Purpose

Rotate an image 90 degrees.

Syntax

```
<oImageWindow>:RotateRight() ---> NIL
```

Arguments

None

Returns

NIL

ImageWindow:SaveAs() Method

Purpose

Display a dialog and save a loaded image to disk using the selected file format.

Syntax

<oImageWindow>:SaveAs() ---> NIL

Arguments

None

Returns

NIL

Description

This will prompt a user with a dialog containing several file formats to select as options. It will then proceed to save the loaded image using the selected file format.

ImageWindow:Scan() Method

Purpose

Scan an image.

Syntax

<oImageWindow>:Scan() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:ScanAndSave() Method

Purpose

Scan an image and save it to disk.

Syntax

<oImageWindow>:ScanAndSave() ---> NIL

Arguments

None

Returns

NIL

ImageWindow:SwapSharedExclusive() Method

Purpose

Toggles the mode of a color palette between Shared and Exclusive.

Syntax

<oImageWindow>:SwapShardExclusive() ---> NIL

Arguments

None

Returns

NIL

Description

An Exclusive color palette provides for best results. However, if you are viewing several images simultaneously you may require setting the color palette to Shared. A Shared color palette is comprised of an optimized selection of colors based on available video colors and the colors required by the images themselves. Light Lib Images uses sophisticated calculations when establishing a Shared palette.

ImageWindow:VerticalScroll() Method

Purpose

Syntax

<oImageWindow>:VerticalScroll(*<oScrollEvent>*) ---> NIL

Arguments

<oScrollEvent>	Scroll Event

Returns

NIL

ImageWindow:Zoom10In() Method

Purpose

Enlarge the image by a 10% factor.

Syntax

<oImageWindow>:Zoom10In() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindow:Zoom10Out() Method

Purpose

Decrease the image by a 10% factor.

Syntax

<oImageWindow>:Zoom10Out() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindow:Zoom25In() Method

Purpose

Enlarge the image by a 25% factor.

Syntax

<oImageWindow>:Zoom25In() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindow:Zoom25Out() Method

Purpose

Decrease the image by a 25% factor.

Syntax

<oImageWindow>:Zoom25Out() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindowControl Class

Purpose

Provide a control capable of displaying an image. This class provides the exact same functionality as the ImageWindow Class.

Properties
None

Methods
Close()
ColorOperations()
Crop()
Destroy()
DisplayBorder()
Expose()
FileExit()
FitInWindow()
FitRelease()
FitToHeight()
FitToWidth()
HorizontalScroll()
Information()
Init()
Load()
MouseButtonDown()
MouseButtonUp()
MouseDrag()
Open()
Original()
Print()
PrinterSetup()
RegisterLightLibDataWindowClient()
Resize()
RotateInvert()
RotateLeft()
RotateRight()
SaveAs()
Scan()
ScanAndSave()
SwapSharedExclusive()
VerticalScroll()
Zoom10In()
Zoom10Out()
Zoom25In()
Zoom25Out()

System Properties
These properties are used internally. The are provided as reference only and should ***NEVER*** *be accessed directly in your applications.*
oBottomLeft Export
oImageInWindow Export
oOwner Export

oScrollBarHor Export
oScrollBarVer Export
oTopRight Export

Inherits From

(No ancestors)

Inherited By

(No descendants)

ImageWindowControl:oBottomLeft Export

Description

Reference to the ImageWindowControl object's bottom left Point object.

Type

OBJECT Point

ImageWindowControl:oImageInWindow Export

Description

Reference to the ImageWindowControl object.

Type

OBJECT

ImageWindowControl:oOwner Export

Description

Reference to the ImageWindowControl owner object.

Type

OBJECT

ImageWindowControl:oScrollBarHor Export

Description

Reference to a Horizontal Scroll Bar object.

Type

OBJECT WindowHorizontalScrollBar

ImageWindowControl:oScrollBarVer Export

Description

Reference to a Horizontal Scroll Bar object.

Type

OBJECT WindowVerticalScrollBar

ImageWindowControl:oTopRight Export

Description

Reference to the ImageWindowControl object's top right Point object.

Type

OBJECT Point

ImageWindowControl:Close() Method

Purpose

Close the window containing an image.

Syntax

<oImageWindowControl>:Close() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:ColorOperations() Method

Purpose

Provide a dialog which allows for performing various operations on an image (eg. dithering, quantising, grayscaling etc.)

Syntax

<oImageWindowControl>:ColorOperations() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Crop() Method

Purpose

Crop a selected part of an image.

Syntax

<oImageWindowControl>:Crop() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Destroy() Method

Purpose

Destroy the ImageWindow object.

Syntax

<oImageWindowControl>:Destroy() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:DisplayBorder() Method

Purpose

Display the border of the control

Syntax

<oImageWindowControl>:DisplayBorder() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Expose() Method

Purpose

Display an image in a window.

Syntax

<oImageWindowControl>:Expose(*<oEvent>*) ---> NIL

Arguments

<oEvent>

Returns

NIL

ImageWindowControl:FileExit() Method

Purpose

Post a message to have the window closed.

Syntax

<oImageWindowControl>:FileExit() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:FitInWindow() Method

Purpose

Fits an image or document completely in a window.

Syntax

<oImageWindowControl>:FitInWindow() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:FitRelease() Method

Purpose

Releases an image that was previously fit in a Window.

Syntax

<oImageWindowControl>:FitRelease() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:FitToHeight() Method

Purpose

Fit the entire height of an image in a window.

Syntax

<oImageWindowControl>:FitToHeight() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:FitToWidth() Method

Purpose

Fit the entire width of an image in a window.

Syntax

<oImageWindowControl>:FitToWidth() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:HorizontalScroll() Method

Purpose

Allows the thumb position on the Horizontal Scroll bar to position an image in a window.

Syntax

<oImageWindowControl>:HorizontalScroll() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Information() Method

Purpose

Display image information in a window.

Syntax

<oImageWindowControl>:Information() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Init() Method

Purpose

Creates a new ImageWindowControl object.

Syntax

<oImageWindowControl>:Init(*<oParentWindow>*, *<sFileName>*, *<lGetFromScanner>*) ---> SELF

Arguments

<oParentWindow>	Window to use when displaying the image
<sFileName>	Image to open.
<lGetFromScanner>	Logical flag if the image is to be retrieved from the scanner.

Returns

SELF	Reference to the image window

ImageWindowControl:Load() Method

Purpose

Load an image file

Syntax

<oImageWindowControl>:Load(*<sFileName>, <liFormat>*) ---> NIL

Arguments

<sFileName>	Image file name to load.
<liFileFormat>	File format. See for supported file formats. If this is not specified, LLI_DISK_AUTO will be used.

Returns

NIL

ImageWindowControl:MouseButtonDown() Method

Purpose

Syntax

<oImageWindowControl>:MouseButtonDown(<*oMouseEvent*>) ---> NIL

Arguments

<*oMouseEvent*>Mouse event.

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindowControl:MouseButtonUp() Method

Purpose

Syntax

<oImageWindowControl>:MouseButtonUp(<*oMouseEvent*>) ---> NIL

Arguments

<*oMouseEvent*>	Mouse event object

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindowControl:MouseDrag() Method

Purpose

Allow a selected region on an image to be dragged.

Syntax

<oImageWindowControl>:MouseDrag(<*oMouseEvent*>) ---> NIL

Arguments

<*oMouseEvent*>	Mouse event object

Returns

NIL

Description

This is used in conjunction with ImageWindow:Crop()

ImageWindowControl:Open() Method

Purpose

Display an Open File dialog.

Syntax

<oImageWindowControl>:Open() ---> NIL

Arguments

None

Returns

NIL

Description

Allows the selection of image or document files from the standard windows file open dialog.

ImageWindowControl:Original() Method

Purpose

Restore the image to the original.

Syntax

<oImageWindowControl>:Original() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:Print() Method

Purpose

Print an image

Syntax

<oImageWindowControl>:Print() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:PrinterSetup() Method

Purpose

Display the Printer Setup dialog.

Syntax

<oImageWindowControl>:PrinterSetup() ---> NIL

Arguments

None

Returns

NIL

Description

Calls the standard windows printer setup dialog.

ImageWindowControl:RegisterLightLibDataWindowClient() Method

Purpose

Register the Image control with the LightLibDataWindow.

Syntax

<oImageWindowControl>:RegisterLightLibDataWindowClient(<*cDataFieldName*>) ---> NIL

Arguments

<*cDataFieldName*>	Field name in the data source which contains the image data.

Returns

NIL

ImageWindowControl:Resize() Method

Purpose

Resize an ImageWindow object.

Syntax

<oImageWindowControl>:Resize(*<oResizeEvent>*) ---> NIL

Arguments

<oResizeEvent>	Resizes a window.

Returns

NIL

ImageWindowControl:RotateInvert() Method

Purpose

Rotate an image 180 degrees.

Syntax

<oImageWindowControl>:RotateInvert() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:RotateLeft() Method

Purpose

Rotate an image 270 degrees.

