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

csASPGif - Version 2.0
COM Object for Creation and Editing of Multi-Frame GIF Images

This is a COM object that can create and edit multi frame GIF images (animated GIFs). A comprehensive range of functions is available to load and save entire images or individual frames, as well as for drawing text and shapes, setting animation properties and manipulating colour maps.

A free, fully functional trial version of csASPGif is available. If you are reading this instruction manual for the first time, it is likely that you have just downloaded the trial version. The trial version has a built in expiry date that causes it 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 control is suitable for your application before considering whether to license the full version.

Using these Instructions

These instructions are divided into a number of sections covering different types of functions available in csASPGif. 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.

There is a section explaining some of the essential details of the GIF format and it is important to understand the concepts of the Global Colour Table and Local Colour Tables before using csASPGif to draw and edit GIF images.

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

Chestysoft, July 2010
www.chestysoft.com

TABLE OF CONTENTS

1. Import and Export of GIFs and Individual Frames

Images can be read into the control and exported from the control in several different ways. The most common method is to read files from disk and write files to disk. Images can be read from a variant array data type as exported by a database field or an upload component. Images can be exported in this binary format and streamed to a web browser for dynamic display. It is also possible to read an image from a remote URL via the internet. All these import and export methods use the GIF format. An additional import/export method uses the handle to a bitmap but this can only be used with individual frames.

Reading a complete GIF overwrites the existing contents of the component. A frame can only be read to replace an existing frame and the *AddFrame* function must be used if there is no existing frame at the required index.

1.1. Reading and Writing Files From/To Disk

Complete GIF images are read into the component using *ReadFile* and saved to disk using *WriteFile*. Individual frames are read using *ReadFrameFromFile* and saved using *WriteFrameToFile*.

Appropriate permissions must be set for reading and writing files. In a web application the Internet Guest User account must have Read permission to open a file, Write permission to save and Modify to overwrite an existing file. The csASPGif component must be added to a COM+ Application in Component Services before it can be used to access files across a network.

ReadFile *FileName* - Reads a GIF image from disk. *FileName* must be a complete physical path to the file, including the file extension.

Example:

```
Gif.ReadFile "C:\images\test.gif"
```

In ASP a virtual path can be mapped to a physical path using Server.MapPath:

```
Gif.ReadFile Server.MapPath(".") & "\test.gif"
```

WriteFile *FileName* - Saves the currently loaded GIF file to disk. *FileName* must be a complete physical path to the file, including the file extension.

ReadFrameFromFile *Filename, Index* - Reads a GIF into the frame specified by *Index*. *FileName* must be a complete physical path to the file. *Index* is an integer where 0 is the first frame. If *Index* specifies a frame that does not yet exist in the current image an error will be generated. If the source image contains multiple frames only the first is used.

Example of creating a GIF and reading an image into the first frame:

```
Set Gif = Server.CreateObject("csASPGif.Gif")
Gif.AddFrame
Gif.ReadFrameFromFile "C:\images\test.gif", 0
```

WriteFrameToFile *FileName, Index* - Saves the frame specified by *Index* to the file name specified by *FileName*, which is the full physical path including the extension.

1.2. Reading and Writing Files From/To a Variable

Complete GIF images can be read from a variant array variable using *ReadStream* and single frames can be read using *ReadFrameFromStream*. Complete GIFs can be exported as a variant array using *GIFData* and single frames using *GIFFrameData*.

Not all languages support variant arrays. ASP uses them and *GIFData* can be used with Response.BinaryWrite to display an image in the browser. Cold Fusion does not support these functions.

ReadStream *Data* - This reads a GIF into the component from a variant array variable, *Data*. This could be a VBScript variable, a binary database filed or from our csASPUpload component.

To read a file directly from csASPUpload using ASP:

```
GifObj.ReadStream UploadObj.FileData(0)
```

This will read a GIF into csASPGif from csASPUpload if the file is the first or only file in the uploaded array. GifObj is the name of the instance of csASPGif and UploadObj is the name of the instance of csASPUpload.

To read data from a database field in ASP the data must first be passed to a temporary variable:

```
TempData = RSet("GIFImage")
GifObj.ReadStream TempData
```

This would read in a GIF file from a binary field in a database, assuming the field name is "GIFImage" and the recordset RSet has been opened. The temporary variable is needed to convert the data to the correct format.

ReadFrameFromStream *Data, Index* - This reads a GIF frame from the variant array variable *Data* into the frame specified by *Index*. If *Index* specifies a frame that does not yet exist in the current image an error will be generated. If the source image contains multiple frames only the first is used.

GIFData - The complete GIF as a variant array.

To send the GIF to a browser an ASP script would use the following, without any other output appearing in the script:

```
Response.ContentType = "Image/gif"
Response.BinaryWrite GIFObj.GIFData
```

GIFFrameData (*Index*) - The frame specified by *Index* as a variant array. The first frame has an index of zero.

Example of sending the first frame of a GIF to the browser:

```
Response.ContentType = "Image/gif"
Response.BinaryWrite GIFObj.GIFFrameData(0)
```

1.3. Reading GIF Images from a Remote URL

GIF images can be read from a remote URL if the calling application has an internet connection. *ReadURL* reads a complete GIF and *ReadFrameFromURL* reads a GIF into a specified frame. The properties *URLUsername* and *URLPassword* can be set if authentication is required. The *HTTPUserAgent* property can be set to specify a user agent in the request header. It is possible that a

firewall would need special configuration to allow the outgoing connection and if a proxy is used the component needs to be added to a COM+ Application in Component Services.

