Website: www.chestysoft.com Email: info@chestysoft.com

csImageLite - Version 1.0
COM Object for Image Resize and Editing

This is a COM object that enables images of various formats to be created, resized or edited. It is a non-visual component developed for server side scripting, especially Active Server Pages, and functionality allows dynamically produced images to be sent straight to the browser. It can also be used by other COM enabled tools including Cold Fusion and ASP.NET.

A free, fully functional trial version of csImageLite is available. This trial version has a built in expiry date that causes most of the functions to stop working after that time. This is the only difference in functionality between the trial and full versions. This means that you can fully test if this component is suitable for your application before considering whether to license the full version.

The component is supplied as two different DLL files. One is 32 bit and the other 64 bit. Refer to the next section for more details of registration and component instantiation.

This component is based on our popular component, csImageFile, but it has been developed to offer a reduced level of functionality at a lower price.

Using These Instructions

These instructions are divided into a number of sections with the relevant methods and properties described in each. There are quick links to some sections below. A full Table of Contents is available on the next page and an index listing all commands in alphabetical order is included at the back for easy reference. The PDF version also has bookmarks for direct navigation to each heading.

Click on one of the links below to go directly to the section of interest:

Chestysoft, August 2019.
www.chestysoft.com

TABLE OF CONTENTS

1. Registering the Component and Getting Started

1.1. Registration and Server Permissions

Before the component can be used the DLL file, must be registered on the server. This can be done using the command line tool REGSVR32.EXE. Take care to use the correct version of this tool as there is a 64 bit version in the Windows\System32 folder and a 32 bit version in the Windows\SysWOW64 folder. The syntax is:

regsvr32 *dllname*

where *dllname* is the path and name of the DLL to register.

There will be two DLL files supplied in the zip archive, one for 32 bit systems and one for 64 bit. Unlike previous components that we have produced, both DLL files have the same name, so they must be stored in different folders. The command to instantiate the component is the same for both DLLs.

Chestysoft has a free utility that performs this function through a Windows interface instead of using regsvr32. This tool can be downloaded from the Chestysoft web site:
https://www.chestysoft.com/dllregsvr/default.asp

We suggest creating a folder specifically for component dlls rather than using the Windows System folder as this makes them easier to manage and avoids the naming confusion on 64 bit systems.

The application that uses the component must have permission to read and execute the DLL. In a web application like ASP this means giving the Internet Guest User account Read and Execute permission on the file. This account must also have the appropriate permissions for file handling. Read permission is required to read/open an image from disk. Write permission is required to create a new file and Modify is required to edit or delete an existing file. These permissions can be set in Windows Explorer and applied to either a folder or individual files.

1.2. Object Creation

In any script or programme that uses the component an object instance must be created. The syntax in ASP is as follows.

For the full version:

```
Set Image = Server.CreateObject("csImageLite.ImageFunctions")
```

For the trial version:

```
Set Image = Server.CreateObject("csImageLiteTrial.ImageFunctions")
```

In each case the object name is "Image", but any variable name could be used.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csImageLiteTrial. This trial version is fully functional but it has an expiry date, after which time it will stop working. The object can still be created after the expiry date but it cannot load or create images.

The expiry date can be found by reading the *Version* property.

Version - String, read only. This returns the version information and for the trial, the expiry date.

Example:

```
Set Image = Server.CreateObject("csImageLiteTrial.ImageFunctions")
Response.Write Image.Version
```

Visit the Chestysoft web site for details of how to buy the full version - https://www.chestysoft.com

1.4. Using csImageLite with Component Services

A COM component can be added to a COM+ Application in Component Services. One reason to do this is to be able to run a 32 bit DLL on a 64 bit system. Another is to specify a Windows account to use the component to allow that component to access network files that would be unavailable if the component was called by the default internet guest user.

An online description of configuring Component Services is available here:

https://www.chestysoft.com/component-services.asp.

On Windows 2008 and later it is necessary to "Allow intrinsic IIS properties" in the COM+ component properties. csImageLite will run without this but the *StreamToBrowser* method requires IIS intrinsic properties.

1.5. System Requirements

csImageLite requires Windows 2003 or later for a server or Windows XP or later for a desktop. It will not register or run on Windows 2000.

2. Import and Export of Images

An image must be loaded into memory if it is to be processed in some way or converted to another format. Images can be loaded from disk, from a variant array variable (which may be a database field or a file uploaded through our csASPUpload component), or by using the handle to a Windows bitmap.

The image in memory can be exported by saving to disk, or as a variant array variable (which may be sent to the browser using the ASP Response.BinaryWrite function, or saved into a binary database field). The image can be exported using the Windows bitmap handle and this is an efficient way of copying images between instances of csImageLite.

Supported file formats are BMP, GIF, JPG, PNG and TIF. Some description of these formats is provided in Section 8.

2.1. Methods for Import and Export

ReadFile *Path* - This loads an image into memory from disk where *Path* must be the full physical path and filename of the image.

Example:

```
Image.ReadFile "C:\images\somefile.jpg"
```

WriteFile *Path* - This saves the current image to disk, where *Path* is the full physical path and filename of the new file. The file extension determines the format used and it must be one of the supported formats. It can be a different extension to that of the original file and this is how images can be converted between formats.

ReadVariant *FileData* - This reads an image into memory from binary data. *FileData* must be a variant array containing the file information in one of the supported image formats. The variant array (or OLE Variant) is an ActiveX data type commonly used in ASP but it is not supported in all programming environments. *ReadVariant* may be used to read from our csASPUpload component, another instance of csImageLite, a VBScript variable or a binary database field.

Example of reading a file directly from csASPUpload:

```
Image.ReadVariant Upload.FileData(0)
```

This will read the file from the csASPUpload object called "Upload" assuming it is the first file in the array, or the only uploaded file. If the file is not a valid image it will generate an error.