Syntax

<oImageWindowControl>:RotateLeft() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:RotateRight() Method

Purpose

Rotate an image 90 degrees.

Syntax

<oImageWindowControl>:RotateRight() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:SaveAs() Method

Purpose

Display a dialog and save a loaded image to disk using the selected file format.

Syntax

<oImageWindowControl>:SaveAs() ---> NIL

Arguments

None

Returns

NIL

Description

This will prompt a user with a dialog containing several file formats to select as options. It will then proceed to save the loaded image using the selected file format.

ImageWindowControl:Scan() Method

Purpose

Scan an image.

Syntax

<oImageWindowControl>:Scan() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:ScanAndSave() Method

Purpose

Scan an image and save it to disk.

Syntax

<oImageWindowControl>:ScanAndSave() ---> NIL

Arguments

None

Returns

NIL

ImageWindowControl:SwapSharedExclusive() Method

Purpose

Toggles the mode of a color palette between Shared and Exclusive.

Syntax

<oImageWindowControl>:SwapShardExclusive() ---> NIL

Arguments

None

Returns

NIL

Description

An Exclusive color palette provides for best results. However, if you are viewing several images simultaneously you may require setting the color palette to Shared. A Shared color palette is comprised of an optimized selection of colors based on available video colors and the colors required by the images themselves. Light Lib Images uses sophisticated calculations when establishing a Shared palette.

ImageWindowControl:VerticalScroll() Method

Purpose

Syntax

<oImageWindowControl>:VerticalScroll(*<oScrollEvent>*) ---> NIL

Arguments

*<oScrollEvent>*Scroll Event

Returns

NIL

ImageWindowControl:Zoom10In() Method

Purpose

Enlarge the image by a 10% factor.

Syntax

<oImageWindowControl>:Zoom10In() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindowControl:Zoom10Out() Method

Purpose

Decrease the image by a 10% factor.

Syntax

<oImageWindowControl>:Zoom10Out() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindowControl:Zoom25In() Method

Purpose

Enlarge the image by a 25% factor.

Syntax

<oImageWindowControl>:Zoom25In() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageWindowControl:Zoom25Out() Method

Purpose

Decrease the image by a 25% factor.

Syntax

<oImageWindowControl>:Zoom25Out() ---> NIL

Arguments

None

Returns

NIL

Description

CA-Visual Objects does not allow passing parameters to methods when they are accessed via the Menu painter. This Zoom() method is provided to allow quick zooming capabilities in such situations.

ImageIdle() CA-Visual Objects

Purpose

Display a gauge if the operation to be performed requires one.

Syntax

```
ImageIdle(   dwState       AS DWORD,
             liValue       AS LONGINT,
             dwLLImage     AS DWORD,
             liCaller      AS LONGINT,
             dwDevice      AS DWORD,
             dwFormat      AS DWORD,
             dwUserParam   AS DWORD     ) --> liStatusMessage CallBack
```

Arguments

dwState The state of the process. A process goes through 3 states:

LLI_IDLE_INIT	First time this function is called
LLI_IDLE_IDLE	During the operation
LLI_IDLE_EXIT	Last time function is called

liValue Number of times to call this function

dwLLImage Reference to the image

liCaller Type of operation being performed.There are 3 values:

LLI_CALLER_COPY	Copy operation
LLI_CALLER_GET	Load or Get operation
LLI_CALLER_PUT	Display or Put operation

dwDevice Device being used. The following are valid

LLI_DISK
LLI_COPY_CLONE
LLI_COPY_ZOOM
LLI_COPY_TURN
LLI_COPY_QUANTIZE

dwFormat Format of the operation. The following are valid
LLI_DISK_BMP
LLI_DISK_PCX
LLI_DISK_TIF
LLI_DISK_GIF
LLI_DISK_JPG
LLI_TURN_90
LLI_TURN_270

dwUserParam Not used.

Returns

liStatusMessage Current state of the idle operation

Description

This function will display a gauge only if one is need. It uses the IImageOperationComplex() function to determine whether the operation to be performed will take enough time to warrant displaying a gauge.

ImageOperationComplex()   CA-Visual Objects

Purpose

Determine the general complexity of an operation being performed.

Note: This calculation is purely subjective and is provided only as a guideline.

Syntax

```
IImageOperationComplex(    dwLLImage    AS DWORD,
                           liCaller     AS LONGINT,
                           dwDevice     AS DWORD,
                           dwFormat     AS DWORD,
                           oImageWindow AS OBJECT ) --> r4Factor
```

Arguments

dwLLImage	An OBJECT or DWORD pointing to any Light Lib Image Object
liCaller	Type of operation being performed.There are 3 values:

LLI_CALLER_COPY	Copy operation
LLI_CALLER_GET	Load or Get operation
LLI_CALLER_PUT	Display or Put operation

dwDevice Device being used. The following are valid

LLI_DISK
LLI_COPY_CLONE
LLI_COPY_ZOOM
LLI_COPY_TURN
LLI_COPY_QUANTIZE

dwFormat Format of the operation. The following are valid
LLI_DISK_BMP
LLI_DISK_PCX
LLI_DISK_TIF
LLI_DISK_GIF
LLI_DISK_JPG
LLI_TURN_90
LLI_TURN_270

oImageWindow Reference to the ImageWindow object

Returns

r4Factor The factor of complexity for the given operation, type of file format and size of image being used.

Description

This function is used to calculate how complex an image operation is. But in general, the length of time required to perform an operation depends on the parameters described above.

For example, an image that is 1000 pixels by 1000 pixels with 256 colors requires approximately 1Meg of memory will require more time to perform any given operation than an image which requires 200K of memory. In addition, it is known that a LLI_COPY_TURN operation takes longer than a LLI_COPY_ZOOM and that turning an image 90 or 270 degrees takes longer than turning an image 180 degrees. Using all of these factors, we have created a simple guideline for calculating the complexity of any given operation. In turn, this value can be used for determining the approximate time needed to complete the operation.

Sample not available yet.

MS-Visual Basic

Functions & Classes

Introduction

Light Lib Images provides several support files for use with MS-Visual Basic.

Files

MS-Visual Basic Functions & Classes

iVBStruct()
iVBString2Num()
iVBNum2String()

C/C++

Functions & Classes

Introduction

LightLib Images provides several support files for use with C/C++.

Files

C/C++ Functions & Classes

Common Problems and Questions

"Bad" Color Palettes

Light Lib Images relies on the Windows API color palette for its use of colors. When operating Windows in 256 color mode, some limitations may apply in displaying full color images. Particularly when trying to display several images simultaneously. The result is "bad" color palettes for images that are not in focus.

Light Lib Images supports shared color palettes. This means that when an image is displayed, Light Lib Images requests Windows for an optimized color palette for the image. If another image is displayed, a second request for a color palette specific to the new image is made. This allows two images to be properly displayed simultaneously through efficient use of the color palette.

Light Lib Images checks for the number of bits or colors available on the screen device. If the number is 256, Light Lib Images automatically switches to LLI_PALETTE_SHARED. If it is an MDI application and 2 images are loaded in Child windows, both images will have a good representation. You will get a perfect representation of an image by giving focus to one of the images in the Child window.

Displaying multiple 256 color images using a 256 color palette is discouraged. Color palettes do not exist when running Windows in 65K color mode. Applications run in "true colors". Images can access as many colors as needed. Using Windows in 65K colors is highly recommended.

LLI_PALETTE_SHARED
LLI_PALETTE_EXCLUSIVE

The second problem is a Commonview bug. If you look at all the message/method linked to closing a DataWindow (see STD sample for CA-Visual Objects), you will find some direct Windows API call. (WM_CLOSE). This leads us to believe that there is a bug in the DataWindow, which in turn is inherited by Light Lib Images.   We suggest that windows be closed using File/Close or File/Exit until this part of CA-Visual Objects is more stable.

Please note that Light Lib Images' AEFs are compatible with Pre-release version of CA-Visual Objects (Build #388) and these problems will be corrected in the future releases.

Strange Colors

A problem arises when you attempt to display simultaneously a 256-color image with lots of yellows and a 256-color image with lots of greens.

If you use LLI_PALETTE_EXCLUSIVE, Light Lib Graphics will use 256 yellow levels to display the yellow picture. This guarantees a very good display. But if you display the green image with the same LLI_PALETTE_EXCLUSIVE parameters, the palette, which is the same for the entire screen, will change to be optimized for the green image and the yellow image already being displayed will be altered.

The solution is to use LLI_PALETTE_SHARED parameters. In this case, Light Lib Images will use a 256-color linear palette, and all images displayed at the same time on the screen will use the same linear palette. You will not get as good a screen display as if the yellow or green image alone was displayed, but the result will be realistic.

In summary, to display one image at a time and get the best possible results, use LLI_PALETTE_EXCLUSIVE. This is the most common case. To display more than one image at a time, use LLI_PALETTE_SHARED to get a realistic result.

Tips & Techniques

The most important programming technique to implement when manipulating or transforming an image (such as zooming), is to maintain 2 separate pointers to the image. One pointer should be for the original image and the second pointer should should be for the transformed or zoomed image.

