Table of Contents

Introduction

Welcome to the Alagad Image Component documentation. This documentation is indented to help you integrate the Alagad Image Component into your ColdFusion application.

The Alagad Image Component is a 100% native ColdFusion component (CFC) for manipulating images. This component requires no third party software to operate and no additional configuration.

This documentation is specific to the Alagad Image Component 2.0. Throughout the rest of the document the Alagad Image Component 2.0 will simply be referred to as the Alagad Image Component.

This documentation assumes that you have some experience with ColdFusion and CFCs.

Conventions Used In This Document:

Words surrounded in angle brackets indicate ColdFusion or HTML tags. For example, <cfset> indicates the ColdFusion cfset tag.

Words highlighted in blue followed by parenthesis indicate Alagad Image Component methods. For example drawRectangle() indicates the drawRectangle method.

ColdFusion and HTML code is highlighted in brown. For example:

```
<cfset example="Hello World" />
```

Paths to files or directories are surrounded in quotation marks. For example "/MyDirectory/".

The "¬" character at the end of a line of example code indicates a line which should be continued on the same line, even though it's shown as wrapping.

About the Alagad Image Component

The Alagad Image Component is a ColdFusion Component (CFC) used to create and manipulate image files. Written entirely in ColdFusion CFML, the Image Component does not require installation of any additional software. The Image Component is not a CFX tag and is not Platform dependant.

Because the Image Component is written in pure ColdFusion and instantiates only native Java objects, it compiles with the rest of your CFML files to Java bytecode and can perform much better than competing products.

Using the Image Component you can perform all of the most frequently requested image related functions such as:

- Read and Write images

- Create and Draw into images
- Resize images
- Find the width and height of images
- Draw text into images
- Draw images into other images (water marking)
- Control Image Compression Quality
- Much, much, more! (Over 80 methods provided.)

The Alagad Image Component requires almost no effort to install and use. Simply place the component into your custom tags directory or any directory in your site and then instantiate it using the ColdFusion CreateObject method. Once you have the Image Component instantiated you can use its methods to begin creating and manipulating images.

Requirements to Use the Image Component

The Alagad Image Component works with ColdFusion MX 6.1 and later on any supported platform.

ColdFusion MX 6.1 users will not need to do any additional configuration for the Alagad Image Component to work.

Users who are running ColdFusion MX without the 6.1 update will need to install the 6.1 update which is freely available on macromedia.com.

The Image Component also works BlueDragon 6.1 and later servers, except for the free versions which do not allow usage of precompiled code.

If you are using the Image Component on a platform which is not supported by ColdFusion MX 6.1, such as OS X, you need to use a 1.4.1 or later JRE.

Using the Image Component on Shared Hosting

The Alagad Image Component will work in a shared hosting environment. However, many hosting providers use “Sandbox” security to disable certain functionality on the server.

For the Alagad Image Component to function, your hosting provider may need to enable the CreateObject() function and the <cffile> tag. Contact your hosting provider for more information.

Strange Error Messages and Headless Systems

Error: This graphics environment can be used only in the software emulation mode

Several users of the Alagad Image Component have run into a limitation of Java on Unix platforms. These unfortunate people receive the, "This graphics environment can be used only in the software emulation mode" error on most calls to Image Component methods.

The problem is not with the Image Component, it stems from limitations of Java running on systems which do not have a mouse, keyboard or display. These systems are said to be "headless". On Unix, the java.awt package requires you to have an X server installed.

As of Java 1.4.1, Sun added a work around for headless systems. By starting Java with "-Djava.awt.headless=true" argument you should be able to resolve the error message.

Macromedia has posted a TechNote 18747 which addresses these problems at http://www.macromedia.com/support/coldfusion/ts/documents/graphics_unix_141jvm.htm

However, the work around does not always work. What follows are two other work-around.

Use Pure Java AWT (PJA).

Pure Java AWT (PJA) is an implementation of the Java AWT libraries in pure java. All you need to do to use it is download the package, extract it, point ColdFusion at it, and it should work.

First, download PJA 2.5 or later. At the time of writing version 2.5 was a beta release, but it has worked well for many Image Component users.

After downloading the PJA ZIP file extract it to a directory, for example, /usr/PJA. This creates a directory /usr/PJA/lib which contains pja.jar. To get ColdFusion to use PJA edit your jvm.config file which is located at {cf directory}/bin/jvm.config. There is a line in this file which reads something like (all on one line):

```
java.args=-server -Xmx512m -Dsun.io.useCanonCaches=false ¬
-Xbootclasspath/a:{application.home}/lib/webchartsJava2D.jar ¬
-XX:MaxPermSize=128m -XX:+UseParallelGC ¬
-Djava.awt.graphicsenv=com.gp.java2d.ExHeadlessGraphicsEnvironment
```

This needs to be changed. First, add the path to the pja.jar file and rt.jar onto the -Xbootclasspath attribute. To add additional paths, separate them with a colon ":" like this (note the addition of /opt/coldfusionmx/runtime/jre/lib/rt.jar and /usr/PJA/lib/pja.jar) (all on one line):

```
-Xbootclasspath/a:{application.home}/lib/webchartsJava2D.jar:/opt/¬
coldfusionmx/runtime/jre/lib/rt.jar:/usr/PJA/lib/pja.jar
```

This allows Java to find the pja.jar and rt.jar files at startup, which are needed for PJA to work. (Simply adding these paths to the Java classpath will not work.)

Next, delete the following:

```
-Djava.awt.graphicsenv=com.eteks.java2d.PJAGraphicsEnvironment ¬
-Djava.awt.fonts=/usr/java/j2sdk1.4.0_03/jre/lib/fonts
```

And replace it with the following:

```
-Djava.awt.graphicsenv=com.eteks.java2d.PJAGraphicsEnvironment ¬
-Djava.awt.fonts=/usr/java/j2sdk1.4.0_03/jre/lib/fonts
```

This tells Java to use PJA for calls to the java.awt packages. It also tells java to look in the /usr/java/j2sdk1.4.0_03/jre/lib/fonts directory for fonts. You will probably want to find the fonts directory which is correct for your server and use that. As a note, this example used a directory from a different JRE than the one provided with ColdFusion. This is because the ColdFusion version does not include a fonts directory. This directory simply contains a set of TTF files so you should be able to point this somewhere else, if you want to.