Example of reading a file from a binary database field:

```
FileData = RS("Image")
Image.ReadVariant FileData
```

The data must first be passed to a VBScript variable before the *ReadVariant* command will accept it. The field name is called "Image" and the recordset is called "RS".

DPMToDPI(*dpm*) - Integer return value. Converts a dots per metre value, *dpm*, to dots per inch. *dpm* is an integer.

DPIToDPM(*dpi*) - Integer return value. Converts a dots per inch value, dpi, to dots per metre. *dpi* is an integer.

2.2. Properties for Import and Export

The following properties are related to the import and export of images and they include general properties such as *Width*, *Height* and *XDPI*. Some are specific to certain formats such as *JpegQuality* and *ProgressiveJpeg*. Some are set when the image is read or created while others can be set before export to alter some feature of the image.

Height - Integer, read only. The height of the currently loaded image in pixels. This is read only. To change the size use *Resize*, *Scale* or *ResizeFit*.

Width - Integer, read only. The width of the currently loaded image in pixels. This is read only. To resize the image use *Resize*, *Scale* or *ResizeFit*. To check if an image is loaded in memory, test for either the *Width* or *Height* having a non zero value.

ColorDepth - Integer, must be 1, 4, 8 or 24. This is the bits per pixel for the current image where 1 bit is a 2 colour image, 4 bit is 16 colours, 8 bit is 256 colours and 24 bit is 16 million RGB colour. These are the only colour depths supported by csImageLite. The property is set when an image is imported and it defaults to 24 for an image created with *NewImage*. *ColorDepth* can be edited. Reducing the colour depth may reduce the memory requirement of the image while some image processing requires a 24 bit setting. Use *ConvertToBW* to convert to 1-bit black and white.

XDPI - Integer. The horizontal pixel density of the image in dots per inch. Note that GIF images do not use this property and it is an optional extension for PNG.

YDPI - Integer. The vertical pixel density of the image in dots per inch.

JpegQuality - Integer in the range 1 to 100. This determines the amount of compression used when an image is exported in JPEG format. A high value is a high quality, large file size. Saving files with a lower value for *JpegQuality* is one method of reducing file size but it does reduce image quality. (Default = 90)

ProgressiveJpeg - Boolean. If true a JPG will be saved using progressive compression instead of baseline. This property is set when a JPG is loaded. (Default = false)

JpegGrayScale - Boolean. JPG images can be stored in an 8-bit greyscale format and this property is set when a JPG is loaded. Setting *JpegGrayScale* to true will convert a colour image to 8 bit greyscale and it will be exported in this format if saved as a JPG. (Default = false)

JpegHigherSpeed - Boolean. When set to true any JPG will be loaded using a reduced colour depth. The image will load at a higher speed at the expense of some image quality. Set the property before calling *ReadFile* or *ReadVariant*. (Default = false)

TransparentColor - String, as a six character "RRGGBB" colour. This is used as the transparent colour when an image is exported in GIF or PNG format, if the *Transparent* property is set. *TransparentColor* is set when a GIF or PNG image with transparency is loaded. *TransparentColor* is used in the merge functions when foreground transparency is used. It also determines the background colour of rotations that are not multiples of 90 degrees and when the *Crop* method is used to enlarge the image area. (Default = white, "FFFFFF")

*** Note *** Colours in csImageLite are always specified as a string value, in the same way as HTML. For example, red is "FF0000" and blue is "0000FF". These strings are not case sensitive and any leading "#" character will be automatically removed.

Transparent - Boolean. When true any image output in GIF or PNG format will contain a transparent colour and this colour is specified by *TransparentColor*. This property is set when a GIF or PNG is loaded. *Transparent* is also used in the merge functions to allow foreground transparency. (Default = false)

FileSize - Integer, read only. This is the size of the image in memory when it was first loaded from disk or a binary variable.

OriginalimageType - String, read only. When an image is loaded this property is assigned a string value to describe the format of that image. The possible values are: "bmp", "gif", "jpg", "png", "tif", or an empty string if no file is loaded.

IgnoreInputFileType - Boolean. csImageLite can read an image that has the wrong extension, and this is the default behaviour. Set *IgnoreInputFileType* to false to raise an error if an image format does not match the extension. (Default = true)

OverwriteMode - Integer, must be 0, 1 or 2. Used with the *WriteFile* method to determine what to do with duplicate file names. The default is 0 and this causes files to be saved with the name specified in *WriteFile*, even if a file by that name already exists. 1 will prevent *WriteFile* from saving a file if the file name exists. 2 to will cause the file to be renamed if the name is already used and *~n* is added to the end of the name where *n* is the lowest available number. A typical use of *OverwriteMode* is to set it to 2 when reading in files from our csASPUpload component to prevent deletion of duplicate file names. (Default = 0)

OverwriteChr - String. If *OverwriteMode* is set to 2 and a numbered suffix is added to the file name the value of *OverwriteChr* is used to separate the file name and the suffix number. The default is "~" but sometimes URLs containing this character are blocked for security reasons. (Default = "~")

FileName - String, read only. After saving a file with *WriteFile* the full path and name actually used for the file is stored in this property. This may be different from the path specified if *OverwriteMode* is used.

NewFileSize(*Type*) - Integer, read only. This is the file size that will be produced if the current image is exported in the format specified by *Type*. *Type* must be the string value "jpg", "bmp", "png", "gif" or "tif".

WasCMYK - Boolean, read only. This gets set to true if a JPG or TIF image is read in CMYK format. It indicates that the image has been converted to RGB.