ReadURL *URL* - Loads the GIF image at address *URL* into the component. *URL* must include the "http://" or "https://" prefix. Remember to use forward slashes in the URL not backslashes.

Example:

```
Gif.ReadURL "http://www.chestysoft.com/images/ch1o.gif"
```

ReadFrameFromURL *URL, Index* - Loads the GIF image at address *URL* into the frame specified by *Index*. If the source image contains multiple frames, only the first will be loaded. *Index* must specify an existing frame in the current image. Use *AddFrame* to create a new empty frame.

URLUsername - String. Username to be passed with *ReadURL* or *ReadFrameFromURL*.

URLPassword - String. Password to be passed with *ReadURL* or *ReadFrameFromURL*.

HTTPUserAgent - String. User agent field to be passed with *ReadURL* or *ReadFrameFromURL*. By default this is null and the user agent is not specified.

HTTPTimeout - Integer. Number of seconds before ReadURL or ReadFrameFromURL will time out due to inactivity. A zero value is an indefinite time, and this is the default.

1.4. Exchanging GIF Frames with Other Controls

The *FrameHandle* property provides a way of transferring an image to another control, or receiving an image that is already held in memory.

FrameHandle (*Index*) - Returns a Windows handle to a copy of the bitmap image currently in the frame specified by *Index*. Setting this property copies the image referenced by the new handle value into the frame specified by *Index*. Any image input to a frame will be converted to the same colour depth as the currently loaded GIF image. This is the only method of loading an image that is not in GIF format.

This property can be used, for example, to copy an image between two instances of the component and this would be a way of swapping frames between GIFs.

```
Gif2.FrameHandle(0) = Gif1.FrameHandle(1)
```

This would copy the second frame from the object Gif1 to the first frame of the object Gif2.

Another example is for copying an image to csASPGif from a VB PictureBox control:

```
GifObj.FrameHandle(0) = Picture1.Picture.Handle
```

Finally, here is an example of copying a JPG image using our csImageFile component:

```
Set Image = Server.CreateObject("csImageFile.Manage")
Image.ReadFile "C:\images\test.jpg"
Set Gif = Server.CreateObject("csASPGif.Gif")
Gif.AddFrame
Gif.FrameHandle(0) = Image.BMPHandle
```

The colour depth for csASPGif is 8 bit, unless changed by setting the *ColorDepth* property. The image will be converted to 256 colours when it is imported and a Local Colour Table will be created for the frame. The *AddFrame* command must be called to create the empty frame before an image can be loaded into it.

2. Colours and Colour Tables

This section describes the methods and properties used for working with the Global Colour Table and Local Colour Tables. For an overview on colour tables, read the section on the GIF format.

2.1. Using Strings for Colours

All the colours are entered as hexadecimal strings as used by HTML, so red is "FF0000", blue is "0000FF" and white is "FFFFFF". If a conversion is needed between this string format and the numerical OLE_COLOR values used by other applications, use *OLEColorToStr* and *StrToOLEColor*.

OLEColorToStr (*Color*) - Returns the 6 character HTML style hexadecimal colour string where *Color* is the OLE_COLOR value.

StrToOLEColor (*ColorString*) - Returns the numeric OLE_COLOR value where *ColorString* is the 6 character HTML style colour.

2.2. Colour Tables

Each GIF has a *ColorDepth* which is the number of pixels per byte. This is a value of 1, 4 or 8 and it applies to all the frames in the GIF. By default a newly created object has the *ColorDepth* set to 8.

A GIF may have a Global Colour Table (GCT) which stores up to 256 RGB colour values. The GCT can be used by any or all of the frames. Each frame can either use the GCT or it can have its own Local Colour Table (LCT). The property *HasGCT* applies to the whole GIF and determines whether a GCT exists. *HasLCT* applies to each frame and sets whether the frame uses the LCT or the GCT.

ColorDepth - Integer value, 1, 4 or 8. The number of pixels per byte in the image. Attempting to set the *ColorDepth* to an invalid value raises an error. Default value is 8.

ColorTableSize - Integer, read only. The number of colours in the GIF colour tables. This is dependent on *ColorDepth* and is always 2 ^ (*ColorDepth*).

HasGCT - Boolean. Set to true when the GIF uses a Global Colour Table. (Default = false)

The value of *HasGCT* is applied to new frames added with *AddFrame* to decide whether they use the GCT or have a LCT created.

HasLCT (*Index*) - Boolean. When true the frame *Index* uses a Local Colour Table, when false, the frame uses the Global Colour Table.

HasLocalColorTables - Boolean, read only. This will return true if at least one frame contains a local colour table.

Individual colour table entries can be read or set and entire colour tables can be copied to each other. When a colour is specified for drawing or filling it will be added to the appropriate colour table, if it is not already present and if there is an unused entry.

GCTEntry (*ColorIndex*) - String. The hex string colour value of the GTC entry at *ColorIndex* where *ColorIndex* is an integer between 0 and *ColorTableSize* - 1.

LCTEntry (*Frame*, *ColorIndex*) - String. The hex string colour value of the LCT entry in *Frame* with index *ColorIndex*. *Frame* and *ColorIndex* are integers.

CopyGCTToLCT (*Index*) - Copies the Global Colour Table to the Local Colour Table for the frame specified by *Index*. The colours in the frame are mapped to the nearest available colours in the new table.