In other words, always perform operations such as zooming and copying on the original image but store the resultant image to the second variable or image pointer. This technique is very useful because it allows an image to maintain its sharpness, especially when zooming in and zooming out.

If you don't implement this technique, you may find that an image may lose sharpness after many transformations. This is logical because each transformation distorts the data.

Editions

Light Lib Images comes in two editions. Light Lib Images and Light Lib Images PRO. Essentially the difference is the types of compression and different file formats supported by each. Both products use the same image processing techniques and imaging algorithms to process images.

Light Lib Images

Format	Type	Compression Support
BMP	B&W,Color	Uncompressed
PCX	B&W,Color	RLE (Run Length Encoding)
TIF	B&W,Color	Uncompressed
TIF	B&W,Color	RLE (Run Length Encoding)
TIF	B&W	CCITT 1D Modified Huffman

Light Lib Images PRO

In addition to the standard features, the PRO edition provides support for more advanced and powerful image/document formats and compression algorithms.

Format	Type	Compression Support
TIF	B&W	CCITT G3 Fax Group 3
TIF	B&W	CCITT G4 Fax Group 4
TIF	B&W,Color	LZW (Lempel Ziv Welch)
GIF	Color	LZW (Lempel Ziv Welch)
JPG	Color	JPEG
BLOB	Color	Native Light Lib BLOB support:

iBlob2Img()

Purpose

Convert a BLOB structure pointer to an image structure pointer.

Syntax

iBlob2Img(*Blob*) -> *ptrImg*

Arguments

Blob	Pointer to a BLOB data structure. It must be the result of iImg2Blob()

Returns

ptrImage	Pointer to an image structure containing the image.

Description

Light Lib Images stores images in a special format which has minimal memory requirements and is optimized for speed.

Therefore, if you try to save *ptrImg* directly to a database, such as Oracle or equivalent such system, in a BLOB field, all of the image structure elements will be saved except the element containing the image itself. This is because the image is represented in a format known only to Light Lib Images. The solution is to convert *ptrImg*'s image element to a character string and then process the character string as if it were the image.

Notes

Check the features of the product supporting the BLOBs. Some limitations may apply like 64Kb maximum size. This limitation is often reached when dealing with images. If this is the case, simply store separate image files on disk and store a reference (file name) to the image.

Example

The most common example is the need to store a photo for each person in a file containing people.

A first solution is to create a PCX or TIF file for each person (or record), and to organize the individual PCX or TIF file by storing the file name in a Character field for each person.

To Display an image:

```
// Get the name of the file containing the employee's picture.
// Assume that the image is in TIF format
cPhotoFile = DBF->PHOTOFILE

// Load the TIF file who's file name
// is stored in a character field
ptrImg = iGet(          dwAppLLIHnd,
                        LLI_DISK,
                        LLI_DISK_TIF,
                        siTop,
                        siLeft,
                        siBottom,
                        siRight,
                        cPhotoFile,
                        LLI_VOID_PARAM,
                        LLI_VOID_PARAM,
                        LLI_VOID_PARAM,
                        LLI_VOID_PARAM
                        dwUserParam)

// Display the image to the screen:

iPut(       ptrImg,
            siTop,
            siLeft,
            siBottom,
            siRight,
            LLI_SCREEN,
            LLI_SCREEN_WINDOW_HANDLE,
            0,
            0,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM
            dwUserParam )
```

Another solution is to store each employee's picture directly into the database. The image's data type would be in BLOB format. Therefore, the database engine or file format would need to support BLOB's. Your application would no longer have multiple image files (PCX, TIF etc), but rather the actual image stored in each employee's each record.

To Display a picture using this technique...

```
//Get the actual image or employee picture
// from the database. This image would be
// stored in BLOB format
Blob = DBF->BLOBDATA

// You will need to convert the BLOB to an
// image structure pointer
ptrImg = iBlob2Img( Blob )

// Display the image to the screen
iPut(       ptrImg,
            siTop,
            siLeft,
            siBottom,
            siRight,
            LLI_SCREEN,
LLI_SCREEN_WINDOW_HANDLE,
            0,
            0,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            dwUserParam)
```

Purpose

Copies an image in memory, possibly modifying it.

Syntax

```
iCopy(  dwLLImage        AS DWORD,
        siX1             AS SHORTINT,
        siY1             AS SHORTINT,
        siX2             AS SHORTINT,
        siY2             AS SHORTINT,
        dwTransformMode      AS DWORD,
        dwParam1         AS DWORD,
        dwParam2         AS DWORD,
        dwParam3         AS DWORD,
        dwParam4         AS DWORD,
        dwParam5         AS DWORD,
        dwUserParam      AS DWORD) -> ptrNewImg
```

Arguments

dwLLImage	Source image. This image must be the result of iGet().
siX1, siY1, siX2, siY2	These represent the portion of the ORIGINAL image to copy, not the TARGET image. Specify these coordinates to work on a portion of the original image. Use LLI_FULL_SIZE to process the entire image. The units are in pixels.
dwTransformMode	iCopy() is a versatile function. It can perform several different operations. This specifies the type of operation to perform on an image and the following predefined values are available:

LLI_COPY_CLONE	Returns a second image or image portion identical to the original image.
LLI_COPY_DITHER	Returns a second image which is dithered
LLI_COPY_ZOOM	Returns a second image or image portion transformed by a Zoom effect.
LLI_COPY_TURN	Returns a second image or image portion transformed by a Rotation effect.

dwParam1-dwParam5 These parameters depend on the type of operation you are performing. The following are valid operations

LLI_COPY_CLONE	Requires no extra parameters.
LLI_COPY_ZOOM	*dwParam1* Width of the Target image in pixels *dwParam2* Height of the Target image in pixels
LLI_COPY_TURN	

dwParam1	Direction to turn the image. LLI_TURN_90 LLI_TURN_180 LLI_TURN_270
dwUserParam	This parameter is passed to your Idle user defined callback function. It should be a pointer to any kind of structure. See Referencing an object from a CallBack Function

Returns

ptrNewImg	Pointer to an image structure containing the image.

Description

iCopy() is actually a combination of the functions iGet() and iPut() , but works only on images already in memory.

Notes

To pan across a large image which doesn't fit entirely in a window, simply iPut() portions of the image (use the size of the window to establish the coordinates) with movement coordinates corresponding to the position of the upper left corner of your window. When the user moves across the image, change the image portion displayed to reflect the movements.

If you want the same functionality in a zoom operation, simple image projection is not enough. As you will probably allow users to zoom in and out of the image, you will need to keep 2 pointers to the same image. One pointer to the original image and a second to the zoomed image. When the user changes the zoom factor, use iCopy() to create a full copy of the original image at the requested factor. This is the second image pointer which contains the zoomed image and this is what you will display. Use iCopy(ptrOrigImg, LLI_COPY_CLONE, ...) to obtain a copy or clone of the original image. This is the same as having a projection factor of 1. This 2 pointer technique can be extended to the rotation of an image.Don't forget to destroy the two pointers to the image structure when you no longer need them!

Examples

Consider a black and white image **ptrImg** in A4-B4 format, about 2400 pixels wide (8 inches) by 3300 pixels high (11 inches). This image pointer was created by either reading a document from a scanner using iGet(LLI_SCANNER) , or by reading a file from disk with iGet(LLI_DISK) or was created in memory with iGet(LLI_MEMORY).

To see the image on screen, we use iPut() with the following syntax:

```
iPut(   ptrImg,
        LLI_FULL_SIZE,
        LLI_FULL_SIZE,
        LLI_FULL_SIZE,
        LLI_FULL_SIZE,
        LLI_SCREEN,          // Screen Device
        LLI_SCREEN_WINDOW_HANDLE,
        0,
        0,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM,
        LLI_VOID_PARAM,
LLI_VOID_PARAM,
        LLI_VOID_PARAM,
        dwUserParam)
```

Since not many 2400x3300 pixel screens exist, the LLI_FULL_SIZE calls are overambitious and when using a common 800x600 display, we are unable to see the entire image.By letting the user dynamically change the origin of the image with the use of arrow keys or mouse, we can imagine having a fixed display of 800x600 pixels which gets these pixels at different addresses on the page.

One image pixel still corresponds to one screen pixel. To obtain a full view, we must use iCopy(), since it can deform the image.

```
     ptrZoomImg = iCopy(  ptrImg,
                          LLI_FULL_SIZE,
                          LLI_FULL_SIZE,
                          LLI_FULL_SIZE,
                          LLI_FULL_SIZE,
                          LLI_COPY_ZOOM,
                          800,
                          600,
                          LLI_VOID_PARAM,
                          LLI_VOID_PARAM,
                          LLI_VOID_PARAM,
                          LLI_VOID_PARAM,
          dwUserParam)
```

This projects a 2400x3300 pixel image onto an 800x600 image. Displaying this gives the effect of a full zoom on the image:

```
     iPut(  ptrZoomImg,
            0,
            0,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_SCREEN,
            LLI_SCREEN_WINDOW_HANDLE,
            0,
            0,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            dwUserParam)
```

iErase() DLL Functions

Purpose

Remove an image from memory.