2.3. Properties for Binary Image Export and Streaming

The following property returns the current image as a variant array, which is the format used by the ASP Response.BinaryWrite command. This can be used to stream an image to the browser, as described in more detail in Section 6. This format can also be used to copy an image into a binary database field. Not all programming environments support the variant array data type.

VariantOut(*Type*) - Variant array, read only. The image in the format specified by *Type*, which must be the string value "jpg", "bmp", "png", "gif" or "tif".

Example of sending a JPG to the browser:

```
Response.ContentType = "image/jpeg"
Response.BinaryWrite Image.VariantOut("jpg")
```

This would be used inside a script that is called from inside a <img> tag, as described in Section 6.

Example of saving a GIF into a binary database field:

```
RS("ImageField") = Image.VariantOut("gif")
```

This assumes the binary database field is called "ImageField" and that the recordset is called "RS".

There is a single method for streaming to the browser, taking a string parameter to specify the file type.

StreamToBrowser (*ImgType*) - Streams the current image to a browser where *ImgType* is the string value "jpg", "bmp", "png", "gif" or "tif". This is used instead of Response.BinaryWrite but Response.ContentType must still be set. This uses the ASP Response object so it can only be used in classic ASP scripts.

Example of sending a JPG to the browser:

```
Response.ContentType = "image/jpeg"
Image.StreamToBrowser "jpg"
```

StreamToBrowser should be used for larger files because it buffers the data into smaller blocks than BinaryWrite. Note that not all image types can be displayed in a browser.

The following property is used for import and export .

BMPHandle - Integer. The Windows handle of the image stored in memory. This can be used to exchange images with other COM enabled tools, such as Visual Basic, but it is also an efficient way to pass an image between instances of csImageLite. Reading the property returns the handle to a copy of the image so any modifications done to the exported image do not affect the image in memory.

Example of copying images between instances of csImageLite:

```
Image1.BMPHandle = Image2.BMPHandle
```

This copies the image from an instance called Image2 to the instance called Image1. It does not copy any of the properties, such as *XDPI* or *Transparent*. Use *ReadVariant* with *VariantOut* to copy the image and the associated properties.

2.4. TIFF Compression

TIF files support several different compression types. Not all of these are supported by csImageLite. See below for the compression types supported.

CompressionType - Integer, must be 0, 1 or 2. This specifies the compression type that will be used when a TIF file is saved. 0 is no compression, 1 is Packbits compression and 2 is Group 4 compression, which can only be used on black and white images. If a colour image is saved as Group 4 (CompressionType = 2) it will be converted to black and white if it is not already. (Default = 0)

CompressionType can also take the values 3 and 4 when a TIF is loaded and these indicate that the original compression was Group 3 and LZW respectively. csImageLite does not save TIFs using these compression types and attempting to do so will result in Group 3 compression saved as Group 4 and LZW compression saved as Packbits.

Please note that csImageLite does not support multi-page TIF documents. When a multi-page TIF is opened, only the first page will be loaded with the rest discarded. Our csImageFile component provides support for multi-page TIFs.

2.5. Base 64 Encoding

csImageLite can import and export images using base64 encoding. The methods used are *Base64In* and *Base64Out*.

Base64In *Input* - Loads an image that is a base64 encoded string, *Input*.

Base64Out *ImgType* - Returns a string using base64 encoding. *ImgType* is the image format as a string, e.g. "jpg" or "png".

Base64 was traditionally used for attaching binary data to documents such as emails. It can be used to store binary data in a string field in a database and it can be used to display an image in a browser, as in the following example.

Example:

```
<img src="<%= Image.Base64Out("jpg") %>" />
```

If an instance of csImageLite is called Image the above code will display the loaded image in a browser img tag as a jpeg, and it avoids the need for a separate script to display the image. It is approximately 33% larger in size than the equivalent as a binary stream.

3. Image Resize and Manipulation

3.1. Methods for Image Manipulation

A common type of image manipulation for which csImageLite is used is the resizing of images. Three methods are provided for this, *Resize*, *ResizeFit* and *Scale*.

Resize *Width*, *Height* - The current image will be resized to new pixel dimensions *Width* and *Height*. If either parameter is zero the other will be used to determine the new size and the aspect ratio will be maintained. *Width* and *Height* are integers.

Example:

```
Image.Resize 100, 0
```

This resizes the image to 100 pixels wide while maintaining the aspect ratio. In VBScript there are no brackets around method parameters. There is no equals sign because it is a method call.

ResizeFit *MaxWidth*, *MaxHeight* - If the current image has a width or height greater than *MaxWidth* or *MaxHeight* it will be resized to fit within those dimensions while maintaining aspect ratio. If the image already fits within *MaxWidth* and *MaxHeight* it will be unchanged. *MaxWidth* and *MaxHeight* are integers.

Scale *Factor* - The current image will be scaled by *Factor* percent, where *Factor* is an integer. Images that are scaled always maintain aspect ratio.

Example:

```
Image.Scale 50
```

This scales the current image to 50% of original size. (The width is scaled by 50% and the height is scaled by 50% so the area is reduced to 25% of its original size.)

The following method, *NewImage*, is important because it is used to create a blank image which can be used for drawing or writing onto or as a background with one of the merge functions.

NewImage *Width*, *Height*, *Color* - This loads a blank image into memory, deleting any previously loaded image. It is *Width* x *Height* pixels in size and the colour is determined by the *Color* parameter. *ColorDepth* is set to 24 after calling *NewImage*. *Width* and *Height* are integers and *Color* is a 6 character string representing the "RRGGBB" colour value.

The following methods provide simple image manipulation functions.

Crop *X1*, *Y1*, *X2*, *Y2* - The parts of the rectangle outside the area defined by the opposite corners (*X1*, *Y1*) and (*X2*, *Y2*) will be removed. Note that coordinates are measured across and down from the top left corner of the image. The parameters are integers.