CopyLCTToGCT (*Index*) - Makes the Local Colour Table from the frame specified by *Index* into the Global Colour Table. Any frames that use the GCT will have their colours mapped to the nearest available colours in the new GCT.

CopyLCTToLCT (*ToIndex, FromIndex*) - Copies the Local Colour Table used by frame *FromIndex* to the Local Colour Table of the frame *ToIndex*. The colours in the frame are mapped to the nearest available colours in the new table.

The index within the colour table can be found for a given colour using *ColourIndex*. The number of pixels in a frame that are a particular colour can be found with *ColourCount* and the first entry in a colour table that is unused can be found with *FindUnusedColor*.

2.3. Colour Functions

ColorIndex (*Color, Frame*) - Returns the index of *Color* in the colour table used by the frame specified. *Color* is the colour as a 6 character hexadecimal string and *Frame* is the zero based frame index. The return value is an integer and is -1 if *Color* is not present in the colour table.

ColorCount (*ColorIndex, FrameIndex*) - Returns the number of pixels that are the colour of *ColorIndex* in the frame specified by *FrameIndex*. *ColorIndex* is the index in the colour table used by the frame.

FindUnusedColor (*FrameIndex*) - Returns the index within the colour table of the first entry that is not used by any pixels. *FrameIndex* is the index of the frame. The return value is an integer and is -1 if all the colour table entries are used by pixels.

A multiframe GIF image can be made to use less memory by removing the Local Colour Tables and using a Global Colour Table for all the frames. The *OptimizeColorTables* function will do this. If the total number of different colours before optimisation are greater than the size of the GCT some approximation must be used. The colours used for transparency will also be changed during the optimisation.

OptimizeColorTables - Creates a Global Colour Table from all the frames and sets *HasGCT* to false for all the frames. If transparency is used in any frames in the image a single colour will be selected as the transparent colour to be used throughout. Where possible, this will be an unused colour, otherwise the least used colour will be selected.

When a new frame is created the value of its colour table will depend on the value of *DefaultPalette*, the colour depth of the GIF and whether it is the first frame.

DefaultPalette - Integer value, 0, 1, 2 or 3. This controls what predefined colour table entries are used when a frame is created using *AddFrame*, and it must be set before calling *AddFrame*. It will apply when the first frame is created if *HasGCT* is true and for every frame when *HasLCT* is true for that frame. The values have the following meaning:

0 - The colour table will be empty with each entry taking the value zero or black. (Default)

1 - An 8 bit image will be given the web safe palette of 216 colours with the remaining entries black. A 4 bit image will take the standard 16 colour Windows palette and a 1 bit image will be black and white.

2 - Greyscale. The graduation of the colours will vary depending on the colour depth.

3 - Unchanged. In most cases this will be the same as 0 above except when used after modifying the Global Colour Table. With this setting the existing GCT entries will remain.

3. Drawing Shapes and Adding Text

There are a number of functions available for drawing lines, shapes and text.

3.1. Pen and Brush Properties

The colour of outlines and text is determined by *PenColor*. The line thickness and style by *PenThickness* and *PenStyle*. The background and filled colours are determined by *BrushColor*. The filled styles are set by *BrushStyle*.

PenColor - String hex colour value. The colour of lines or text that are drawn using the functions described in this section. (Default = "000000", black).

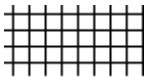
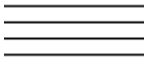
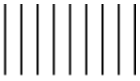
PenThickness - Integer. The thickness in pixels of drawn lines and shapes. This property does not apply to text. (Default = 1).

PenStyle - Integer value 0 to 5. The pattern of drawn lines and shape outlines. Only *PenStyle* = 0 can be used if the *PenThickness* is greater than 1. (Default = 0, solid).

0, Solid	————	3, DashDot	- - — - —
1, Dash	— — —	4, DashDotDot	- - - — - - —
2, Dot	- - - - - - - - -	5, Clear	

BrushColor - String hex colour value. The colour of filled areas. This is not used when *BrushStyle* is clear. Default = "FFFFFF", white.

BrushStyle - Integer value 0 to 7. The pattern of filled areas. (Default = 1, clear).

0, Solid		4, Cross	
1, Clear		5, DiagCross	
2, BDiagonal		6, Horizontal	
3, FDiagonal		7, Vertical	

The colours used by *PenColor* or *BrushColor* will be added to the active colour table, if there is a spare table entry and if they are not already present, otherwise the nearest available colour will be used instead.

3.2. Clearing the Frame

An empty frame is added to the GIF using *AddFrame* but this does not give the frame an area or any other features. An image must be either loaded into the frame or *ClearImage* must be used to give the frame a height and width and a background colour before anything can be drawn on it.

ClearImage *Width, Height, Index* - This makes the image in the frame defined by *Index* into a blank rectangle with dimensions *Width* x *Height*, the colour of *BrushColor*.

3.3. Shapes, Lines and Pixels

The following functions are used for drawing.

DrawArc *X1, Y1, X2, Y2, X3, Y3, X4, Y4, Index* - Draws an elliptically curved line the colour of *PenColor* on the frame specified by *Index*. The arc follows the perimeter of the ellipse bounded by (*X1, Y1*) and (*X2, Y2*), and moves anticlockwise from the start point to the end point. The start point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X3, Y3*). The end point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X4, Y4*).