The resulting java.args line should like this (all on one line):

```
java.args=-server -Xmx512m -Dsun.io.useCanonCaches=false -
Xbootclasspath/a:{application.home}¬
/lib/webchartsJava2D.jar:/usr/PJA/lib/pja.jar -XX:MaxPermSize=128m -
XX:+UseParallelGC -
Djava.awt.graphicsenv=com.eteks.java2d.PJAGraphicsEnvironment -
Djava.awt.fonts=/usr/java/j2sdk1.4.0_03/jre/lib/fonts
```

After saving your updated jvm.config, restart ColdFusion and the Alagad Image Component should work.

Installation of the Image Component

Installation of the Alagad Image Component is quite simple. First, download the Alagad Image Component in .zip format from http://www.alagad.com. Once you have the Image Component, extract the Image.cfc file from the Zip file from the correct folder. If you are using ColdFusion server you will want to extract the Image.cfc file from “ColdFusion MX 6.1 and MX 7 Version” folder. If you are using BlueDragon you will want to extract the Image.cfc file from the “BlueDragon JX 6.x Version (Beta)” folder, where x indicates the version of BlueDragon you are using.

If you want to use the Image Component from any website on your server you can place the Image.cfc file in any custom tags folder.

If you want to use the Image Component in only one website, copy the Image.cfc file into the website’s root or any directory under your web site’s root.

If you want to place the Image Component outside of your web site directory hierarchy and out side of any of custom tags folder, then simple copy the Image.cfc file anywhere you want on your file system and create a mapping to that directory using the ColdFusion administration interface. For more information see the ColdFusion documentation.

Instantiating the Image Component

You must instantiate the Image Component before you can call its methods. The following examples demonstrate instantiating the Image Component using the ColdFusion <cfobject> tag and CreateObject() function.

For more information on using ColdFusion components see the Using ColdFusion components section of the ColdFusion documentation.

Instantiation from a Local Directory

If you installed the Image Component in a directory under your web site's root you would instantiate it as follows below. In both cases we are assuming the Image.cfc file was placed in a directory "/path/to/" under your web site's root. We are creating an instance of the Image.cfc named myImage.

Using <cfobject>:

```
<cfobject component="path.to.Image" name="myImage" />
```

Using CreateObject():

```
<cfset myImage = CreateObject("Component", "path.to.Image") />
```

Instantiation from a Mapped Directory

If you installed the Image Component in a ColdFusion mapped directory you would instantiate it as follows below. In both cases we are assuming the Image.cfc file was placed in a directory which was mapped to with the logical path "/mapped/path/to/". We are creating an instance of the Image.cfc named myImage.

For more information on ColdFusion mappings see the ColdFusion documentation.

Using <cfobject>:

```
<cfobject component="mapped.path.to.Image" name="myImage" />
```

Using CreateObject():

```
<cfset myImage = CreateObject("Component", "mapped.path.to.Image") />
```

Instantiation from a Custom Tag Path

If you installed the Image Component in a ColdFusion custom tags path you would instantiate it as follows below. In both cases we are creating an instance of the Image.cfc named myImage.

Using <cfobject>:

```
<cfobject component="Image" name="myImage" />
```

Using CreateObject():

```
<cfset myImage = CreateObject("Component", "Image") />
```

Using the Instantiated Image Component

Once you have instantiated the Image Component you can call methods on the component to perform image related actions such as reading, creating, and manipulating images. Below are some complete examples of the most common actions.

These examples use the CreateObject() to instantiate the object, but the same process works equally well with <cfobject>.

Reading and Writing an Image

The following example demonstrates how to read a GIF image and save it as a PNG.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- read the source GIF image --->
<cfset myImage.readImage("d:\examples\buy-now.gif") />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\buy-now.png", "png") />

<!--- output both images --->
<p>
<b>buy-now.gif:</b><br>
<img src="/examples/buy-now.gif">
</p>