Crop can also be used to increase the area of the image by specifying coordinates that are outside the image. In this case the new area will be the colour of *TransparentColor*.

Rotate *Angle* - Rotates the image by *Angle* degrees anticlockwise. If the angle is not a right angle the new image will be centred on a larger rectangle, the colour of which will be defined by the *TransparentColor* property. If *Resample* is true and if the image has a colour depth of 24 bit the image will be smoothed during rotation. *Angle* is a floating point number.

FlipX - The image is reflected about an axis parallel to the x-axis running through the centre of the image. The top row of pixels becomes the bottom row and the bottom becomes the top.

FlipY - The image is reflected about an axis parallel to the y-axis running through the centre of the image. The left column of pixels becomes the right column and the left becomes the right.

The following methods perform image effects and colour adjustments.

GrayScale - Converts the image to a 256 colour greyscale image.

ConvertToBW - Converts the image to a 2 colour black and white image.

Invert - Converts the image to its negative by changing the colour of each pixel to the opposite colour, i.e., black is changed to white, white to black, etc.

The following functions can convert between OLE_COLOR values and the 6 character strings that csImageLite uses for colours.

OLEColorToStr(*Color*) - This function takes a 4 byte OLE_COLOR value, *Color*, and returns a 6 character string of the form "RRGGBB".

StrToOLEColor(*ColorStr*) - This function takes a 6 character string of the form "RRGGBB" and returns a 4 byte OLE_COLOR value.

3.2. Properties for Image Manipulation

The property listed below controls resampling of resized and rotated images.

Resample - Boolean. When true a resampling filter will be applied during an image resize or rotation. When resampling is used the image will be converted to 24 bit colour.

For image resizing the quality is usually high enough when applied to 24 bit images and *Resample* does not need to be set.

For rotations using an angle that is not a multiple of 90 degrees the image will develop some jagged edges and setting *Resample* can reduce this effect.

3.3. The Merge Methods and Properties

There are 6 methods available for combining the current image with a second image. *MergeFront* and *MergeBack* take the second image from a file stored on disk. *MergeFrontBin* and *MergeBackBin* take the second image as a binary data stream which can come from a database or an upload. *MergeFrontHDC* and *MergeBackHDC* take the second image as a Windows handle which can come from another instance of the component. The "Front" methods make the current image the foreground and the "Back" methods make the current image the background.

The foreground image can be partially transparent by setting the *TransPercent* property. *TransPercent* is a percentage, taking a value between 0 and 100 where 0 is fully opaque and 100 is fully transparent (invisible). A single colour in the foreground can be fully transparent by setting the *Transparent* property to true and setting *TransparentColor* to the required colour. If the foreground image contains alpha transparency, this will be used in producing the composite image, in addition to the transparency options specified by *TransPercent* and *TransparentColor*. If both the foreground and background images contain alpha transparency the default behaviour will produce a composite image with no alpha transparency. To maintain alpha transparency set the *KeepAlpha* property to true.

The foreground image can be made to tile over the background by setting the *Tile* property to true. This ignores the *X* and *Y* coordinates and completely covers the background image. It would often be used with some transparency settings to create a watermark.

If the foreground image is 24 bit and the background image uses a palette the result will only contain the colours of the palette. This can be prevented by using the *MergeBack* method and using

ColorDepth to change the paletted image to 24 bit before merging. If the background image uses a palette, any foreground alpha transparency will be ignored.

If the foreground image specified in *MergeBack* and *MergeBackBin* contains transparency, this will be used during the merge and will overrule the settings of *Transparent* and *TransparentColor*.

MergeFront *Background*, *X*, *Y* - *Background* is the full physical path and filename for another image. The current image in memory is placed onto this background image. The top left corner is *X* and *Y* pixels across and down from the top left of the background image. *Background* is a string, *X* and *Y* are integers.

MergeBack *Foreground*, *X*, *Y* - *Foreground* is the full physical path and filename for another image. This foreground image is placed onto the current image. The top left corner is *X* and *Y* pixels across and down from the top left of the current image. *Foreground* is a string, *X* and *Y* are integers.

MergeFrontBin *BackgroundData*, *ImgType*, *X*, *Y* - This is the same as the *MergeFront* command except that it reads the background image from binary data. *BackgroundData* is a variant array containing another image (see also *ReadVariant*). The current image in memory is placed onto this background image. The top left corner is *X* and *Y* pixels across and down from the top left of the background image. *ImgType* is a string indicating the format of the background image and it must be one of "jpg", "gif", "bmp", "png", "tif". An empty string can be accepted for *ImgType* and it will use the image if it is a supported format. *X* and *Y* are integers.

Example:

```
FileData = RS("ImageField")
Image.MergeFrontBin FileData, "jpg", 25, 50
```

This places the current image onto another JPG image read from a database field. The top left corner of the current image will be at coordinates 25, 50 on the background image.

MergeBackBin *ForegroundData*, *ImgType*, *X*, *Y* - This is the same as the *MergeBack* method except it reads the foreground image from binary data. *ForegroundData* is a variant array containing another image (see also *ReadVariant*). This image is placed on top of the current image. The top left corner is *X* and *Y* pixels across and down from the top left of the current image. *ImgType* is a string indicating the format of the background image and it must be one of "jpg", "gif", "bmp", "png", "tif". An empty string can be accepted for *ImgType* and it will use the image if it is a supported format. *X* and *Y* are integers.

MergeFrontHDC *BackgroundHandle*, *X*, *Y* - This is the same as the *MergeFront* method except that it reads the background image from a Windows bitmap handle. The current image is placed onto this background image. The top left corner is *X* and *Y* pixels across and down from the top left of the background image.