Syntax

iErase(*ptrImg*) -> *ptrImg*

Arguments

ptrImg	The image to be erased or destroyed. *ptrImg* must be the result of iGet() or iCopy()

Returns

ptrImg	Calling iErase(*ptrImg*) is not enough, you **must** use the syntax: *ptrImg* = iErase(*ptrImg*)

Description

Light Lib Images requires that you explicitly erase the pointer to an image structure from memory when you no longer need it. See also oNew() and oDel()

Examples

```
// This example function reads a TIF file "cImgFile" from disk
Function PrintImg( cImgFile )

// Read or get the TIF file
AptrImg = iGet(     LLI_DISK,           // Device
                    LLI_DISK_TIF,       // Format
                    LLI_FULL_SIZE,
                    LLI_FULL_SIZE,
                    LLI_FULL_SIZE,
                    LLI_FULL_SIZE,
                    LLI_VOID_PARAM,
                    LLI_VOID_PARAM,
                    LLI_VOID_PARAM,
                    LLI_VOID_PARAM,
                    LLI_VOID_PARAM,
                    LLI_VOID_PARAM)

// At this point, you can perform operations on the image etc.

// Erase the image from memory
ptrImg = iErase( ptrImg )

return ( NIL )
```

iGet() DLL Functions

Purpose

Retrieve image data from a device.

Syntax

```
iGet(   dwLLOwner     AS DWORD,
        dwDevice      AS DWORD,
        dwFormat      AS DWORD,
        siX1          AS SHORTINT,
        siY1          AS SHORTINT,
        siX2          AS SHORTINT,
        siY2          AS SHORTINT,
        dwParam1      AS DWORD,
        dwParam2      AS DWORD,
        dwParam3      AS DWORD,
        dwParam4      AS DWORD,
        dwParam5      AS DWORD,
        dwUserParam   AS DWORD ) -> ptrImg
```

Arguments

dwLLOwner	A reference to the application handle created with oNew()
dwDevice	Device from which the image data is read. Select one of the above bitmaps for details on specific devices.
dwFormat	Depends on the value of *wDevice*.

The following are the possible predefined combinations of values for these parameters:

wDevice	*wFormat*
LLI_DISK	LLI_DISK_BMP LLI_DISK_GIF LLI_DISK_JPG LLI_DISK_PCX LLI_DISK_TIF
LLI_MEMORY	LLI_MEMORY_BW LLI_MEMORY_16 LLI_MEMORY_256 LLI_MEMORY_16M
LLI_SCANNER	LLI_SCANNER_TWAIN LLI_SCANNER_TWAIN_DIALOG
LLI_SCREEN	LLI_SCREEN_DEVICE_CONTEXT LLI_SCREEN_WINDOW_HANDLE

siX1, siY1, siX2, siY2	Represent the coordinates of the data rectangle retrieved from the device. You can use LLI_FULL_SIZE to specify the entire device.
dwParam1-dwParam5	The extra parameters depend on the device. Select the specific device for further

	info.
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

ptrImg	Pointer to an image structure containing the image. If there is an error the value returned is NIL.

Description

iGet() and iPut() are the two main functions of Light Lib Images.

iGet() reads data from any device (Disk, Screen, Scanner). The data is always read in the form of a rectangle.

iImg2Blob() DLL Functions

Purpose

Convert an image to an array of characters strings.

Syntax

iImg2Blob(*ptrImg*) -> *ptrBlob*

Arguments

ptrImg	A pointer to the image to be converted. *ptrImg* must be the result of iGet() or iCopy()

Returns

ptrBlob	A pointer to the BLOB.

Description

See iBlob2Img()

Example

See iBlob2Img()

iPack() DLL Functions

Purpose

Compress a Character string

Syntax

iPack(*cToBeCompressed*) ->*cCompressed*

Arguments

cToBeCompressed	Character string to be compressed

Returns

cCompressed	The compressed character string

Description

This function is used in conjunction with iUnPack() to compress character strings.

There is no direct connection between compressing data using iPack() and iUnPack() and image manipulation. But since many languages need the ability to work with character strings instead of pointers, Light Lib Images includes the ability to compress and uncompress character data which in turn can represent images. These functions are very useful in dealing with large images. iPack() uses a LZW algorithm which is efficient on strings with sizes greater than 256 Characters.

Purpose

Send an image to a device.

Syntax

```
iPut(   dwLLImage     AS DWORD,
        siX1          AS SHORTINT,
        siY1          AS SHORTINT,
        siX2          AS SHORTINT,
        siY2          AS SHORTINT,
        dwDevice      AS DWORD,
        dwFormat      AS DWORD,
        siOffsetX     AS SHORTINT,
        siOffsetY     AS SHORTINT,
        dwParam1      AS DWORD,
        dwParam2      AS DWORD,
        dwParam3      AS DWORD,
        dwParam4      AS DWORD,
        dwParam5      AS DWORD
        dwUserParam   AS DWORD ) -> nil
```

Arguments

dwLLImage	Pointer to the image structure containing the image. This pointer must be the result of iGet() or iCopy()
siX1, siY1, siX2, siY2	These values represent the coordinates of the image to send to the device. These coordinates are usually specified when working on a portion of the image. You can use the predefined value LLI_FULL_SIZE to process the entire image. Note that these are the coordinates of the image and not of the destination device. To shift or offset the origin of the image on the destination device, you must use *siOffsetX* and *siOffsetY.*
dwDevice	Device to which the image data is sent. Select one of the above bitmaps for details on specific devices.
dwFormat	The type of device, to which the data will be sent. *wFormat* depends on the value of *wDevice.* The following are the possible predefined combinations of these 2 parameters.

wDevice	*wFormat*
LLI_DISK	LLI_DISK_BMP
	LLI_DISK_GIF
	LLI_DISK_JPG
	LLI_DISK_PCX
	LLI_DISK_TIF
LLI_MEMORY	LLI_MEMORY_BW
	LLI_MEMORY_16
	LLI_MEMORY_256
	LLI_MEMORY_16M

	LLI_PRINTER	LLI_PRINTER
	LLI_SCREEN	LLI_SCREEN_DEVICE_CONTEXT LLI_SCREEN_WINDOW_HANDLE
siOffsetX, siOffsetY	These values represent the offset coordinates on the output device.The values 0,0 represent no offset. These coordinates are generally used to move or offset the image on the target device. To use a portion of the image use the siX1, siY1, siX2, siY2	
dwParam1- dwParam5	These parameters depend on the device. Select the specific Device for further help.	
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.	

Returns

NIL

Description

iPut() is one of the two main Light Lib Images functions, along with iGet() .

iPut() lets you send image data to any device (Disk, Screen, Printer). The data written is always in the form of a rectangle.

iUnPack()   DLL Functions

Purpose

Uncompress a compressed Character string.

Syntax

iUnPack(*cCompressed*) -> *cUnCompress*

Arguments

cCompress	Character string previously compressed using iPack()

Returns

cUnCompress	An uncompressed Character string of a previously compressed string.

Description

This function is used in conjunction with iPack() to restore compressed Character strings to its original value.

There is no direct connection between compressing data using iPack() and iUnPack() and image manipulation. But since many languages need the ability to work with character strings instead of pointers, Light Lib Images includes the ability to compress and uncompress character data which in turn can represent images. These functions are very useful in dealing with large images. iPack() uses a LZW algorithm which is efficient on strings with sizes greater than 256 Characters.

iGet() Disk DLL Functions

Purpose

Retrieve image data from a Disk.

Syntax

```
iGet( dwLLOwner    AS DWORD,
      LLI_DISK,
      dwFormat     AS DWORD,
      siX1         AS SHORTINT,
      siY1         AS SHORTINT,
      siX2         AS SHORTINT,
      siY2         AS SHORTINT,
      dwParam1     AS DWORD,
      dwParam2     AS DWORD,
      dwParam3     AS DWORD,
      dwParam4     AS DWORD,
      dwParam5     AS DWORD,
      dwUserParam  AS DWORD) -> ptrImg
```

Arguments

dwLLOwner A reference to the application handle created with oNew()

dwFormat The image or document file type. The following are the predefined values:

LLI_DISK_AUTO	Format is determined based on file name extension
LLI_DISK_BMP	BMP file format
LLI_DISK_GIF	GIF file format
LLI_DISK_JPG	JPG file format
LLI_DISK_PCX	PCX file format
LLI_DISK_TIF	TIF file format

siX1, siY1, siX2, siY2 These are coordinates that represent a rectangle of the image to be retrieved from the file. You can use the predefined LLI_FULL_SIZE value to specify the entire image. Devices use pixels to define these coordinates.

IdwParam1 Image file name.

dwParam2-dwParam5 Not used. Substitute each with LLI_VOID_PARAM

dwUserParam This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

ptrImg Pointer to an image structure containing the image. If there is an error the value returned is NIL.