<p>
<b>buy-now.png:</b><br>
<img src="/examples/buy-now.png">
</p>
```

Results

Creating and Drawing Into a New Image

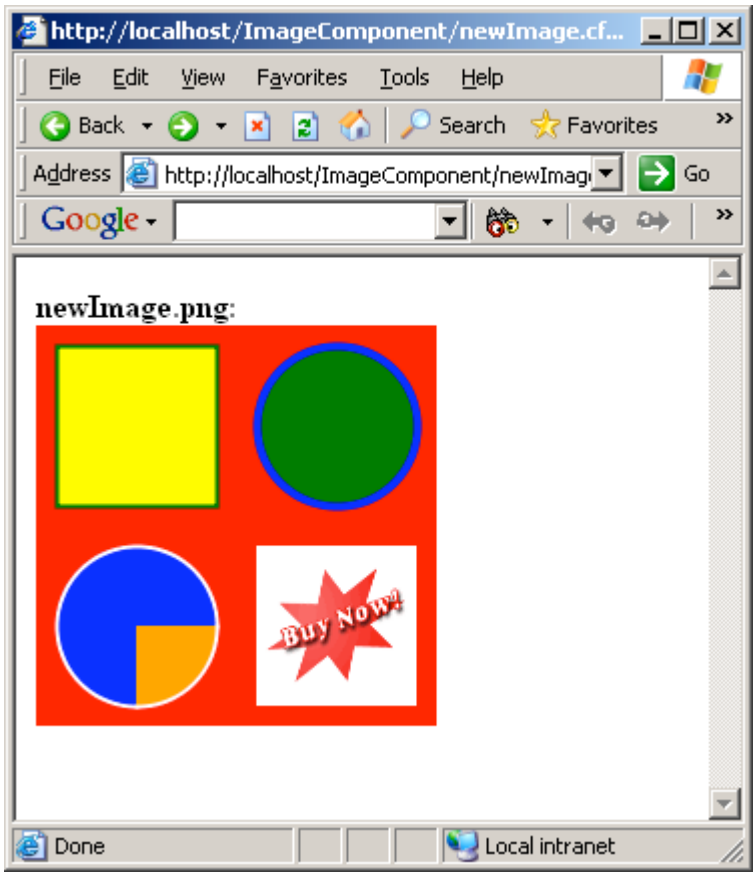
The following source code demonstrates how to create colors, set the background color of the image, create a new image, and draw filled and stroked shapes and images into the image.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- create some colors --->
<cfset red = myImage.getColorByName("red") />
<cfset yellow = myImage.getColorByName("yellow") />
<cfset green = myImage.getColorByName("green") />
<cfset blue = myImage.getColorByName("blue") />
<cfset orange = myImage.getColorByName("orange") />
<cfset white = myImage.getColorByName("white") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(red) />
```

```
<!--- create a new image --->
<cfset myImage.createImage(200, 200) />

<!--- draw a few shapes into the new image --->
<!--- draw a square --->
<cfset myImage.setFill(yellow) />
<cfset myImage.setStroke(2, green) />
<cfset myImage.drawRectangle(10, 10, 80, 80) />

<!--- draw a circle --->
<cfset myImage.setFill(green) />
<cfset myImage.setStroke(4, blue) />
<cfset myImage.drawOval(110, 10, 80, 80) />

<!--- draw two arcs --->
<cfset myImage.setStroke(2) />
<cfset myImage.setFill(blue) />
<cfset myImage.drawArc(10, 110, 80, 80, 0, 270) />
<cfset myImage.setFill(orange) />
<cfset myImage.drawArc(10, 110, 80, 80, 270, 90) />

<!--- draw another image into this image --->
<cfset myImage.drawImage("d:\examples\buy-now.gif", 110, 110, 80, 80)/>

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>newImage.png:</b><br>
<img src="/examples/newImage.png">
</p>
```

Results



Resize an Image

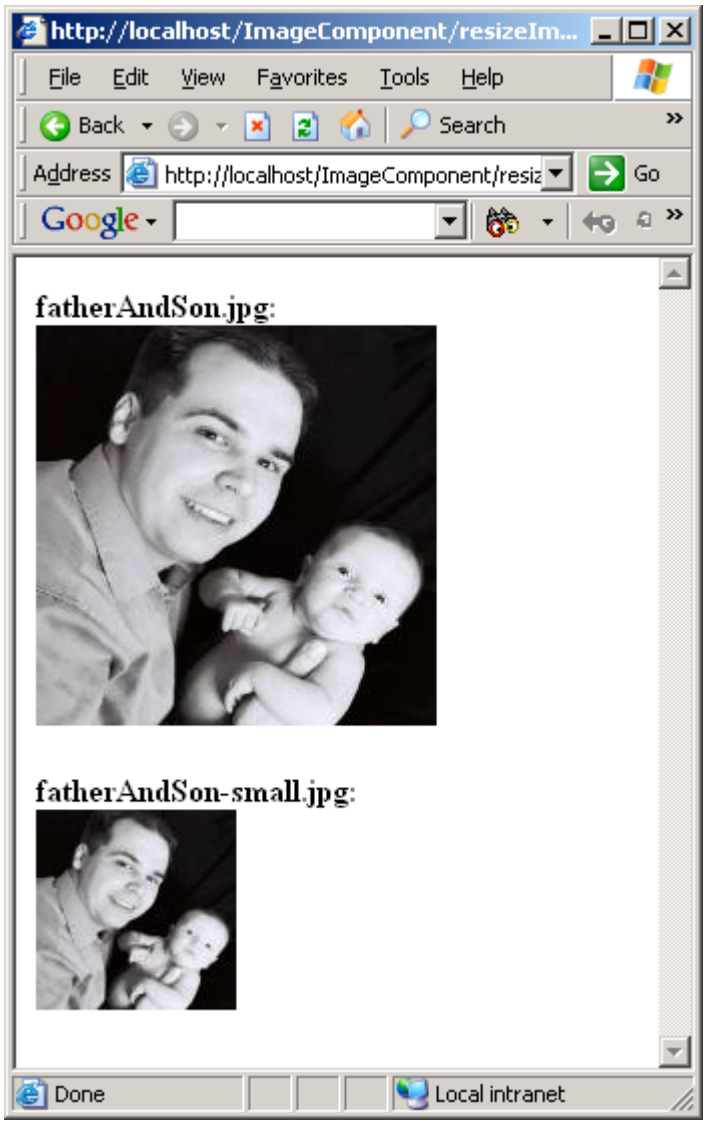
The following source code demonstrates how to resize an image to a specific width and height using the scalePixels() method. The Image Component provides several additional methods for resizing images. For more information see Image Size Methods.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image to a specific width and height --->
<cfset myImage.scalePixels(100, 100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\fatherAndSon-small.jpg", "jpg") />

<!--- the new images --->
<p>
<b>fatherAndSon.jpg:</b><br>
<img src="/examples/fatherAndSon.jpg">
</p>
<p>
<b>fatherAndSon-small.jpg:</b><br>
<img src="/examples/fatherAndSon-small.jpg">
```