DrawChord *X1, Y1, X2, Y2, X3, Y3, X4, Y4, Index* - Draws a shape bounded by an arc and a line that joins the endpoints of the arc, on the frame specified by *Index*. The arc is an elliptically curved line that follows the perimeter of the ellipse bounded by (*X1, Y1*) and (*X2, Y2*), and moves anticlockwise from the start point to the end point. The start point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X3, Y3*). The end point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X4, Y4*). The outline is drawn the colour of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*.

DrawEllipse *X1, Y1, X2, Y2, Index* - Draws a circle or ellipse that fits into the rectangle defined by the top left corner (*X1, Y1*) and the bottom right corner (*X2, Y2*), on the frame specified by *Index*. The outline is drawn the colour of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*.

DrawLine *X1, Y1, X2, Y2, Index* - Draws a draws a line from point (X1, Y1) up to but not including the point (X2, Y2). The style, colour and width of the line are determined by the properties *PenStyle*, *PenColor* and *PenThickness*. The line is drawn on the frame specified by *Index*.

DrawPie *X1, Y1, X2, Y2, X3, Y3, X4, Y4, Index* - Draws a pie shaped wedge defined by an arc of an ellipse and lines joining the ends of the arc with the centre of the ellipse. The arc follows the perimeter of the ellipse bounded by (*X1, Y1*) and (*X2, Y2*), and moves anticlockwise from the start point to the end point. The start point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X3, Y3*). The end point is defined by the intersection of the ellipse with a line drawn from the centre to the point (*X4, Y4*). The outline is drawn in the color of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*. It is drawn on the frame specified by *Index*.

DrawRectangle *X1, Y1, X2, Y2, Index* - Draws a rectangle defined by the top left corner (*X1, Y1*) and the bottom right corner (*X2, Y2*). The outline is drawn in the color of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*. It is drawn on the frame specified by *Index*.

DrawRoundRect *X1, Y1, X2, Y2, X3, Y3, Index* - Draws a rectangle defined by the top left corner (*X1, Y1*) and the bottom right corner (*X2, Y2*). The corners will be rounded with a curve matching an ellipse with width *X3* and height *Y3*. The outline is drawn in the color of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*. It is drawn on the frame specified by *Index*.

A polygon can be drawn with a variable number of vertices. The points are first specified by repeated calls to the *PointAdd* function and the polygon is drawn using *DrawPolygon*.

PointAdd *X, Y* - Adds a point to be used as a vertex by the DrawPolygon function.

DrawPolygon *Index* - Draws a polygon on the frame defined by *Index* using the points added by *PointAdd* as vertices. The points stored by *PointAdd* are cleared after the polygon is drawn. The outline is drawn in the color of *PenColor* and filled using the pattern defined by *BrushStyle* and the colour defined by *BrushColor*.

The colour of an individual pixel can be read or set using the *Pixel* property.

Pixel (*X, Y, Index*) - String hex colour value. *X* and *Y* are the coordinates of the pixel and *Index* is the GIF frame. This property can be set to a colour value to change the pixel colour or the property can be read to return the colour of the pixel.

Example:

```
Gif.Pixel(10, 10, 0) = "0000FF"
```

This sets the pixel at point (10, 10) in frame 0 to blue, or the nearest available colour in the colour table for that frame.

3.4. Drawing Text

Text can be drawn onto a frame of the image using the *DrawText* function. The text colour and background are set by *PenColor* and *BrushColor*. There are properties to control the font face, style and size. Text can be drawn antialiased but this will change the colour table entries because extra colours are needed. Default property values are shown in brackets.

DrawText *X, Y, Index, Text* - This draws the text string *Text* at the coordinates *X, Y* on the frame specified by *Index*. The text can use Unicode characters, subject to a suitable font being used, and the string can contain carriage return characters to make it span multiple lines. To give the text a transparent background set *BrushStyle* to 1 for clear.

TextFont - String. The name of the font face to be used. This must be an installed font. ("Arial")

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

TextAngle - Real. The angle of rotation of the text in degrees measured anticlockwise from the horizontal. 0 is left to right text, 90 is written up the page from bottom to top. (0)

TextBold - Boolean. Set to true for bold text. (false)

TextItalic - Boolean. Set to true for italic text. (false)

TextStrikeout - Boolean. Set to true for a strikeout text style. (false)

TextUnderline - Boolean. Set to true for underlined text. (false)

Example of drawing text at co-ordinates (10, 10) on the first frame using Comic Sans MS font at a size of 20 pixels in a bold style:

```
Gif.TextFont = "Comic Sans MS"
Gif.TextSize = 20
Gif.TextBold = true
Gif.DrawText 10, 10, 0, "Sample Text"
```

TextJustify - Integer value 0 to 2. A choice of options for justifying text that spans multiple lines when carriage returns are used to split the lines.

0 - Left justify (default)
1 - Centre justify
2 - Right justify

Antialias - Boolean. When true, text will be drawn antialiased. This will change the colour table for the frame on which the text is drawn, and it will set *HasLCT* to true for this frame. This could also change the transparent colour for the frame, if it had one. (false)

The size of a text string can be found without drawing anything on the image using *TextWidth* and *TextHeight*.

TextWidth (*Text*) - Returns the width (length) of the string *Text*, in pixels, if it was to be drawn using the current text properties.

TextHeight (*Text*) - Returns the height of the string *Text*, in pixels, if it was to be drawn using the current text properties.

3.4.1. Automatically Wrapping Text

Text can be made to wrap to fit into a rectangular area, although it will only be broken at a space. To do this, *TextWrap* is set to true and *TextRectX* specifies the width of the text area. *TextRectY* can be set to limit the vertical size of the text block but this can lead to text getting clipped.