Example:

```
Image1.MergeFrontHDC Image2.BMPHandle, 25, 50
```

This places the image stored in the first instance of csImageLite (Image1) onto an image stored in a second instance of csImageLite (Image2). The top left corner of the first image will be at coordinates 25, 50 on the second image. Note that the resulting image is stored in Image1 and the image in Image2 is unchanged.

MergeBackHDC *ForegroundHandle*, *X*, *Y* - This is the same as the *MergeBack* method except that it reads the foreground image from a Windows bitmap handle. This image is placed on top of the current image. The top left corner is *X* and *Y* pixels across and down from the top left of the current image.

The following properties affect image merging.

TransPercent - Floating point number between 0 and 100. When images are merged this specifies the percentage transparency of the foreground image. By setting this to a high value a watermark is produced. The background image needs to be 24 bit colour for this to be effective.

Tile - Boolean. Used with the merge functions. When true the foreground image is tiled to fill the background and the coordinates are ignored. (Default = false)

4. Adding Text to an Image

Text can be placed on the current image using the *Text* method. Properties are available to control the font, style, size, colour and angle of rotation. Unicode characters can be used if the font supports them. Carriage returns can be placed in the string to allow the text to span multiple lines and there is a choice of justification options for this multi-line text.

Antialiasing is possible by setting the *Antialias* property to true and this enables Windows font smoothing. This requires the text to be above a certain size before antialiasing takes place, for example the Arial font does not antialias until the size is at least 18. Antialiasing should be avoided if the image is to be exported as a GIF or PNG with background transparency because the antialiased pixels are not the same colour as the background and are therefore visible.

Text *X*, *Y*, *TextString* - This places a string of text onto the image. The string is *TextString*, and the top left corner of the text is positioned at the point (*X*, *Y*). The text will break onto the next line if the string contains a carriage return, line feed or CRLF pair (in VBScript the constant vbCRLF can be used). Unicode characters will be displayed if the font specified by *TextFont* supports them. *X* and *Y* are integers, *TextString* is a string.

TextFont - String. The name of the font to be used. It defaults to the system font for the server and this font will be used if *TextFont* specifies an unavailable font. Note that if a font is installed on a remote server using Terminal Services, the server must be rebooted before csImageLite will be able to use the font.

TextColor - String. The colour of the text as a 6 character string. (Default = "000000", black)

TextSize - Integer. The height of the text in pixels. (Default = 16)

TextBG - String. The colour of the text background as a 6 character string. (Default = "FFFFFF", white)

TextOpaque - Boolean. When true the text is drawn on a background the colour of *TextBG*. When false the background is transparent. (Default = true).

TextBold, TextItalic, TextUnderline, TextStrikeout - Boolean values to set text styles. All default to false.

TextAngle - Integer. The angle of rotation of the text, in degrees, measured anticlockwise from the horizontal. Only true type fonts can be drawn rotated. (Default = 0)

TextJustify - Integer, must be 0, 1 or 2. This applies to text spanning multiple lines and determines the justification. Text on a single line is not affected. The *X* parameter of the *Text* method always specifies the left position of the text string, even when text is right justified. 0 - Left, 1 - Centre, 2 - Right. (Default = 0, Left)

Text example:

```
Image.TextSize = 20
Image.TextBold = true
Image.TextFont = "Arial"
Image.TextOpaque = false
Image.Text 100, 50, "Some Text"
```

This sets the size and the font and sets the style to bold before drawing some text at coordinates (100, 50). *TextOpaque* is set to false so that the text background is transparent. Note that the text properties require an equals sign to assign a value but *Text* is a method and does not use an equals sign.

Antialias - Boolean. When true, text will be drawn antialiased. (Default = false)

The font must be large enough or small enough to support it and the image must be 24 bit colour depth. If necessary the *ColorDepth* property should be set to 24 before drawing the text. If the computer running csImageLite has the display properties set to smooth screen fonts this will overrule the *Antialias* property and text will be antialiased even when the property is false.

The *TextHeight* and *TextWidth* functions return the height and width of a text string. This can be useful when positioning an item of text because the *X* and *Y* parameters of the *Text* method always set the top left corner of the text.

TextHeight(*Text*) - Integer return value. This returns the height in pixels if the string *Text* is to be drawn using the current font and size settings.

TextWidth(*Text*) - Integer return value. This returns the width in pixels if the string *Text* is to be drawn using the current font and size settings.

Text can be made to span multiple lines by adding carriage returns to the string in the *Text* method. Alternatively, text can be made to fit into a rectangle.

TextWrap - Boolean. When true the text will wrap onto multiple lines depending on the length and the values of the *TextRectX* and *TextRectY* properties. The line will only break at a space between characters and it will be justified as specified by the TextJustify property. (false)

TextRectX - Integer. When *TextWrap* is true *TextRectX* is the width of the rectangle containing the text, in pixels. If zero, the text will extend to the right of the image before wrapping to the next line. (Default = 0)

TextRectY - Integer. When *TextWrap* is true *TextRectY* is the height of the rectangle containing the text in pixels. If zero, the text will fill as many lines as required up to the height of the image. (Default = 0)

Text can be made partially transparent by setting the *TextTransPercent* property before drawing the text.

TextTransPercent - Integer in the range 0 to 100. This specifies an amount of transparency for text where 0 is completely opaque and 100 is invisible. Setting it to a high value makes the text appear as a watermark. The image must be 24 bit colour for this property to have any effect. (Default = 0)

5. Alpha Transparency and PNG Images

Some PNG images contain an "alpha channel" which describes a level of transparency for each individual pixel. csImageLite can extract this when a PNG image is loaded. If the image is saved again in PNG format, this alpha channel will be saved as part of the image.