```
</p>
```

Results



Find the Width and Height of an Image

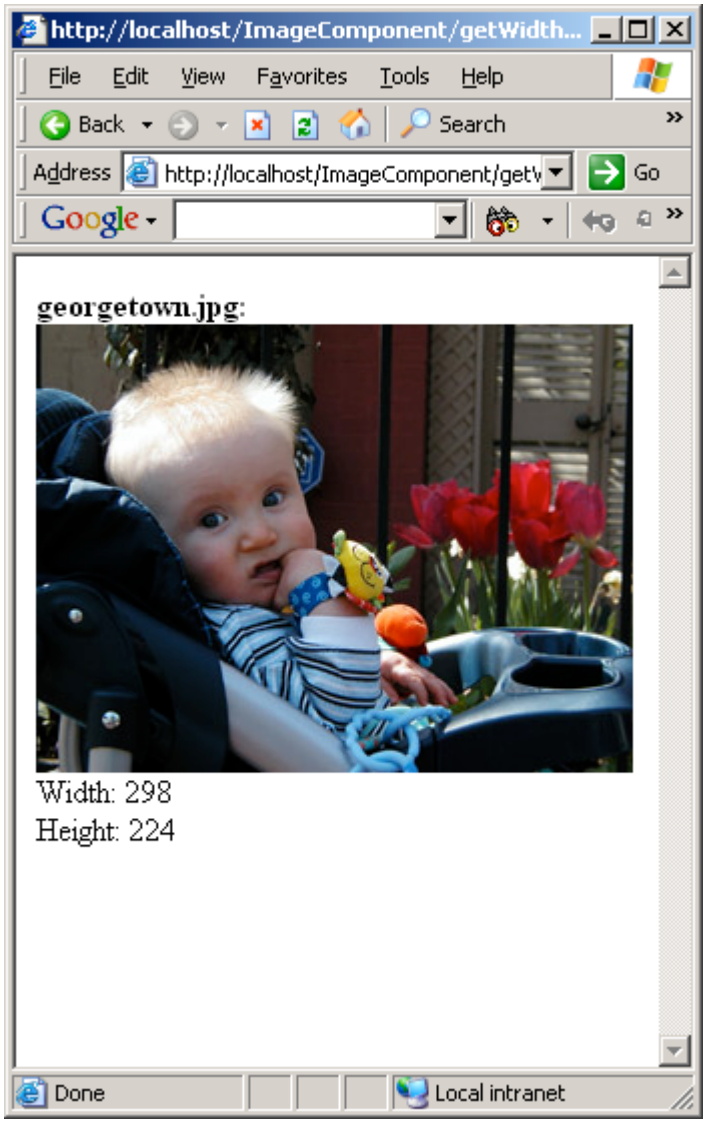
The following source code demonstrates how to open an image and get its width and height.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\georgetown.jpg") />

<!--- get the width --->
<cfset width = myImage.getWidth() />
<!--- get the height --->
<cfset height = myImage.getHeight() />

<!--- output the image size --->
```

```
<p>
<b>georgetown.jpg:</b><br>
<img src="/examples/georgetown.jpg"><br>
<cfoutput>
      Width: #width#<br>
      Height: #height#
</cfoutput>
</p>
```

Results



Draw Text on an Image

The following source code demonstrates how to use the Image Components simple methods to draw text into an image with a specific font and style. The Image Component also provides support for more advanced string formatting and styling. See also Advanced Text Formatting.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
```

```
<!--- open the image to write into --->
<cfset myImage.readImage("d:\examples\catAndCup.jpg") />

<!--- set the font --->
<cfset timesNewRoman = myImage.loadSystemFont("Times New Roman", 20, ¬
"boldItalic") />

<!--- create a string to write into the image --->
<cfset myString = "Where the heck is my milk?" />

<!---
      Find the metrics (width, height, etc) for the string.
      We will use this to center the string in the image.
--->
<cfset metrics = myImage.getSimpleStringMetrics(myString, ¬
timesNewRoman) />

<!--- determine the X coordinate so we can center the text ¬
in the image --->
<cfset x = (myImage.getWidth() - metrics.width) / 2 />

<!--- draw the text, centered at the top of the image --->
<cfset myImage.drawSimpleString(myString, x, 30, timesNewRoman) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\catCupAndText.jpg", "jpg") />