Description

iGet() and iPut() are the two main functions of Light Lib Images.

iGet() lets you read an image from disk. The image is always read in the form of a rectangle.

Examples

```
// This example shows how to retrieve a TIF image
// stored on a hard disk whose name is MyImage.TIF :

ptrImg = iGet(    dwAppLliHnd,
                  LLI_DISK,        // Device
                  LLI_DISK_TIF,          // File format
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  "MyImage.TIF", // Name of the image File
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM
                  dwUserParam )
```

iGet() Memory DLL Functions

Purpose

Retrieve image data from memory. Reserves unused memory, (presizing a new image)

Syntax

iGet(	*dwLLOwner*	AS DWORD,
	LLI_MEMORY,	
	dwFormat	AS DWORD,
	siX1	AS SHORTINT,
	siY1	AS SHORTINT,
	siX2	AS SHORTINT,
	siY2	AS SHORTINT,
	dwParam1	AS DWORD,
	dwParam2	AS DWORD,
	dwParam3	AS DWORD,
	dwParam4	AS DWORD,
	dwParam5	AS DWORD,
	dwUserParam	AS DWORD) -> *ptrImg*

Arguments

dwLLOwner	A reference to the application handle created with oNew()
wDevice	LLI_MEMORY
wFormat	The following are the possible predefined combinations of values for these parameters:

LLI_MEMORY	LLI_MEMORY_BW
	LLI_MEMORY_16
	LLI_MEMORY_256
	LLI_MEMORY_16M

siX1, siY1, siX2, siY2	Represent the coordinates of the data rectangle retrieved from the device. You can use LLI_FULL_SIZE to specify the entire device.
dwParam1-dwParam5	The extra parameters depend on the device. Select the specific device for further info.
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

ptrImg	Pointer to an image structure containing the image. If there is an error the value returned is NIL.

Description

iGet() and iPut() are the two main functions of Light Lib Images.

iGet(***LLI_MEMORY***) lets you read an image in memory.The data read is always in the form of a rectangle.

Examples

```
// This example shows how to create an empty image in memory.

ptrCopyImg = iGet( dwLLOwner,
                  LLI_MEMORY,
                  LLI_MEMORY_256,
                  0,
                  0,
                  1000,
                  1000,
                  LLI_VOID_PARAM,
            LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
LLI_VOID_PARAM
                  dwUserParam)
```

Purpose

Retrieve image/document data from a scanner.

Syntax

```
iGet(   dwLLOwner     AS DWORD,
        LLI_SCANNER,
        dwFormat      AS DWORD,
        siX1          AS SHORTINT,
        siY1          AS SHORTINT,
        siX2          AS SHORTINT,
        siY2          AS SHORTINT,
        dwParam1      AS DWORD,
        dwParam2      AS DWORD,
        dwParam3      AS DWORD,
        dwParam4      AS DWORD,
        dwParam5      AS DWORD,
        dwUserParam   AS DWORD) -> ptrImg
```

Arguments

dwLLOwner	A reference to the application handle created with oNew()
dwFormat	The type of scanner. The following are the available values: LLI_SCANNER_TWAIN This allows you to set the following extra parameters. End users will not be able to modify anything. LLI_SCANNER_TWAIN_DIALOG This will present users with the Twain scanning dialog box from the installed scanner. End users will be able to set all available scanner options.
siX1, siY1, siX2, siY2	These values represent the coordinates of the data rectangle retrieved from the device. Youcan use the predefined LLI_FULL_SIZE value to specify the entire device. Coordinates are in pixels. As the number of pixels involved depends on the Scan Density, use millimeters to define these coordinates.
dwParam1	The scan **density**. *dwParam1* can have one of the predefined density values: LLI_DPI_75 LLI_DPI_100 LLI_DPI_150 LLI_DPI_300 LLI_DPI_600 LLI_DPI_1200
dwParam2	The scan **luminosity**. *dwParam2* is represented in % and ranges from-100% to +100%. A value of 0 corresponds to a standard image. To obtain a darker document, use a negative value. To obtain a lighter document, use a positive value.
dwParam3	The scan **contrast**. *dwParam3* is represented in % and ranges from -100% to

	+100%. This value has an effect on color scans only. A value of 0 corresponds to a standard image. For more contrast, use a positive value. For less contrast, use a negative value.
dwParam4-dwParam5	Not used. Substitute with LLI_VOID_PARAM
dwUserParam	This parameter is passed to yourIdle callback function. It should be a pointer to any kind of structure.

Returns

ptrImg	Pointer to an image structure containing the image. If there is an error the value returned is NIL.

Description

iGet() and iPut() are the two main functions of Light Lib Images.

iGet(**LLI_SCANNER**) lets you read an image/document directly from a scanner. The data read is always in the form of a rectangle. The value returned is always a pointer to structure containing the image.

Examples

```
// This example shows how to read an entire document
// from a TWAIN compatible scanner using default values
// for for luminosity and contrast.

ptrImg = iGet(    dwAppLliHnd,
                  LLI_SCANNER,
                  LLI_SCANNER_TWAIN,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_DPI_600,  // Density
                  0,                  // Luminosity (0 is average)
                  0,                  // Contrast (0 is average)
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM.
                  dwUserParam )
```

iGet() Screen DLL Functions

Purpose

Retrieve image data from the screen.

Syntax

```
iGet(   dwLLOwner    AS DWORD,
        LLI_SCREEN,
        wFormat      AS WORD,
        siX1         AS SHORTINT,
        siY1         AS SHORTINT,
        siX2         AS SHORTINT,
        siY2         AS SHORTINT,
        dwParam1     AS DWORD,
        dwParam2     AS DWORD,
        dwParam3     AS DWORD,
        dwParam4     AS DWORD,
        dwParam5     AS DWORD,
        dwUserParam  AS DWORD) -> ptrImg
```

Arguments

dwLLOwner A reference to the application handle created with oNew()

wFormat

LLI_SCREEN_WINDOWS_HANDLE	Window handle
LLI_SCREEN_DEVICE_CONTEXT	Device context

siX1, siY1, siX2, siY2 These values represent the coordinates in pixels of the data rectangle retrieved from the device. You can use the predefined LLI_FULL_SIZE value to specify the entire device.

dwParam1 This parameter depends on the value of *wFormat*.

wFormat	*dwParam1*
LLI_SCREEN_WINDOW_HANDLE	Window handle
LLI_SCREEN_DEVICE_CONTEXT	Device Context handle

dwParam2 The type of color palette to use. The following are valid

LLI_PALETTE_SHARED
LLI_PALETTE_EXCLUSIVE

This parameter is very important if windows is being run in 256 color mode or less and you need to display 2 images simultaneously. See Common Problems for details.

dwParam3-dwParam5 Not used. Substitute each with LLI_VOID_PARAM

dwUserParam This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

ptrImg Pointer to an image structure containing the image. If there is an error NIL is returned.

Description

iGet() and iPut() are the two main functions of Light Lib Images.

iGet(***LLI_SCREEN***) reads image data on a screen. The data read is always in the form of a rectangle.

Examples

```
// This example shows how to read an image from a window.

ptrImg= iGet(     dwAppLliHnd,
                  LLI_SCREEN,
                  LLI_SCREEN_WINDOW_HANDLE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_FULL_SIZE,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM,
                  LLI_VOID_PARAM
                  dwUserParam )
```

Purpose

Save or send an image to disk. Open and write to an image file.

Syntax

iPut(	*dwLLImage*	AS DWORD,
siX1	AS SHORTINT,	
siY1	AS SHORTINT,	
siX2	AS SHORTINT,	
siY2	AS SHORTINT,	
LLI_DISK,		
dwFormat	AS DWORD,	
siOffsetX	AS SHORTINT,	
siOffsetY	AS SHORTINT,	
dwParam1	AS DWORD,	
dwParam2	AS DWORD,	
dwParam3	AS DWORD,	
dwParam4	AS DWORD,	
dwParam5	AS DWORD ,	
dwUserParam	AS DWORD) -> nil	

Arguments

dwLLImage A reference to an image structure containing the image. This must be the result of iGet() or iCopy()

siX1, siY1, siX2, siY2 These values represent the coordinates of the image to be saved on disk. They are NOT optional. You can use the predefined value LLI_FULL_SIZE to work on the entire image. Note that these are the coordinates of the image to be saved. These coordinates should not be used when working on a portion of the image. To offset the position of the image on the destination disk file,you must use *siOffsetX* and *siOffsetY.*

dwFormat The image or document file type. The following are predefined values

LLI_DISK_BMP	BMP file format
LLI_DISK_GIF	GIF file format
LLI_DISK_JPG	JPG file format
LLI_DISK_PCX	PCX file format
LLI_DISK_TIF	TIF file format

siOffsetX, siOffsetY Represent the coordinates of a starting point offset in the disk file. They are NOT optional. The values 0,0 represent no offset.

These are the coordinates of the disk file and not those of the image. These coordinates are generally used to offset the image in the disk file. To save only a portion of the image use the siX1, siY1, siX2, siY2.

dwParam1 Pointer to the image's complete file name.

dwParam2 Compression Type.