The alpha channel will be preserved if the image is resized, cropped, rotated or flipped. Other edits will cause the alpha channel to be discarded unless the *KeepAlpha* property is first set to true. Before discarding any alpha channel the image will be merged with the background.

A PNG image can specify a background colour. This is to be shown behind transparent pixels if the viewer is unable to show whatever is behind the image. csImageLite will merge a PNG containing an alpha channel with this background when the image is exported to another format, or when the alpha channel is removed due to editing as described above. If no background colour is specified, white will be used by default.

The following properties are related to alpha transparency and background colours.

HasAlpha - Boolean, read only. This is set to true if the image currently loaded contains an alpha channel. Use *ClearAlpha* to remove an alpha channel.

KeepAlpha - Boolean. When true, edits applied to the image will not remove the alpha channel. This property is also used to maintain the background alpha channel when two images with alpha channels are merged. (Default = false)

BackgroundColor - String. The colour of the background that will be shown behind transparent pixels when an image containing an alpha channel is exported. (Default = "FFFFFF", white)

HasBackground - Boolean. This determines whether the current image contains a background colour which will be stored when the image is exported in PNG format. (Default = false)

ExtraAlphaValue - Integer in the range 0 - 255. When an image containing an alpha channel is rotated by an angle that is not a multiple of 90 degrees, or cropped to coordinates that make the image bigger, the new area created will have an alpha value specified by this property. 0 is fully transparent and 255 is fully opaque. (Default = 0)

AlphaPixel (*X*, *Y*) - Integer in the range 0 – 255. This property allows the alpha value of a specified pixel to be read or written.

The following methods are related to alpha transparency and background colours.

ClearAlpha - This removes any alpha channel from the current image without merging it with the background.

NewAlpha *Value* - This creates an alpha channel for the current image and sets the value for each pixel to *Value*, which is an integer between 0 and 255 where 0 is fully transparent and 255 is fully opaque. This will convert the image to 24 bit colour if it is not already.

CombineAlphaWithBG - This combines the image with the background colour, if it contains an alpha channel. The alpha channel is then removed.

AlphaText (*X*, *Y*, *TextOut*) - This draws antialiased text on an image with an alpha channel. *X* and *Y* are the coordinates of the text and *TextOut* is the text string. It uses the same properties for controlling the font, size and colour as the *Text* command in Section 5. An exception will be raised if there is no alpha channel in the current image.

PNG images containing a colour palette (colour images that are 8 bit or less) can specify a variable amount of transparency for each palette entry. This information is also preserved when loading and

saving, but it is removed when the image is edited, except for resize, crop, rotate and flip operations. These images will not be merged with a background when exported or edited, but the transparency is used in merge functions..

6. Streaming an Image to the Browser in ASP

An active server page will return HTML output by default. An HTML page is formatted text which can include spaces to display images. The images themselves are not part of the HTML but are separate files, the location of which is specified inside the <img> tag. An ASP page can be an image if the ContentType is set to "image/gif", "image/jpeg" or "image/png" and the binary data of the image is output using the Response.BinaryWrite command. The ASP image is generated by placing the path to the script inside the <img> tag.

For example, this page will display the image produced by "resize.asp":

```
<html>
<head><title>HTML page containing an image</title></head>
<body>
<img src="resize.asp">
</body>
</html>
```

Resize.asp may look like this:

```
<%@ language=vbscript %>
<%
      Response.Expires = 0
      Response.Buffer = true
      Response.Clear
      Set Image = Server.CreateObject("csImageLite.ImageFunctions")
      Image.ReadFile "c:\images\bigimage.jpg"
      Image.Scale 25
      Response.ContentType = "image/jpeg"
      Response.BinaryWrite Image.VariantOut("jpg")
%>
```

When the first HTML page is loaded it looks for the image at "resize.asp", runs the script and is sent a stream of binary data in JPG format, so the browser displays the image. It is not possible to place the BinaryWrite command inside the <img> tag to produce the image, it must be in a separate file.

If the line setting the ContentType is missing, the image should still display in modern web browsers but it might not in some older browsers. It is important to specify the correct ContentType to maintain compatibility.

It is useful to know that parameters can be passed to the ASP image script using the URL string and this can be read using Request.QueryString and used somewhere in the script.

Example:

```
<img src="resize.asp?NewSize=150">
```

This can be used by resize.asp:

```
Image.Resize Request.QueryString("NewSize"), 0
```

The *StreamToBrowser* method can be used instead of Response.BinaryWrite. If large images are to be displayed in the browser, *StreamToBrowser* may be preferreable because it puts less data into the response buffer at any time.

Example:

```
Response.ContentType = "image/jpeg"
Image.StreamToBrowser "jpg"
```

An alternative to streaming the image from a script specified in the <img> tag is to use Base64 encoding and the *Base64Out* command. This produces a larger output file size but removes the need for a separate script to stream the image.

For details of using Base64 output, see Section 2.5.

7. Language Specific Issues

All the examples that are shown so far in these instructions use ASP and VBScript. The csImageLite component is a COM object and can be used in most COM enabled environments running on a Windows platform. We cannot cover all the possible development environments here so instead we concentrate on ASP using VBScript and in this section we show the syntax for Javascript, Cold Fusion and ASP.NET.

7.1. Active Server Pages

ASP and VBScript is already covered in these instructions but the following points are worth noting.

Calls to methods (functions) do not use brackets, although their use does not generate an error if there is only one parameter. For example the correct syntax for *Resize* is:

```
Image.Resize 200, 0
```

These instructions show methods without brackets surrounding the parameters, unless that method has a return value.