<!--- the new image --->
<p>
<b>catCupAndText.jpg:</b><br>
<img src="/examples/catCupAndText.jpg"><br>
</p>
```

Results

Draw Images into an Image

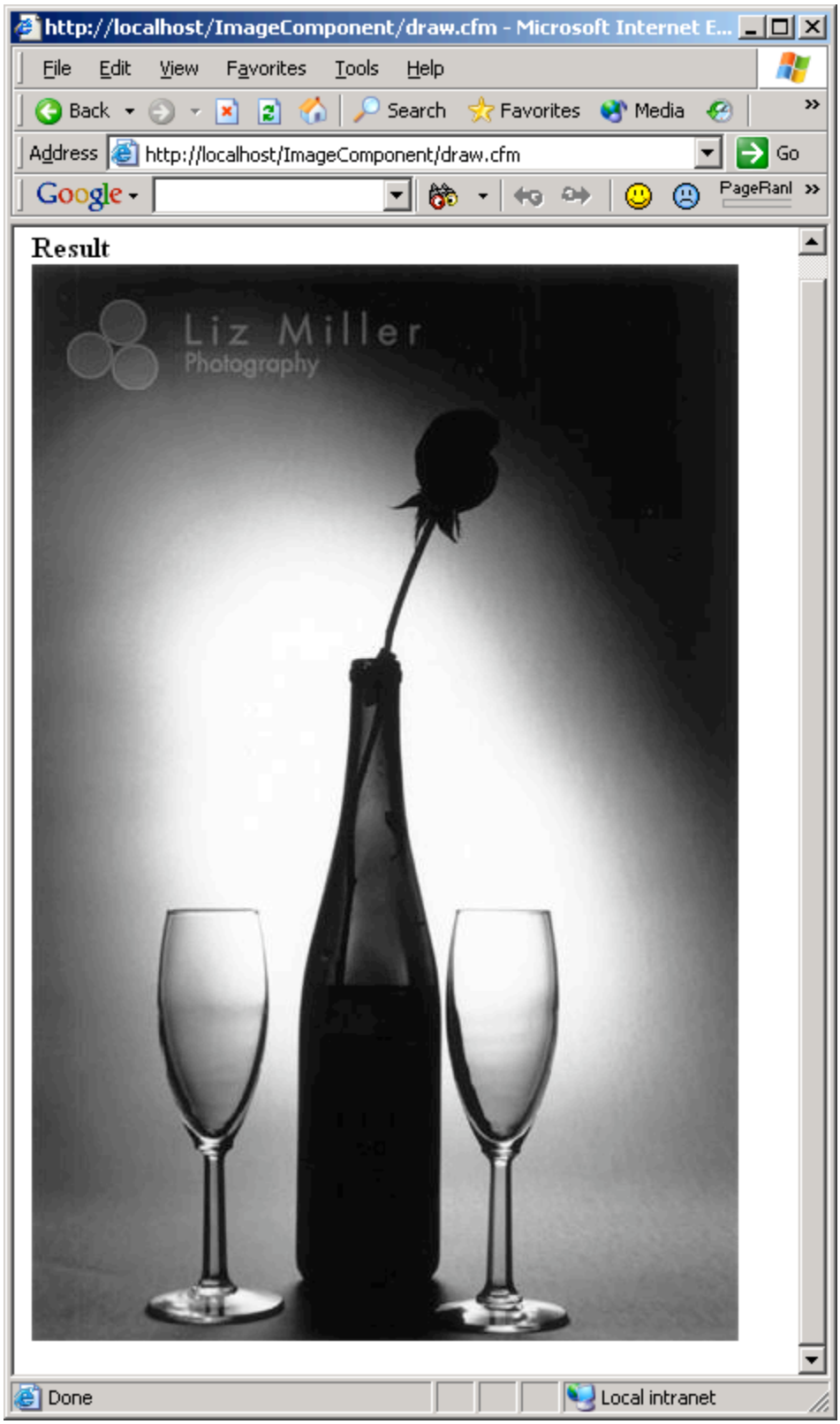
The following example draws a company logo into the upper left corner of an image. The logo has an alpha channel which is used to composite the two images so that the image shows through the transparent portions of the logo. Additionally, the logo is made to be slightly transparent.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- open a new image --->
<cfset myImage.readImage("d:\examples\wineRose.jpg") />

<!--- set the transparency used when drawing into the image --->
<cfset myImage.setTransparency(25) />
```

```
<!--- draw an image into the image --->
<cfset myImage.drawImage("d:\examples\logo.png", 20, 20) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\wineRoseLogo.jpg", "jpg") />

<!--- output the image --->
<b>Result</b><br>
<img src="/examples/wineRoseLogo.jpg">
```

Results



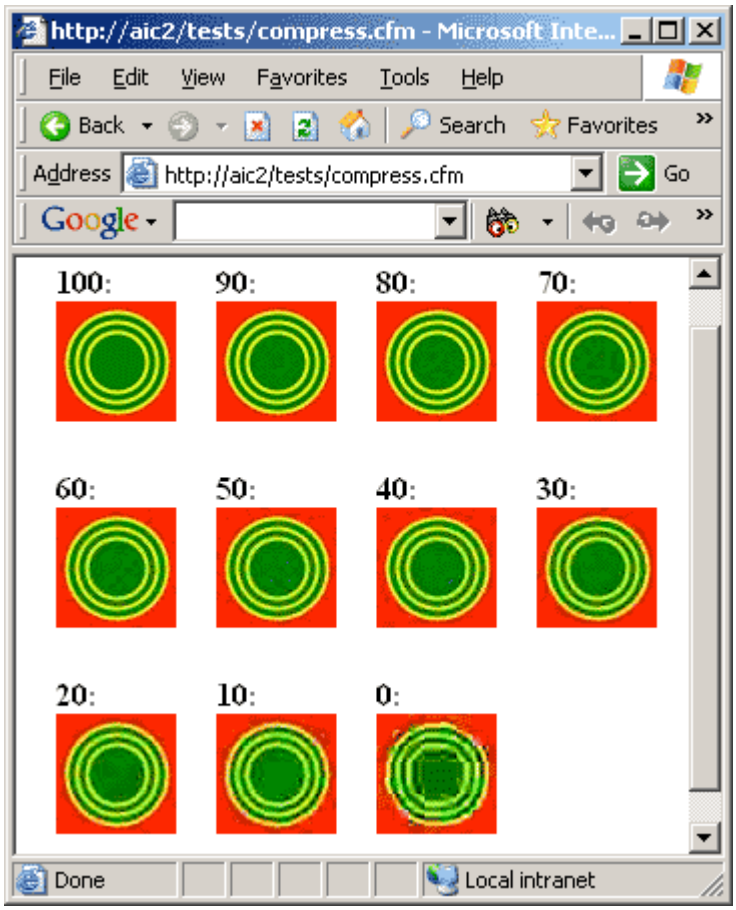
Control Image Compression Quality

The following source code demonstrates how to control compression settings using the Alagad Image Component.