	Note: that the compression used must be compatible with the format. See Editions/Versions for the Light Lib Images Compression Chart. One of the following values are valid for *dwParam3* LLI_DISK_COMPRESS_NIL LLI_DISK_COMPRESS_AUTO LLI_DISK_COMPRESS_CCITT1D LLI_DISK_COMPRESS_CCITTG3 LLI_DISK_COMPRESS_CCITTG4 LLI_DISK_COMPRESS_LZW LLI_DISK_COMPRESS_RLE
dwParam3-dwParam5	Not used. Substitute each with LLI_VOID_PARAM
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

NIL

Description

iPut() is one of the two main Light Lib Images functions, along with iGet(). iPut() lets you write image data to Disk. The data written is always in the form of a rectangle.

Example

This example shows how to save a compressed TIF image to the disk:

```
iPut(           ptrImg,                 // Image pointer
                LLI_FULL_SIZE,
                LLI_FULL_SIZE,
                LLI_FULL_SIZE,
                LLI_FULL_SIZE,
                LLI_DISK,               // Disk Device
                LLI_DISK_TIF,           // Type of compression to use
                0,
                0,
"MyImage.TIF", // Name of the file
                LLI_DISK_COMPRESS_RLE,
                LLI_VOID_PARAM,
                LLI_VOID_PARAM,
                LLI_VOID_PARAM
dwUserParam)
```

iPut() Memory DLL Functions

Purpose

Create an image in memory.

Syntax

```
iPut(                *dwLLImage*    AS DWORD,
        siX1         AS SHORTINT,
        siY1         AS SHORTINT,
        siX2         AS SHORTINT,
        siY2         AS SHORTINT,
        LLI_MEMORY,
        dwFormat     AS DWORD,
        siOffsetX    AS SHORTINT,
        siOffsetY    AS SHORTINT,
        dwParam1     AS DWORD,
        dwParam2     AS DWORD,
        dwParam3     AS DWORD,
        dwParam4     AS DWORD,
        dwParam5     AS DWORD ,
        dwUserParam  AS DWORD) -> nil
```

Arguments

dwLLImage	A reference to an image structure containing the image. This must be the result of iGet() or iCopy()
siX1, siY1, siX2, siY2	These values represent the coordinates of the image to be saved on disk. They are NOT optional. You can use the predefined value LLI_FULL_SIZE to work on the entire image. Note that these are the coordinates of the image to be saved. These coordinates should not be used when working on a portion of the image. To offset the position of the image on the destination disk file, you must use *siOffsetX* and *siOffsetY.*
dwFormat	
siOffsetX, siOffsetY	Represent the coordinates of a starting point offset in memory. They are NOT optional. The values 0,0 represent no offset. These are the coordinates of the image and not those of the image. These coordinates are generally used to offset the image in memory. To use only a portion of the image use the siX1, siY1, siX2,siY2.
dwParam1	
dwParam2	Compression Type.
dwParam3-dwParam5	Not used. Substitute each with LLI_VOID_PARAM
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

NIL

Description

iPut() is one of the two main Light Lib Images functions, along with iGet(). iPut() lets you put image data to memory. The data written is always in the form of a rectangle.

iPut() Printer DLL Functions

Purpose

Send an image or portion of an image to a printer.

Syntax

```
iPut(                dwLLImage     AS DWORD,
        siX1         AS SHORTINT,
        siY2         AS SHORTINT,
        siX2         AS SHORTINT,
        siY2         AS SHORTINT,
        LLI_PRINTER,
        dwFormat     AS DWORD,
        siOffsetX    AS SHORTINT,
        siOffsetY    AS SHORTINT,
        dwParam1     AS DWORD,
        dwParam2     AS DWORD,
        dwParam3     AS DWORD,
        dwParam4     AS DWORD,
        dwParam5     AS DWORD
        dwUserParam  AS DWORD) -> nil
```

Arguments

dwLLImage	A reference to an image structure containing the image. This must be the result of iGet() or iCopy().
siX1, siY1, siX2, xiY2	Coordinates of the image to be sent to the printer. They are NOT optional. You can use the predefined value LLI_FULL_SIZE to send the entire image. These are the coordinates of the image, not of the device. These coordinates are used when working on a portion of the image. To specify an offset on the printer use *siOffsetX* and *siOffsetY.*
dwFormat	
siOffsetX, siOffsetY	The coordinates of the projection point on the printer. They are NOT optional. The values 0,0 indicate no offset. These are the coordinates of the device and not those of the image. These coordinates are generally used to shift or offset the image on the printer. To send only a portion of the image use siX1, siY1, siX2, siY2
dwParam1	Printer port to be use in printing the image. LLI_PRINTER_LPT1 LLI_PRINTER_LPT2
dwParam2	siDensity LLI_DPI_75 LLI_DPI_100 LLI_DPI_150 LLI_DPI_300 LLI_DPI_600 LLI_DPI_1200

dwParam3	siEjection LLI_PRINTER_EJECT LLI_PRINTER_NO_EJECT
dwParam4	siPageFormat LLI_PRINTER_PAGE_EXECUTIVE LLI_PRINTER_PAGE_LETTER LLI_PRINTER_PAGE_NOTE LLI_PRINTER_PAGE_LEGAL LLI_PRINTER_PAGE_A3 LLI_PRINTER_PAGE_A4 LLI_PRINTER_PAGE_A5 LLI_PRINTER_PAGE_A6 LLI_PRINTER_PAGE_B4
dwParam5	LLI_VOID_PARAM The image density is stored in the image structure member *LLI_IMG_DPI*. If an images was created by scanning, then the density is known. When the image is read in from disk, the density depends on the method used at the time of saving. When Light Lib Images cannot determine the density of an image, 300 DPI is used. *siDensity* is very important for preserving document quality. For example, if you scan a document at 150 Dpi and don't use the parameter LLI_DPI_150 to tell the printer to go into 150 DPI mode, it will remain in its current mode (probably 300 DPI) and your printed image will represent only the upper left quarter of the page, instead of the whole page. *siEjection* Eject page after image printing. In the most common case (PCL4), ejecting a page involves sending a CHR(12) to the printer. In the POSTSCRIPT language you need to issue a *ShowPage* message to the printer. *siPageFormat* Paper format in printer.Accepts the following parameters : Light Lib Images uses an origin located at the top-left of the device. In Postscript language origin is located at the bottom-left. When you specify the printer paper format, you allow Light Lib Images to print transparently whenever you want to use a PCL or a PostScript device.
dwUserParam	This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

NIL

Description

iPut() is one of the two main Light Lib Images functions, along with iGet(). iPut() lets you send an image to a printer.

Example

```
// This example shows how to send an image to a printer.

iPut(       ptrImg,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_PRINTER,
            LLI_PRINTER_PCL4_BW,
            0,
            0,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM
            dwUserParam )
```

Purpose

Display an image on the screen.

Syntax

```
iPut(                   dwLLImage      AS DWORD,
        siX1            AS SHORTINT,
        siY2            AS SHORTINT,
        siX2            AS SHORTINT,
        siY2            AS SHORTINT,
        LLI_SCREEN,
        dwFormat        AS DWORD,
        siOffsetX       AS SHORTINT,
        siOffsetY       AS SHORTINT,
        dwParam1        AS DWORD,
        dwParam2        AS DWORD,
        dwParam3        AS DWORD,
        dwParam4        AS DWORD,
        dwParam5        AS DWORD,
        dwUserPararm    AS DWORD) -> nil
```

Arguments

dwLLImage A reference to a structure containing the image. This must be the result of iGet() or iCopy().

siX1, siY1, siX2, siY2 Coordinates of the image to be displayed. They are NOT optional. You can use the predefined value LLI_FULL_SIZE to work on the entire image. These are the coordinates of the image to display, not of the screen. These coordinates are usually used when working on a portion of the image. To shift or offset the position of the image on the screen, use *siOffsetX* and *siOffsetY.*

dwFormat The following are valid pre-defined values

LLI_SCREEN_WINDOW_HANDLE	Window handle
LLI_SCREEN_DEVICE_CONTEXT	Device Context

siOffsetX, siOffsetY The coordinates of the projection point on the screen. They are NOT optional. The values 0,0 represent no offset. These are the coordinates on the screen and not those of the image. These coordinates are generally used to move or offset the image on the screen. To use only a portion of the image use siX1, siY1, siX2, siY2

dwParam1 Depending on *dwFormat*, this can be one of the following:

wFormat	*dwParam1*
LLI_SCREEN_WINDOW_HANDLE	Window handle
LLI_SCREEN_DEVICE_CONTEXT	Device Context handle

dwParam2-dwParam5 Not used. Substitute each with LLI_VOID_PARAM

*dwUserParam*This parameter is passed to your Idle callback function. It should be a pointer to any kind of structure.

Returns

NIL

Description

iPut() is one of the two main Light Lib Images functions, along with iGet() .

Notes

Light Lib Images has no problem displaying a black and white image in 16 or 256 color mode since colors 0 and 1 are used to display black and white pixels respectively.