TextWrap - Boolean. When true, text written with *DrawText* will be broken to fit inside a box of width and height defined by *TextRectX* and *TextRectY*. The text will only be broken at a space. (false)

TextRectX - Integer. The maximum length of the text in pixels when *TextWrap* is true. When zero the text will wrap at the edge of the image. (0)

TextRectY - Integer. The maximum height of the text, in pixels, when *TextWrap* is true. When zero the text is not restricted. (0)

3.5. Filling Areas

There are two commands for filling an area with a colour, *FloodFill* and *FillToBorder*. The fill pattern is defined by *BrushStyle* and it will have no effect if this is set to 1 for clear fill.

FloodFill *X, Y, Index* - Fills an area on the frame defined by *Index* with *BrushColor* spreading outward from the point (X, Y) until a different colour is reached. All the pixels filled will have the same original colour as the pixel at (X, Y).

FillToBorder *X, Y, Index, Color* - Fills an area on the frame defined by *Index* with *BrushColor* spreading outward from the point (X, Y) until a boundary of *Color* is reached. *Color* is a 6 character hex string of the form "RRGGBB".

4. Image Manipulation (Resize, Rotate, Crop and Flip)

This section describes functions for manipulating the image and they can either be applied to the entire image or to an individual frame. Where a command manipulates the entire image the frame offest properties, *ImageLeft* and *ImageTop*, will be changed to keep the frames positioned relative to each other.

ResizeAll *Width, Height* - Resizes all the frames in the image so that the overall size is *Width* x *Height*. If either parameter is zero the other parameter is used for the new width or height and the aspect ratio is maintained. The values of *ImageTop* and *ImageLeft* are taken into consideration when calculating the overall size of the image.

ResizeFrame *Width, Height, Index* - Resizes the frame specified by *Index* to dimensions *Width* x *Height*. If either parameter is zero the other parameter is used for the new width or height and the aspect ratio is maintained. The values of *ImageTop* and *ImageLeft* are not changed.

ScaleAll *Factor* - Scales all the frames in the image by a percentage scaling of *Factor*. The values of *ImageTop* and *ImageLeft* are also scaled so that any offset frames will maintain their relative position.

ScaleFrame *Factor, Index* - Scales the frame specified by *Index* by a percentage scaling of *Factor*. The values of *ImageTop* and *ImageLeft* are unchanged.

RotateAll *Angle* - The whole GIF image is rotated by *Angle* degrees anticlockwise, where *Angle* is a real number (single precision). The values of *ImageTop* and *ImageLeft* are recalculated to maintain the relative position of offset frames. Rotations by an angle that is not a multiple of 90 degrees create triangular areas on the image that are the colour specified by *ExtraColor*.

RotateFrame *Angle, Index* - The frame specified by *Index* is rotated *Angle* degrees anticlockwise. The values of *ImageTop* and *ImageLeft* are unchanged. Rotations by an angle that is not a multiple of 90 degrees create triangular areas on the image that are the colour specified by *ExtraColor*.

CropAll *X1, Y1, X2, Y2* - Crops every frame to a rectangle with opposite corners at (X1, Y1) and (X2, Y2). These coordinates are measured from the top left of the logical screen so the values of *ImageTop* and *ImageLeft* are taken into consideration and can be changed. Frames are not increased in size by cropping to a larger area.

CropFrame *X1, Y1, X2, Y2, Index* - Crops the frame specified by *Index* to a rectangle with opposite corners at (X1, Y1) and (X2, Y2). The values of *ImageTop* and *ImageLeft* are unchanged. Cropping to a larger rectangle than the frame increases the frame size and the new area is the colour specified by *ExtraColor*.

ExtraColor - String hex colour value property. This is the colour used for the extra areas created during a rotation or an oversized crop. The colour must be in the active colour table or an approximation will be used. (Default = "FFFFFF")

ApplySmoothing - Boolean property. When true a resize, scale or rotation will use some resampling to give a smoother appearance. Each resampled frame will have its colours converted to a local colour table. In many cases this is not a desirable effect and that is why the property is false by default. Rotations by a multiple of 90 degrees are never resampled because they can be performed without degradation. Transparency is often lost during resampling. The resampling filter used is defined by the *FilterType* property, described below. (Default = false)

FilterType - Integer in the range 0 to 4. This sets the filter used when a resize is performed using *ApplySmoothing*. The available values are 0 - Halftone, 1 - Bilinear, 2 - Hermite, 3 - Bell, 4 - B-Spline. (Default = 0)

FlipAllX - Flips (reflects) the entire GIF image around a horizontal axis so that the top becomes the bottom and the bottom becomes the top. The values of *ImageTop* and *ImageLeft* are recalculated to maintain the relative position of offset frames.

FlipFrameX *Index* - Flips the frame specified by *Index* around a horizontal axis so that the top becomes the bottom and the bottom becomes the top. The values of *ImageTop* and *ImageLeft* are unchanged.

FlipAllY - Flips (reflects) the entire GIF image around a vertical axis so that the left becomes the right and the right becomes the left. The values of *ImageTop* and *ImageLeft* are recalculated to maintain the relative position of offset frames.

FlipFrameY *Index* - Flips the frame specified by *Index* around a vertical axis so that the left becomes the right and the right becomes the left. The values of *ImageTop* and *ImageLeft* are unchanged.

5. Frame Properties and Animation

This section covers methods and properties that are used for adding and reordering frames, as well as properties that control the appearance and display of animated GIFs.