Source Code

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset red = myImage.getColorByName("red") />
<cfset yellow = myImage.getColorByName("yellow") />
<cfset green = myImage.getColorByName("green") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(red) />

<!--- create a new image --->
<cfset myImage.createImage(60, 60, "rgb") />

<!--- draw a circle --->
<cfset myImage.setFill(green) />
<cfset myImage.setStroke(2, yellow) />
<cfset myImage.drawOval(5, 5, 50, 50) />
<cfset myImage.drawOval(10, 10, 40, 40) />
<cfset myImage.drawOval(15, 15, 30, 30) />

<!--- loop 11 times and output low to high quality --->
<cfloop from="100" to="0" step="-10" index="x">
 <cfset myImage.writeImage(expandPath("example#x#.jpg"), "jpg", x) />
 <cfoutput>
  <span style="float: left; padding: 10px;">
  <b>#x#:</b><br>
  <img src="example#x#.jpg">
  </span>
 </cfoutput>
</cfloop>
```

Results



Supported Image Formats

By default the Image Component can read GIF, JPEG and PNG files and can output JPEG and PNG images. Depending on the configuration of your server you may be able to use additional file formats. For a complete list of supported file formats call the getReadableFormats() and getWritableFormats() methods.

A tutorial on Alagad.com explains how to add support for the following file formats:

BMP, JPEG, JPEG 2000, PNG, PNM, Raw, TIFF, and WBMP image formats.

The tutorial can be found at http://www.alagad.com/index.cfm/name-aicformats.

Adding additional file formats may not be supported by Alagad.

Image Component Methods by Category

The Alagad Image Component has dozens of methods that you can use for reading, creating, manipulating and writing image files.

The following is a categorized list of Image Component methods.

Working with Images and Image Files

Overview

The Image Component is designed to work with and manipulate one image at a time. When you load or create an image, its data and current state is stored in instance variables within the Image Component. As you call various methods on the Component you are manipulating the state of the image and the image data.

At any time, you can call the writeImage() method to write the current image to disk. At any time you can also overwrite the instance's image with another image or create a whole new image by calling createImage() or readImage().

If you need to manipulate more than one image at the same time, then use multiple instances of the Image Component. Each instance operates separately.

createImage()

Description

The createImage() method sets the current image being manipulated to a new Image. The image created is of the specified width, height and type. The image is uniformly set to the current background color.

Syntax

createImage(width, height, type)

Parameter	Required	Type	Description
Width	Yes	Numeric	Width of the new image in pixels.
Height	Yes	Numeric	Height of the new image in pixels.
Type	No	String	Type of the new image. Options are: • **ARGB** – *(default)* Creates an RGB image with an alpha transparency channel. Note: JPEGs can not be ARGB. If you're creating an image to save as a JPEG use RGB. • **RGB** – Creates an RGB image without an alpha transparency channel. • **Gray** – Creates a grayscale image.

Example

The following example creates a new ARGB image which is 400 pixels wide and 300 pixels tall with a gray background. It draws a black oval into the image and then writes the new image to disk.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
```

```
<!--- create some colors --->
<cfset gray = myImage.getColorByName("gray") />
<cfset black = myImage.getColorByName("black") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(gray) />

<!--- create a new image --->
<cfset myImage.createImage(400, 300) />

<!--- draw an oval --->
<cfset myImage.setFill(black) />
<cfset myImage.drawOval(10, 10, 380, 280) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>newImage.png:</b><br>
<img src="/examples/newImage.png">
</p>
```

See Also

readImage() setBackgroundColor()

readFromBase64()

Description

The readFromBase64() method can decode image data from a string of base64 encoded data. Base64 is a way to describe binary data as a printable string of characters and has many uses.

Syntax

readFromBase64(data, mode)

Parameter	Required	Type	Description
Data	Yes	String	A base64 encoded string of image data.
Mode	No	String	This is the “mode” to convert an image to as it is read. Options are: • Grayscale • RGB • ARGB • BGR • Indexed • Binary

See Also

writeToBase64()

readFromBinary()

Description

The readFromBinary() method can decode image data from binary data. This maybe useful if you have, for whatever reason, binary image data in a ColdFusion which you want to read from directly, without first writing a file to disk.

One example usage would be working with image data retrieved from a database.

Syntax

readFromBinary(data)

Parameter	Required	Type	Description
Data	Yes	String	Binary image data.
Mode	No	String	This is the "mode" to convert an image to as it is read. Options are: • Grayscale • RGB • ARGB • BGR • Indexed • Binary

See Also

writeToBinary()

readImage()

Description

The readImage() method reads an image file from disk into the Image Component instance's current image being manipulated. For a list of file formats which can be read see getReadableFormats().

Syntax

readImage(path)

Parameter	Required	Type	Description
Path	Yes	String	A valid path to a supported image file.
Mode	No	String	This is the "mode" to convert an image to as it is read. Options are: • Grayscale • RGB • ARGB • BGR • Indexed • Binary
Type	No	String	This argument can be used to explicitly

			identify the type of image being read. This is useful in the case that the image doesn't have a file extension or the extension is incorrect.

Example

The following example reads a GIF file from disk and outputs it as a PNG file.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- read the source GIF image --->
<cfset myImage.readImage("d:\examples\buy-now.gif") />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\buy-now.png", "png") />

<!--- output both images --->
<p>
<b>buy-now.gif:</b><br>
<img src="/examples/buy-now.gif">
</p>

<p>
<b>buy-now.png:</b><br>
<img src="/examples/buy-now.png">
</p>
```

See Also

createImage() getReadableFormats()

readImageFromURL()

Description

The readImageFromURL() method can be used to read images at a specific URL address. For a list of file formats which can be read see getReadableFormats().

Syntax

readImageFromURL(URL)

Parameter	Required	Type	Description
URL	Yes	String	A valid URL address to a supported image file.
Mode	No	String	This is the "mode" to convert an image to as it is read. Options are: • Grayscale • RGB • ARGB • BGR • Indexed • Binary

Example

The following example reads an image from a URL and writes it to disk.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- read the image from a URL --->
<cfset myImage.readImageFromURL("http://www.alagad.com/images/logo.gif")/>

<!--- output the image as a png --->
<cfset myImage.writeImage(expandPath("mylogo.png"), "png") />
```

See Also

getReadableFormats(), writeToBrowser()

writeToBase64()

Description

The writeToBase64() method can be used to encode image data as a string of printable characters. This is useful for several applications including sending images via email, storing images in database text fields, as well as others.

Syntax

Data = writeToBase64(format, quality)

Parameter	Required	Type	Description
Format	Yes	String	The image file format to write. For allowed options see getWritableFormats(). Typically, these are: • JPG • PNG
Quality	No	Numeric	The quality setting to use when writing the image. Valid settings are from 0 to 100. This argument can only be provided with formats that support compression settings. Typically this is only JPG.