Another problem arises when you attempt to display simultaneously a 256 color image with many shades of yellow and a 256 color image with many shades of green.

If you use LLI_PALETTE_EXCLUSIVE, Light Lib Images will use 256 yellow levels to display the yellow picture. This guarantees a very good display. But if you display the green image with the same LLI_PALETTE_EXCLUSIVE parameters, the palette, which is the same for the entire screen, will change to be optimized for the green image and the yellow image already being displayed will be altered.

The solution is to use LLI_PALETTE_SHARED parameters. In this case, Light Lib Images will use a 256 color linear palette, and all images displayed at the same time on the screen will share the same linear palette. You will not get quite as good a display as when the yellow or green image is displayed by itself, but the result will be realistic.

In summary, to display one image at a time, use LLI_PALETTE_EXCLUSIVE for the best possible result. This is the most common case. To display images simultaneously, use LLI_PALETTE_SHARED.

Example

```
// This example shows how to display an image on screen.

iPut(       ptrImg,                  // Image pointer
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_FULL_SIZE,
            LLI_SCREEN,                    // Device
            LLI_SCREEN_WINDOW_HANDLE,     // Type of device
            0,
            0,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            LLI_VOID_PARAM,
            dwUserParam )
```

Who is DFL? With corporate offices in Toronto, Paris and a research and development center in the south of France, DFL is fast becoming a leading developer of advanced add-on products for Windows and DOS. We are committed to providing the very best tools for serious software development. Let DFL's *Light Lib* family of products help you develop better applications.

Thank you for your support,
The DFL Team.

Light Lib Library User License

This document is a contract between you (the licensee) and DFL.

DFL supplies the software and grants a license for its use. You are totally responsible for choosing to use the software for the desired purpose, as well as for installing and using it, and for the results obtained.

DFL grants no rights for the software other than those explicitly stipulated in this agreement, and reserves all rights not explicitly granted to the licensee.

License

DFL grants to the licensee a non-exclusive and non-transferable right to use the software for an unlimited duration. The licensee may make ONE copy of the software in readable form on a computer or other support for backup purposes.

You have the right to load the software into RAM and use it on a single computer, to install it on the hard disk of the computer, to produce executable files (.EXE) containing the Light Lib software.

You are not required to pay royalties to DFL on the sale of any applications using a Light Lib product. If you expect to sell or distribute more than 500 copies of a commercial product which uses one or more Light Lib products, it is required that you inform DFL and that you grant DFL permission to publicize the fact that your product uses Light Lib technology.

Forbidden

The following are forbidden: sharing of the software on a network: each programmer that uses Light Lib software in part or in whole,must possess his or her own registered license, distribution of the software, decompilation of supplied files, disassembly of supplied files, rewriting and any form of reverse engineering of the software, and any other action or activity involving the software that is not explicitly authorized.

Term

The license shall remain in force until it is cancelled. The licensee may cancel it at any time by destroying the software and all copies. It is also cancelled without notification if the licensee violates any terms or conditions of the present agreement. The licensee agrees to destroy the software and all copies if the agreement is cancelled.

Limited warranty

The software is furnished as is, with no warranty of any sort, explicit or implicit, including, but not restricted to, implicit warranties as to its fitness or sale for any particular use. Any risks stemming fromthe quality and performance of the program are entirely bourne by the licensee. If there is any defect in the program, the licensee (and not DFL or any intermediary) will assume any expenses for help, repair, or correction.

DFL does not guarantee that the functions included in the programs correspond to the needs of the licensee or that the program will function without interruptions or errors.

Nevertheless, DFL warrants that the diskettes on which the program is supplied are without defects in material and packaging for a period of ninety (90) days, to be counted from the date of delivery to the licensee as indicated on the bill of sale.

The only responsibility of DFL and the only compensation due to the licensee are:

1) Replacement of any diskette not in conformity with the limited warranty of DFL and which has been returned to DFL or an intermediary authorized by DFL, with a copy of the licensee's receipt, or,

2) If DFL or an authorized intermediary is unable to supply a diskette free of material or packaging defects the licensee may cancel this agreement by returning the software, and the licensee will be reimbursed.

In no case will DFL be responsible to the licensee for damages of any kind, including loss of profits or savings or direct or indirect loss resulting from use of or inability to use the program, even if DFL or an authorized intermediary of DFL was informed of the possibility of such damages, or of any claim forany third party.

General considerations

You certify that you have read and understood the present agreement, and that you agree to be bound by its terms and conditions. You also recognize that it constitutes the sole and exclusive basis for our contract, replacing any earlier proposal or contract, verbal or written, and any other communication between us relating to the object of the present agreement.

Registered trademarks

All trademarks are the property of their registered owners.

Technical Support

The three primary means of technical support are via Compuserve, FAX and on our BBS. If it is an emergency, you can call or fax us.

North America

DFL Software Inc.	Voice (416) 789-2223
1712 Avenue Road	Fax (416) 789-0204
Box 54616	BBS (416) 784-9712
Toronto, ON, M5M 4N5	Compuserve 74723,3321
CANADA	Internet 74723.3321@compuserve.com

Europe

DFL Europe	Voice (33 1) 46 05 20 66
39-41, rue de la Saussière	Fax (33 1) 46 04 10 39
Boulogne	BBS (33 1) 46 05 26 88
FRANCE	Compuserve 100067,652
	Internet 100067.652@compuserve.com

products by DFL

All Light Lib products have been designed and developed to be implemented easily and execute quickly .

Windows	Light Lib Business
	Light Lib Images
	Light Lib Multimedia
DOS	Light Lib Business
	Light Lib Images
	Light Lib Graphics

Light Lib Business is a revolutionary graphing library. It provides the unprecedented power to present users with "live" graphs. Your users will now be able to dynamically scroll and interact with graph data as if they were scrolling text data. The days of static graphs are over!

Light Lib Images is the most comprehensive image and document managing library available. Scanning, loading, saving, printing images or documents has never been easier.

Light Lib Multimedia is the easiest-to-use multimedia library for Windows. Adding the ability to play or record sound and display video, will bring your applications to new heights.

Light Lib Graphics for CA-Clipper is the first Replaceable Terminal Driver (RTD) for CA-Clipper. It will immediately transforms your text mode applications into graphic mode.

All Light Lib products for DOS are upward compatible with their Windows counterpart. Each product comes with complete help files and source code to the extensive support functions and classes.

All Light Lib products for CA-Clipper are fully compatible with Real and Protected mode linkers (Exospace, Blinker and Causeway) and each product is fully integrated with CA-Clippper's VMM system.

Light Lib Objects Functions

DLL Functions

Constants

Light Lib Objects (LLO)

Light Lib Objects is not another Light Lib product. LLO manages memory allocation and the proper creation and deletion of all objects within the Light Lib DLLs themselves. Every Light Lib product for Windows relies on this support DLL. Please review the specific language implementation carefully because the usage of LLO differs slightly from language to language.

LLO provides object oriented technology to languages that do not support object oriented programming and provides enhanced features to languages that support OOP. In addition to standard OOP features such as inheritance, polymorphism, and encapsulation, LLO implements advanced OOP concepts such as inheriting from an owner class which is not the immediate parent, dynamic class creation, BLOB aggregation and much more. The following is an example:

ABSTRACT Class - GRAPH Class

ABSTRACT Class - COLUMN Class

There is no relationship between the GRAPH Class and the COLUMN Class. However, if a method or property is not available in an instance of the COLUMN Class, LLO will not use the ABSTRACT parent class definition, which is how OOP systems work today. Instead, LLO is able to use the class Owner's definition which could, for example, be a GRAPH.

Register and unregister an application

You need to call dwLightLibAppRegister() at the start of your program. This allows the Light Lib DLLs to be properly initialized. If this registration is not executed, you will receive errors.

At the end of execution, you will need to unregister your application with the Light Lib DLLs by calling dwLightLibAppUnRegister().