If the method has a return value that is assigned to another variable, the brackets are required. For example:

```
ImageSize = Image.NewFileSize("jpg")
```

Assigning a property value requires an equals sign. Missing the equals sign results in the unhelpful error "Object doesn't support this property or method". The correct syntax is:

```
Image.JpegQuality = 70
```

A lot of ASP syntax errors are caused by adding brackets when they are not required, or missing them when they are required, missing equals signs when assigning a property, or accidentally including an equals sign in a method call.

7.1.1. ASP with Javascript

We don't provide any examples of using ASP with other scripting languages, other than VBScript. We will mention the following about using Javascript with ASP.

Brackets are needed around function parameters.

The backslash character is used as an escape character in Javascript and two should be used together when a backslash is needed:

```
Image.WriteFile("C:\\output\\newimage.gif");
```

7.2. Cold Fusion

In Cold Fusion, a COM object is created using the <cfobject> tag:

<cfobject action="create" name="Image" class="csImageLite.ImageFunctions">

Each command must be placed inside a `<cfset>` tag and all method parameters must be enclosed by brackets:

```
<cfset Image.ReadFile("c:\images\bigimage.jpg")>
<cfset Image.Scale(20)>
<cfset Image.WriteFile("c:\images\smallimage.jpg")>
```

Alternatively, the commands can be put inside a <cfscript> block:

```
<cfscript>
Image.ReadFile("c:\images\bigimage.jpg");
Image.Scale(20);
Image.WriteFile("c:\images\smallimage.jpg");
</cfscript>
```

Cold Fusion version 5 does not support variant arrays and so commands such as *VariantOut* and *ReadVariant* cannot be used. Without these commands, images cannot be streamed directly to the browser so any dynamically produced image must be saved to a temporary file first and then displayed using a <cfcontent> tag.

It is possible to stream images using Cold Fusion without saving to a temporary file first. The following commands will stream an image in jpeg format assuming an image is loaded into a csImageLite object called "Image".

```
<cfscript>
Context = GetPageContext();
Context.SetFlushOutput(false);
Response = Context.GetResponse().GetResponse();
Out = Response.GetOutputStream();
Response.SetContentType("image/jpeg");
Out.Write(Image.VariantOut("jpg"));
Out.Flush();
Response.Reset();
Out.Close();
</cfscript>
```

Initially, 64 bit versions of Cold Fusion had no support for COM objects. COM support was added with Cold Fusion 10 Update 11.

7.3. ASP.NET

csImageLite can be used with ASP.NET. The component must be registered on the server as described earlier and it can be called using Server.CreateObject.

For example:

```
Dim Image = Server.CreateObject("csImageLite.ImageFunctions")
```

The object is created using Dim instead of Set. For the trial version the class name is "csImageLiteTrial.ImageFunctions".

The image cannot be streamed to the browser in a single line because of incompatibilities between ActiveX and .NET, but there is a workaround which we describe in the VB.NET example below.

```
<%@ Page language="vb" debug="true" %>
<%
Response.Expires = 0
Response.Buffer = true
Response.Clear

Dim Image = Server.CreateObject("csImageLite.ImageFunctions")
Image.NewImage(150, 50, "00ff00")
Image.TextOpaque = false
Image.TextSize = 22
Image.Text(5, 10, "Sample image")
Dim OutArray As Array = Image.VariantOut("gif")
```

```
Dim ByteArray(OutArray.Length - 1) As Byte
Array.Copy(OutArray, ByteArray, OutArray.Length)
Response.ContentType = "image/gif"
Response.BinaryWrite(ByteArray)
%>
```

This creates a new image and draws some text onto it. In order to stream it to the browser the output from the *VariantOut* property needs to be passed to an array of bytes so that it can be sent out through BinaryWrite. Brackets are used to enclose function parameters.

csImageLite can be used in C# scripts, but early binding must be used as described in the next section. The code needed to stream the image is shown below:

```
Array OutArray = (Array)(Image.VariantOut("gif"));
Byte[] ByteArray = new Byte[OutArray.Length];
Array.Copy(OutArray, ByteArray, OutArray.Length);
Response.ContentType = "image/gif";
Response.BinaryWrite(ByteArray);
```

7.3.1. Early Binding

The previous example used late binding, which is the easier way of calling an ASP component in ASP.NET. It is more efficient to use early binding, but this requires the creation of a .NET Framework Interop Assembly using the TLBIMP tool, supplied with the Framework. This assembly is a DLL which acts as a wrapper for the ASP component.

After registering the component, run TLBIMP.exe from the command prompt or from the Run box in the Start Menu. The syntax is:

```
TLBIMP ComponentName.dll /out:NewName.dll
```

Full paths are required for both DLLs. The new DLL needs to be put in the website's BIN directory. The script that calls the component must import the Interop Assembly as a NameSpace. The component instance is created using the following VB.NET syntax:

```
Dim ObjName As New ClassNameClass()
```

ObjName is the name of the object instance and *ClassName* is the name of the class in the ASP component, which is ImageFunctions in csImageLite.

The script that uses the component must import the Interop Assembly as a Namespace. If the Interop Assembly is called "csimagelitenet.dll" the following line imports it:

```
<%@ Import Namespace = "csimagelitenet" %>
```

In the previous example the only other change required is to replace the Server.CreateObject line with:

```
Dim Image As New ManageClass()
```

8. Supported Image Formats

csImageLite supports the reading and writing of files in the most popular formats commonly found on the web. These formats are briefly summarised here.

8.1. BMP (Bitmap)

BMP is the standard graphics format for Windows systems. The image is stored uncompressed and can be in either 24 bit or a paletted format. The palette can be either 256 colour (8 bit), 16 colour (4 bit) or monochrome (1 bit).

Bitmaps are usually larger file sizes than other images because they are uncompressed but they are often faster to import and export because there is no compression to calculate.