See Also

readFromBase64(), Control Image Compression Quality

writeToBinary()

Description

The writeToBinary() method is used to get the image's raw binary data. This is useful for several applications including writing images into databases.

Syntax

Data = writeToBinary(format, quality)

Parameter	Required	Type	Description
Format	Yes	String	The image file format to write. For allowed options see getWritableFormats(). Typically, these are: • JPG • PNG
Quality	No	Numeric	The quality setting to use when writing the image. Valid settings are from 0 to 100. This argument can only be provided with formats that support compression settings. Typically this is only JPG.

See Also

readFromBinary(), Control Image Compression Quality

writeToBrowser()

Description

The write to browser method will output image data directly to the browser. This is useful in many circumstances where you do not want to write a file to disk but you want to display it to the end user. One possible use for this might be a one time use image which displays difficult to read text, similar to what is used on many popular websites for authentication.

In the example given above you would probably want to access the cfm file as if it were an image. This can easily be done by referring to a CFM file which will output the image to the browser from an image tag.

For instance:

```
<img src="hardToReadImage.cfm">
```

You can also passs in URL parameters to CFM files addressed this way. Here's another possible example:

```
<img src="button.cfm?buttonText=Hello%20World">
```

In both of these examples you would need to code the appropriate logic into your CFM file and have the Image Component output via the writeToBrowser() method.

Syntax

writeToBrowser(format, quality)

Parameter	Required	Type	Description
Format	Yes	String	The image file format to write. For allowed options see getWritableFormats(). Typically, these are: • JPG • PNG
Quality	No	Numeric	The quality setting to use when writing the image. Valid settings are from 0 to 100. This argument can only be provided with formats that support compression settings. Typically this is only JPG.

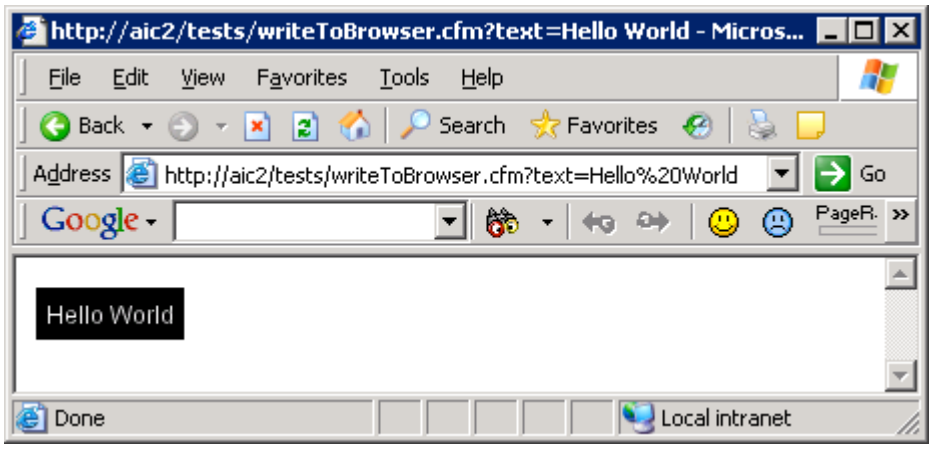
Example

This example creates an image based on the text passed into the url.text variable and draws the image directly to the browser.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- get the font metrics for the url variable 'text' --->
<cfset fontMetrics = myImage.getSimpleStringMetrics(url.text) />

<!--- create a new image --->
<cfset myImage.createImage(fontMetrics.width + 10, ¬
fontMetrics.height + 10) />

<!--- draw a string into the image --->
<cfset myImage.drawSimpleString(url.text, 5, fontMetrics.ascent + 5) />

<!--- write the image to the browser --->
<cfset myImage.writeToBrowser("png") />
```

Results

Note that the URL variable "text" is set to “Hello World”.



See Also

readImageFromURL(), Control Image Compression Quality

writeImage()

Description

The writeImage() method writes the current image to disk in the format specified by the method call.

For information on adding additional file formats you can install the Java Advanced Imaging libraries. Instructions are available here: http://www.alagad.com/index.cfm/name-jaiformats

Syntax

writeImage(path, format)

Parameter	Required	Type	Description
Path	Yes	String	A valid path to output the file to. The directory must exist. The Image Component will overwrite existing files.
Format	Yes	String	The image file format to write. For allowed options see getWritableFormats(). Typically, these are: • JPG • PNG
Quality	No	Numeric	The quality setting to use when writing the image. Valid settings are from 0 to 100. This argument can only be provided with formats that support compression settings. Typically this is only JPG.

Example

The following example reads a GIF file from disk and outputs it as a PNG file using writeImage().

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- read the source GIF image --->
<cfset myImage.readImage("d:\examples\buy-now.gif") />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\buy-now.png", "png") />

<!--- output both images --->
<p>
<b>buy-now.gif:</b><br>
<img src="/examples/buy-now.gif">
</p>
```

```
<p>
<b>buy-now.png:</b><br>
<img src="/examples/buy-now.png">
</p>
```

See Also

readImage() getWritableFormats()

getReadableFormats()

Description

The getReadableFormats() method returns a list of image formats supported for reading. Typically readable formats are GIF, JPG and PNG.