AddFrame - Adds a new empty frame to the current GIF. The integer return value is the index of the new frame. The first frame has an index of zero. *AddFrame* must be called before a new frame can be loaded using one of the import methods or before calling *ClearImage* to create an empty drawing surface. The only other methods that add frames to a GIF are the import functions that load a complete GIF image containing multiple frames.

When *HasGCT* is true, the new frame uses the Global Colour Table, i.e. *HasLCT* is false for the new frame. When *HasGCT* is false, the new frame has a Local Colour Table.

DeleteFrame *Index* - This deletes the frame specified by *Index*.

CopyFrame *ToIndex, FromIndex* - This copies the frame specified by *ToIndex* and uses it to overwrite the frame specified by *FromIndex*.

ExchangeFrames *Index1, Index2* - This swaps the frames specified by indices *Index1* and *Index2*.

FrameCount - Integer, read only. The number of frames in the GIF.

Each GIF has a "logical screen" which can be larger than the frames. This logical screen size is stored in the properties *ScreenWidth* and *ScreenHeight*. Each frame is positioned on this logical screen relative to the top left corner and this position is set by *ImageLeft* and *ImageTop*. This enables a small image to be animated on a larger background to reduce the amount of pixel data that is stored. By default *ScreenWidth* and *ScreenHeight* are calculated as the smallest values that can store all the frames but they can also be set to higher values.

ImageLeft (*Index*) - Integer. The distance in pixels between the left of the logical screen and the frame specified by *Index*. (Default = 0)

ImageTop (*Index*) - Integer. The distance in pixels between the top of the logical screen and the frame specified by *Index*. (Default = 0)

FrameWidth (*Index*) - Integer, read only. The width of the frame specified by *Index*. This does not include *ImageLeft*. Set the frame width with *ClearImage*, *Resize*, *Scale* or *CropFrame*.

FrameHeight (*Index*) - Integer, read only. The height of the frame specified by *Index*. This does not include *ImageTop*. Set the frame height with *ClearImage*, *Resize*, *Scale* or *CropFrame*.

ScreenWidth - Integer. The width of the logical screen. This is calculated as the largest value of *ImageLeft* + *FrameWidth* or it can be set to a higher value.

ScreenHeight - Integer. The height of the logical screen. This is calculated as the largest value of *ImageTop* + *FrameHeight*or it can be set to a higher value.

Each frame of a GIF can have a transparent colour, which should allow the background from behind the image to show through, subject to the GIF reader interpreting it. If several frames have transparency they do not have to use the same colour.

Transparent (*Index*) - Boolean. When true, the frame specified by *Index* contains a transparent colour. (Default = false)

TransparentColor (*Index*) - Integer. This is the colour used as the transparent colour by the frame specified by *Index*. It is the index of the colour inside the colour table used by the frame, not the colour value itself. (Default = 0)

Example:

```
Gif.Transparent(0) = true
Gif.TransparentColor(0) = Gif.ColorIndex("FF0000", 0)
```

This sets transparency for the first frame and sets the transparent colour to red, assuming red is present in the frame colour table. When the colour index is known it can be used directly:

```
Gif.TransparentColor(0) = 180
```

TransparentColorString (*Index*) - String, read only. This returns the colour that is used as the transparent colour in the frame specified by *Index*. It is the value of the colour as a string.

RemoveTransparency - This method sets the transparent property of each frame to false.

Interlaced (*Index*) - Boolean. When true the frame specified by *Index* will be stored interlaced. This should not be used with multiple frame GIFs and csASPGif will not interlace multiple frames during export, regardless of this property value. Interlacing was traditionally used when images were transmitted over slow connections. (Default = False).

Delay (*Index*) - Integer. The length of time the frame specified by *Index* will be displayed before moving to the next frame. This is measured in 1/100ths of a second. (Default = 0)

Most web browsers will not show more than 10 frames per second, so when values of less than 10 are used for *Delay*, the animation may run slower than expected. The same animation may run at the correct speed when viewed in other image viewing software.

LoopCount - Integer. The number of times the entire animation will repeat. When this value is zero it will repeat indefinitely. The maximum value is 65536. The first run through the animation is not counted so setting *LoopCount* to 1 results in two complete iterations. (Default = 0).

HideLoopCount - Boolean. The data block inside the GIF that specifies the loop count can be omitted if necessary by setting this property to true. Technically it is not a standard part of the GIF format and was added by Netscape so that Netscape 2.0 could display animations, but most GIF readers use it. Set *HideLoopCount* to true if only one pass through the animation is required. (Default = false).

DisposeMethod (*Index*) - Integer value 0 - 3. This property describes what should happen to the frame specified by *Index* when the next image is displayed. The exact behaviour depends on the GIF reader.

0 - Unspecified. This is usually interpreted in the same way as 1, No Disposal.

1 - No Disposal. The image remains in the display and will still be visible if the next image is smaller or offset. (Default)

2 - Restore Background. The image is removed and the background behind the GIF is displayed.

3 - Restore Previous. This restores the display to whatever was there before the current frame.

The following properties are rarely supported by GIF readers and should not generally be used.

BackgroundColor - Integer. The colour to be displayed behind the GIF when frames do not fill the logical screen, defined by the index within the Global Colour Table. Most GIF readers will show the background as transparent instead of this colour. (Default = 0)

UserInput (*Index*) - Boolean. When true, the frame specified by *Index* will remain displayed until some input is received from the user. (Default = false)

6. Frame Optimisation

It is sometimes possible to reduce the file size of an animated GIF by either cropping some frames and allowing the background to fill the remaining area, or by allowing some of the previous frame to show. csASPGif provides an optimisation method called *OptimizeFrames*.