Light Lib Objects Constants

Class Constants

LLO_CLASS_ABSTRACT	Abstract Class (Hidden)
LLO_CLASS_APPLICATION	Application Class
LLO_CLASS_CONTEXT	Context Class (Hidden)
LLO_CLASS_ERROR	Error Class

Abstract Constants

LLO_ABSTRACT_APPLICATION
LLO_ABSTRACT_CARGO
LLO_ABSTRACT_CARGO_COUNT
LLO_ABSTRACT_CLASS_ID
LLO_ABSTRACT_CLASS_NAME
LLO_ABSTRACT_CLASS_VERSION
LLO_ABSTRACT_ERROR
LLO_ABSTRACT_LIBRARY_ID
LLO_ABSTRACT_LIBRARY_NAME
LLO_ABSTRACT_LIBRARY_VERSION
LLO_ABSTRACT_OWNER

Application Constants

LLO_APPLICATION_CARGO_COUNT_DEFAULT
LLO_APPLICATION_CONTEXT
LLO_APPLICATION_HANDLE
LLO_APPLICATION_NAME

Error Constants

Error Class

LLO_ERROR_ACTION	
LLO_ERROR_OBJECT	
LLO_ERROR_MESSAGE	Error message
LLO_ERROR_NUMBER	
LLO_ERROR_PARAM	Extended depending on Error Type
LLO_ERROR_PROPERTY	Property define#
LLO_ERROR_PROPERTY_NAME	Property name

LLO_ERROR_NUMBER

LLO_ERROR_CARGO_OUT_OF_LIMIT
LLO_ERROR_INVALID_CLASS_DEFINE
LLO_ERROR_INVALID_OWNER_TYPE
LLO_ERROR_INVALID_PARAMETERS
LLO_ERROR_INVALID_ACCESS_NEW
LLO_ERROR_INVALID_ACCESS_DEL
LLO_ERROR_INVALID_ACCESS_ACCESS
LLO_ERROR_INVALID_ACCESS_ASSIGN
LLO_ERROR_MEMORY_ALLOCATION
LLO_ERROR_NO_ERROR
LLO_ERROR_OBJECT_ACCESS_DENIED
LLO_ERROR_OBJECT_ASSIGN_DENIED
LLO_ERROR_READONLY_PROPERTY
LLO_ERROR_UNDEFINED_PROPERTY

LLO_ERROR_ACTION

LLO_ACTION_ACCESS
LLO_ACTION_ASSIGN
LLO_ACTION_DEL
LLO_ACTION_NEW

User Defined Constants

LLI_UDF_ABORT	User Defined `Function Abort return value`
LLI_UDF_CONT	`User Defined Function Continue return value`
LLI_UDF_ERROR	`Error append during a Light Lib function execution`
LLI_UDF_EXIT	`Exit phase for a Light Lib function execution`
LLI_UDF_IDLE	`Idle phase for a Light Lib function execution`
LLI_UDF_INIT	`Init phase for a Light Lib function execution`

Overview

Light Lib Objects

oAccess() DLL Functions

Purpose

Access an object's instance variable. See also Light Lib Objects

Syntax

```
oAccess(   dwLLObject    AS DWORD,
           dwProperty    AS DWORD,
           dwExtraParam  AS DWORD ) ---> dwData
```

Arguments

dwLLObject	A Light Lib object.
dwProperty	A property belonging to this Light Lib object.
dwExtraParam	Used to access the LLI_IMAGE_CARGO value. For example, if LLI_IMAGE_CARGO is a structure, *dwExtraParam* would represent the byte offset into the structure.

Returns

dwData	The value of the requested object member

Description

dwExtraParam must be cast to DWORD. This allows the Light Lib DLL to pass a POINTER, SHORTINT, LONGINT etc.

Examples

```
// This returns the name of the class to which the object belongs.
oAccess( dwMyObject, LLO_ABSTRACT_CLASS_NAME, 0 )

// This returns the value of the second cargo
// instance variable for this object.
 oAccess( dwMyObject, LLO_ABSTRACT_CARGO, 2 )
```

oAssign()   DLL Functions

Purpose

Assign any value to a defined variable of an object.See also Light Lib Objects

Syntax

```
oAssign(    dwLLObject    AS DWORD,
            dwProperty    AS DWORD,
            dwValue       AS DWORD,
            dwExtraParam  AS DWORD ) ---> liError
```

Arguments

dwLLObject	A Light Lib object
dwProperty	The predefined value to change. You can only change or assign to the symbols noted as Assignable. You are not able to modify symbols that are Read Only symbols.
dwValue	The value to be assigned.
dwExtraParam	Used to access the LLI_IMAGE_CARGO value. For example, if LLI_IMAGE_CARGO is a structure, *dwExtraParam* would represent a byte offset into the structure.

Returns

liError	An error code.

Description

The *dwExtraParam* and *dwValue* must be cast to DWORD.This allows the DLL to pass a POINTER, SHORINT, LONGINT etc.

Examples

```
// This sets the cargo size for this object to 4 DWORD.
oAssign(dwMyObject, LLO_ABSTRACT_CARGO_SIZE, 4 )

//This sets the second cargo instance variable to dwMyValue.
oAssign(dwMyObject, LLO_ABSTRACT_CARGO, dwMyValue, 2 )
```

oNew() DLL Functions

Purpose

Used to create a new Application Object. See also Light Lib Objects

Syntax

```
oNew(          dwLLClass      AS DWORD,
               dwLLObject     AS DWORD,
               siSizeOfCargo  AS SHORTINT,
               dwValue        AS DWORD
               dwExtraParam   AS DWORD ) ---> ptrAppHnd
```

Arguments

dwLLClass	Represents the class of the object to be created.
dwLLObject	Represents the object to be created. If the class to which the object belongs is an application, the dwLLObject doesn't need to be defined (pass zero).
siSizeOfCargo	The size or number of DWORD parameters in an object's cargo.
dwValue	This is an optional value containing extra information. For example, when you create a new Column object inside a Graph object, *dwValue* dictates where the column should be inserted. If *dwValue* is 0, the new column becomes the last column. If *dwValue* is an existing Column number, the new Column is inserted before the passed number.
dwExtraParam	An optional parameter.

Returns

dwAppHnd	A pointer to a Light Lib Objects application handle.

Description

This allows you to register a Light Lib application with the Light Lib Objects DLL. This registration allows the Light Lib DLL to be used simultaneously by several applications in a multitasking operating system and to automate memory garbage collection. You must ensure that your applications always terminate with oDel().

When you create an application that uses aLight Lib DLL, you need to register that application with the Light Lib Objects DLL. This needs to be done at the very start of your application by calling oNew() and by passing the proper arguments.

Once registered, Light Lib Objects, automatically keeps track of all objects created within the registered application. This guarantees that all objects are properly connected to the Light Lib DLL.

When terminating an application, you need to unregister it from the Light Lib Images DLL with oDel(). This frees all memory allocated to objects in the application, even if the objects have not been explicitly erased. It is, however, always better to erase images from memory when they are no longer needed using oDel().

This oNew() and oDel() technique needs to be implemented to ensure that Light Lib Objects can properly manage all memory and processes when being called simultaneously from multiple applications. This is very important in a multitasking operating system.

oDel() DLL Functions

Purpose

Delete any Light Lib object. This frees all memory allocated to objects in a registered Light Lib application. See also Light Lib Objects

Syntax

oDel(*dwLLObject* AS DWORD) ---> dwAppHnd

Arguments

dwLLObject	A DWORD representing any Light Lib object.

Returns

dwAppHnd	An empty pointer to a Light Lib Images application handle.

Description

You must ensure that your Light Lib applications always terminate with oDel().

Be aware, that deleting a Light Lib object will also delete all of its child objects (if any) as well. As an example, deleting the Application object in turn deletes all objects created by that application from memory. It is highly recommended to delete the registered Application object by calling oDel() prior to exiting any Light Lib application.

This oNew() and oDel() technique (registering and unregistering) must be implemented to ensure that Light Lib Objects can properly manage the Light Lib DLLs when being called simultaneously from multiple applications.This is very important in a multitasking operating system.

dwLightLibApp() CA-Visual Objects

Purpose

Get the current Light Lib Application. See also Light Lib Objects

Syntax

dwLightLibApp() ---> *dwLightLibRegisteredApp*

Arguments

None.

Returns

dwLightLibRegisteredApp	The registered application.

dwLightLibAppRegister() CA-Visual Objects

Purpose

Register this instance of application into the LLO.DLL This must be done only once in an application's execution, and prior to any calls to the Light Lib library you are using. See also Light Lib Objects

Syntax

dwLightLibAppRegister(*oApp* AS OBJECT,
oWindow AS OBJECT) ---> *dwLightLibRegisteredApp*

Arguments

oApp	Application to register.
oWindow	Owner window.

Returns

dwLightLibRegisteredApp	The value of the registered application.

Description

This function is used to register your Light Lib application with Light Lib Objects. If this registration process is not done, your application will not work properly.

dwLightLibAppUnRegister()   CA-Visual Objects

Purpose

Unregister a Light Lib application from the Light Lib Objects DLL. See also Light Lib Objects

Syntax

dwLightLibAppUnRegister() ---> *dwLightLibRegisteredApp*

Arguments

None

Returns

dwLightLibRegisteredApp	Unregister an application.

Out of Memory

If you are experiencing memory problems in applications using Light Lib DLLs, there is a good chance that you are keeping unnecessary references to objects such as images or graphs in memory. When your application no longer needs an object, you should formally remove or delete it from memory by calling oDel() with the proper parameters.

DLL Crashes

This error could occur when multiple applications that use Light Lib DLLs are running simultaneously. In order to prevent conflicts between them, you must ensure that each application is registered with Light Lib Objects.

This involves making a call to oNew() with the proper parameters at the beginning of your program. Also, remember to make a call to oDel() just before your application terminates.

Unable to Load a DLL at Runtime

Make sure that the proper Light Lib DLL is available in your WINDOWS\SYSTEM directory. At installation time, Light Lib DLLs are installed to this directory. If they are not present when your application runs, the applications will cause a LoadError().