8.2. JPG / JPEG / JPE (Joint Photographic Experts Group)

The JPEG file format is probably the most common method of saving full colour images, typically photographs, when a high level of compression is required. It uses a "lossy" compression method, which means that the image read from a saved file will not be identical to the original image before it was saved.

Repeated loading and saving of a JPEG file will result in deterioration of the quality, and should be avoided, but at the levels of compression typically used for saving photographs, the loss of quality is hardly noticeable when the file is saved once from its original. The type of compression used by JPEGs does not produce good quality images that contain sharp edges and text as these edges often become blurred on saving, even using a low compression level. The *JpegQuality* property controls the amount of compression used when a JPEG is saved.

JPEG compression / quality is not a universal value and different software packages use different scales. It cannot be read directly from an image. If *JpegQuality* is set to a higher value than that originally used to save the image, the new image will have a larger file size without gaining any quality. Once quality has been removed by applying compression it cannot be recovered.

JPEG files stored in CMYK colour space can be read by csImageLite and are automatically converted to RGB. The *WasCMYK* property is set to true to indicate a conversion has taken place. Saving to CMYK is not supported.

JPEG images are usually stored as 24 bit RGB images but they can also be stored as 8 bit greyscale. See the *JpegGrayScale* property for details.

csImageLite provides an option for opening JPEG files at a higher speed while reducing image quality. This can be useful if speed is important, for example when creating thumbnails dynamically. Set the *JpegHigherSpeed* property to true before loading the image for this to take effect.

8.3. GIF (Graphics Interchange Format)

GIF format saves files using 256 colours or less. It can compress images with large areas of a single colour very efficiently and is commonly used in web pages for backgrounds, logos, etc. The compression is "lossless". GIFs use the LZW compression method.

GIF images support the use of a transparent colour. See the *Transparent* and *TransparentColor* properties for details.

GIF files can contain multiple images (or frames) and these are usually used to create animations for use in web pages. Multi frame GIFs are not supported by csImageLite, but they are supported by csImageFile and csASPGif.

8.4. PNG (Portable Network Graphics)

The PNG format provides an efficient lossless compression of image data and it uses the same compression method as zip files. It provides a wide range of colour depths, but these are not all supported for writing by csImageLite, although they can be read. Supported colour depths are 24 bit, 8 bit, 4 bit and 1 bit. Paletted images support single colour transparency in the same way as GIF files or they can specify a variable amount of transparency for each palette entry.

The PNG format also allows for an alpha channel where a transparency amount is specified for each pixel. csImageLite supports this, and the alpha channel can be used when images are merged. An image with an alpha channel can be resized, rotated, cropped or flipped while preserving the transparency. Any other editing will clear the alpha channel unless instructed otherwise by setting the *KeepAlpha* property. If the image is exported to another format, the pixels are merged with the background colour of the PNG image and the alpha channel is lost.

8.5. TIF / TIFF (Tagged Image File Format)

The TIF file format allows images to be stored in many different ways. The baseline TIF specification is fully supported by csImageLite and this allows for images to be stored as 24 bit RGB, paletted (8 bit or 4 bit), greyscale or black and white. The image can be uncompressed or stored using the Packbits compression method which is a lossless run-length compression.

In addition to the baseline specification, some extensions to the TIF format are supported as described below.

Images stored using the CCITT Group 3 or Group 4 fax compression schemes can be read. Files can be written using Group 4 compression. These compression schemes apply to black and white images only and are very efficient.

Files stored in CMYK colour space can be read and are automatically converted to RGB. The *WasCMYK* property is set to true to indicate a conversion has taken place. Saving to CMYK is not supported.

Files stored using LZW compression can be read, but this compression method is not available for writing.

TIF files can contain multiple pages but csImageLite does not support them. They are supported by csImageFile.

There are further extensions to the TIF format that are not supported by csImagLite or csImageFile, including JPEG compression, YcbCr and LAB.

9. Revision History

The current version of csImageLite is 1.0, and it is a new release. Where functionality has been copied from csImageFile, it was taken from version 8.4.

10. Comparison with csImageFile

csImageLite is based on our more comprehensive image component, csImageFile, but it has reduced functionality at a lower price. A brief summary of the differences between the two is shown below but refer to the instructions for full details of what is supported.

10.1. What Functionality is Missing?

csImageLite supports fewer image formats with no support for PSD, PCX, WBMP and no export to PDF.

There is no multipage or multiframe support for GIF or TIF formats.

Drawing and editing functions are significantly reduced with no line or shape drawing, no colour fills, or brightness and similar adjustments.

There is no functionality to edit a colour palette (colour table).

Support for alpha transparency in PNG images does not include the ability to edit the alpha channel.

There is no support for meta data, which is a major feature of csImageFile.

ICC colour profiles are not preserved.

Images cannot be loaded directly from a remote URL.

10.2. What is Included?

There is support for JPG, PNG, BMP and single frame GIF and TIF images.

Image resize.

Merging images and watermarking.

Adding text to images.

11. Other Products From Chestysoft

Visit the Chestysoft web site for details of other COM objects.

ActiveX Controls

csXImage	-	An ActiveX control to display, edit and scan images.
csXGraph	-	An ActiveX control to draw pie charts, bar charts and line graphs.

ASP Components

csImageFile	-	Image editing component with more functionality than csImageLite
csDrawGraph	-	Draw pie charts, bar charts and line graphs in ASP.
csASPGif	-	Create and edit animated GIFs.
csASPUpload	-	Process file uploads through a browser.
csASPZipFile	-	Create zip files and control binary file downloads.
csFileDownload	-	Control file downloads with an ASP script.
csFTPQuick	-	ASP component to transfer files using FTP.

Web Hosting

We can offer ASP enabled web hosting with our components installed. Click for more details.

12. Alphabetical List of Commands