OptimizeFrames - This method will attempt to reduce the file size of an animated GIF by cropping frames where possible and changing areas to transparent to allow the background or previous frame to show through. A smaller frame will produce a smaller file size and increasing the amount of a frame that is a uniform colour can improve compression.

If the GIF contains any local colour tables, these will be replaced with a global colour table. Any previous optimisation will be removed. This is the equivalent of calling the *OptimizeColorTables* and *UnOptimize* methods. There is no guarantee that *OptimizeFrames* will reduce the file size and if the GIF already contained some optimisation it could even increase the file size.

UnOptimize - This method removes any previous optimisation. It will make each frame the same size as the logical screen. Where the dispose method and transparency has been used to generate an image made from one or more previous frames this will be copied to each frame.

UnOptimize can be used to display each frame as a stand alone image.

Both *OptimizeFrames* and *UnOptimize* can be time consuming to run and it might not be advisable to use them when generating images "on the fly".

7. Merging Images

External images can be merged with existing frames. This is simply sticking one image over another although one of the colours of the foreground image can be transparent.

There are three merge commands, *MergeFile*, *MergeData* and *MergeHandle* which give three different options for the format of the external image, either a file, a variant array, or a handle to a bitmap.

There are several properties that need to be set before calling a merge function. *MergePalette* specifies what happens to the colour table of the frame. *MergeReverse* determines which image is the foreground. *TransparentMerge* and *TransparentMergeColor* control the transparency during the merge operation.

MergeFile *X, Y, Index, FileName* - This merges the GIF file at *FileName* with the frame specified by *Index*. *FileName* is a string and must be a full physical path to a GIF file. The top left corner of the foreground image is positioned at coordinates (X, Y) on the background image.

MergeData *X, Y, Index, GIFData* - This merges the GIF file stored in the variable *GIFData* with the frame specified by *Index*. *GIFData* is a variant array in the same format as that read by *ReadStream*. The top left corner of the foreground image is positioned at coordinates (X, Y) on the background image.

MergeHandle *X, Y, Index, Handle* - This merges the image defined by the bitmap *Handle* with the frame specified by *Index*. *Handle* is the handle to a bitmap such as is exported by the *FrameHandle* function. The top left corner of the foreground image is positioned at coordinates (X, Y) on the background image.

MergeReverse - Boolean. This property determines which image is the foreground during a merge. By default *MergeReverse* is false and the frame image is the background and the external image is the foreground. Setting *MergeReverse* to true causes the external image to be the background.

TransparentMerge - Boolean. When true, the colour specified by *TransparentMergeColor* will be transparent during a merge operation. (Default = false)

TransparentMergeColor - String hex colour value. This colour will be transparent during a merge operation if *TransparentMerge* is true. It applies to the merge only and is unrelated to frame transparency. (Default = "FFFFFF")

MergePalette - Integer value 0 - 2. This property determines what happens to the colour table during a merge operation. Frequently the number of different colours involved in the merge is greater than the size of the colour table so some compromise must be used.

0 - Combine the colour tables. A new colour table will be created from the two images and this will be a Local Colour Table. *HasLCT* will be set to true for the frame merged.

1 - Existing. The merged frame will keep its colour table. (Default)

2 - New. The colour table for the external image will be adopted by the frame. *HasLCT* will be set to true for the merged frame.

8. Registering the Component and Getting Started

8.1. Registration and Server Permissions

Before the component can be used the DLL file, csASPGif.dll (or csASPGifTrial.dll for the trial version) must be registered on the server. This can be done using the command line tool REGSVR32.EXE which should be in the Windows System folder. The syntax is:

regsvr32 *dllname*

where *dllname* is the path and name of the DLL to register. Chestysoft has a free utility that performs this function through a Windows interface which can be easier although the result is identical. This tool can be downloaded from the Chestysoft web site: www.chestysoft.com/dllregsvr/default.asp

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 any file reading, writing or deleting that the component is to perform. For network access the csASPGif component must be added to a COM+ Application in Component Services.

8.2. 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 Gif = Server.CreateObject("csASPGif.Gif")
```

For the trial version:

```
Set Gif = Server.CreateObject("csASPGifTrial.Gif")
```

In both cases the object name is "Gif", but any variable name could be used.

8.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csASPGifTrial.dll, with a class name of "csASPGifTrial.Gif". 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 Gif = Server.CreateObject("csASPGifTrial.Gif")
Response.Write Gif.Version
```

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

9. Streaming an Image to the Browser

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" 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 Gif = Server.CreateObject("csASPGif.Gif")
      Gif.ReadFile "C:\images\big.gif"
      Gif.ScaleAll 25
      Response.ContentType = "image/gif"
      Response.BinaryWrite Gif.GIFData
%>
```

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 GIF 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 Internet Explorer but it might not in other 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="image.asp?NewSize=150">
```

10. Language Specific Issues

All the examples that are shown with the command descriptions use ASP and VBScript. The csASPGif component is a COM object and can be used in most COM enabled environments running on a Windows platform. We cannot begin to cover all the possible development environments here so instead we concentrate on ASP and in this section we show the syntax for Cold Fusion and Visual Basic.

10.1. 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 from IIS 5 their use does not generate an error. For example:

```
Gif.ReadFile "C:\images\a.gif"
```

These instructions show methods without brackets surrounding the parameters.

Properties with an input parameter do require brackets. Failure to include the brackets results in the error "Object doesn't support this property or method". The correct syntax is:

```
Response.Write Gif.GCTEntry(0)
```

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

```
Gif.HasGCT = true
```

10.1.1. 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:

```
Gif.ReadFile("C:\\images\\a.gif");
```

10.2. Cold Fusion

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

<cfobject action="create" name="gif" class="csASPGif.Gif">

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

```
<cfset Gif.ReadFile("c:\images\big.gif")>
<cfset Gif.ScaleAll(25)>
<cfset Gif.WriteFile("c:\images\small.gif")>
```

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

```
<cfscript>
Gif.ReadFile("c:\images\big.gif");
Gif.ScaleAll(25);
Gif.WriteFile("c:\images\small.gif");
</cfscript>
```

Cold Fusion version 5 does not support variant arrays and so the *GIFData*, *ReadStream* and *MergeData* commands 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 MX without saving to a temporary file first. The following commands will stream an image assuming an image is loaded into a csASPGif object called "Gif".

```
<cfscript>
Context = GetPageContext();
Context.SetFlushOutput(false);
Response = Context.GetResponse().GetResponse();
Out = Response.GetOutputStream();
Response.SetContentType("image/gif");
Out.Write(Gif.GIFData);
Out.Flush();
Response.Reset();
Out.Close();
</cfscript>
```

10.3. Visual Basic

For best results import the csASPGif type library into VB by selecting "Project" from the menu bar, then "References". The dialogue box will then show available type libraries. Scroll down to "csASPGif Library", check the box and click OK. This will add the "Gif" class from csASPGif to the Object Browser, making it available for early binding.

To create an instance of the object called "GifObj" use the following code:

```
Dim GifObj As Gif
Set GifObj = CreateObject("csASPGif.Gif")
```

11. Important Features of GIFs

Some knowledge of the GIF format is needed to be able to work with the csASPGif component. The important features are summarised here.

11.1. Colours and Colour Tables

The GIF image format allows multiple images, called frames, to be stored in the same file. Usually this is done to create an animation, usually for displaying on web pages. All the frames must be stored at the same colour depth (the number of pixels per byte) and this can be up to a maximum of 8 bits, or 256 colours.

The frames use colours that are stored as an indexed palette or "colour table". The GIF may have a Global Colour Table, which can be applied to some or all of the frames. Each frame may have a Local Colour Table, or it can use the Global Colour Table. It is possible to have more than 256 colours in the image but no more than 256 colours can be in any one frame. Each colour is an RGB triple with red, green and blue components having values between 0 and 255.

If a small file size is required it is preferable to use the Global Colour Table for all the frames. When the frames are small, having a Local Colour Table for each frame can add a significant size to the file.

11.2. Compression

The image data is compressed using a system that is most efficient over areas of continuous colour. Simple shapes and areas filled with single colours compress very well. Photographic images and images with a lot of different colours in a small area will not compress efficiently.

11.3. The Logical Screen and Frame Co-ordinates

The frames do not need to be the same size. A GIF has a "logical screen" and each frame is positioned at co-ordinates measured from the left and top of this logical screen. During animation each frame has a "dispose method" where it can be left in place with the next frame drawn on top, or it can be removed. This means that an initial background frame could be drawn with smaller frames placed on this background. This is another method that can be used for reducing the file size if only a small part of an image is changed during animation.

11.4. Transparency

Each frame can have a transparent colour. It does not have to be the same colour in each frame. Transparent pixels will be the colour of whatever is behind the GIF. In web pages this allows images to blend into the background which is useful if they are to appear as an irregular shape.

11.5. Timing and Looping an Animation

In an animated GIF each frame has a delay time, specified in 1/100ths of a second, which is the time it will display before being replaced by the next frame. This time can be slowed down if the computer displaying the GIF is busy. It is possible to specify that the entire animation is to be repeated and this can either be for a set number of loops or it can repeat indefinitely. This is an extension to the GIF format sometimes called the Netscape 2.0 Application Extension, because it was introduced in the Netscape 2 browser. It is not possible to repeat only part of the GIF.

12. Revision History

The current version of csASPGif is 2.0.

The main changes since version 1.0 are described below.

New in Version 2.0:

OptimizeFrames method added to reduce final image sizes.
UnOptimize method added to allow the extraction of individual frames from optimised images.
ApplySmoothing property added to provide optional resampling on resize and rotate.
HasLocalColourTables and TransparentColorString properties added.

13. Other Products From Chestysoft

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

ActiveX Controls

csXImage	-	An OCX control to display, edit and scan images.
csXGraph	-	An OCX control to draw pie charts, bar charts and line graphs.
csXThumbUpload	-	Upload multiple files by HTTP or FTP with previews and image edits.
csXPostUpload	-	Uploads batches of files from a client to a server using an HTTP post.
csXMultiUpload	-	Select and upload multiple files and post to a server using HTTP.

csImageFile	-	Similar functionality to csXImage but in an ASP component.
csDrawGraph	-	Component to draw pie charts, bar charts and line graphs.
csIniFile	-	Read and Edit Windows style inifiles.
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.

ASP.NET

csASPNetGraph	-	A .NET component to draw pie charts, bar charts and line graphs.
csNetUpload	-	ASP.NET component for saving HTTP uploads.
csNetDownload	-	ASP.NET class to control file downloads.

Web Hosting

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

14. Alphabetical List of Commands