Syntax

List = getReadableFormats()

Example

The following example might output “jpeg,gif,JPG,png,jpg,JPEG”.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- output readable file formats --->
<cfoutput>
      #myImage.getReadableFormats()#
</cfoutput>
```

See Also

getWritableFormats()

getSize()

Description

The getSize() method is used to return the size an image file would be if written to disk using the options provided. The size is returned in bytes.

Syntax

Size = getSize(format, quality)

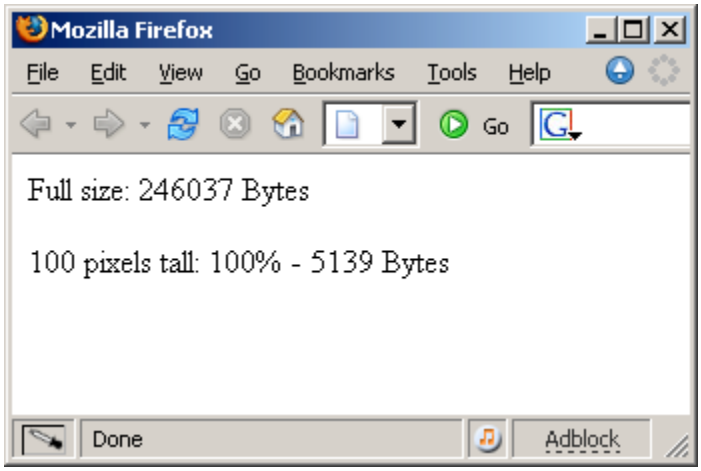
Parameter	Required	Type	Description
Format	Yes	String	The image file format to write. For allowed options see getWritableFormats(). Typically, these are: • JPG • PNG
Quality	No	Numeric	The quality setting to use when writing the image. Valid settings are from 0 to

			100. This argument can only be provided with formats that support compression settings. Typically this is only JPG.

Example

```
<cfset myImage = CreateObject("Component", "Image2.Image") />
<cfset myImage.readImage(expandPath("snow.jpg")) />

<cfoutput>
<p><b>Full size:</b>
#myImage.getSize("jpg")# Bytes</p>
</cfoutput>

<cfset myImage.scaleHeight(100) />

<cfoutput>
<p><b>100 pixels tall:</b>
100% - #myImage.getSize("jpg")# Bytes</p>
</cfoutput>
```

Results



See Also

writeImage()

getWritableFormats()

Description

The getWritableFormats() method returns a list of image formats supported for writing. Typically writable formats are JPG and PNG.

Syntax

List = getWritableFormats()

Example

The following example might output “jpeg,JPG,png,jpg,PNG,JPEG”.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- output readable file formats --->
<cfoutput>
      #myImage.getWritableFormats()#
</cfoutput>
```

See Also

getReadableFormats()

getVersion()

Description

The getVersion() method returns a structure of information on the Image Component. This includes:

- Product Name
- Version Information
- Release Date

Syntax

VersionInfo = getVersion()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
<!--- get the version info --->
<cfdump var="#myImage.getVersion()#">
```

Results

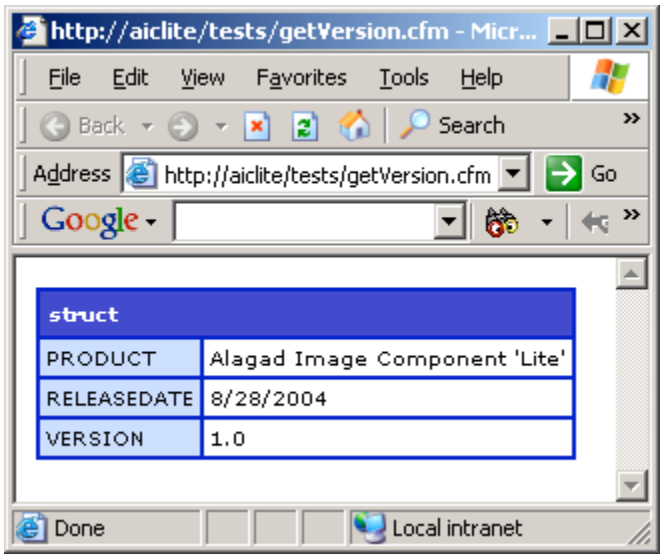


Image Size Methods

Overview

The Image Component provides several methods for changing the size of an image. You can crop a section out of an image, scale an image by percent or to a specific pixel size and resize the image canvas without resizing the image itself.

Additionally, you can use getHeight() and getWidth() to find the image's width and height.

crop()

Description

The crop() method trims the image down to a subsection of the image defined by the methods parameters. The cropping region must be inside the image. To enlarge the size of the images canvas beyond it's current bounds without resizing the image data you should use setImageSize().

Syntax

crop(x, y, width, height)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner to use for the crop.
Y	Yes	Numeric	The Y location of the upper left corner to use for the crop.
Width	Yes	Numeric	The width of the cropping area.
Height	Yes	Numeric	The height of the cropping area.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\catAndCup.jpg") />

<!--- crop out a subregion of this image --->
<cfset myImage.crop(110, 85, 150, 150) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\cropCat.jpg", "jpg") />

<!--- the old image --->
<p>
<b>catCupAndText.jpg:</b><br>
<img src="/examples/catAndCup.jpg"><br>
</p>
<!--- the new image --->
<p>
<b>cropCat.jpg:</b><br>
```

```
<img src="/examples/cropCat.jpg"><br>
</p>
```

Results

See Also

setImageSize()

scaleHeight()

Description

The scaleHeight() method resizes image data to a new height using bilinear interpolation. The height can be scaled to a specific pixel value or by a percentage of the current size. Values greater than the current height value will increase the size of the image, values less than the current height will decrease the size of the image. The values must be greater than 0.

Syntax

Parameter	Required	Type	Description
value	Yes	Numeric	The value to scale the image height.
percentOrPixels	No	String	Indicates how the image is scaled. Options are: • percent • *(default)* pixels
Proportional	No	Boolean	Indicates if the image is scaled proportionately. Defaults to true.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />


<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image Proportionately to 50 pixels tall --->
<cfset myImage.scaleHeight(100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\fatherAndSon-small.jpg", "jpg") />

<!--- the new images --->
<p>
<b>fatherAndSon.jpg:</b><br>
<img src="/examples/fatherAndSon.jpg">
</p>
<p>
<b>fatherAndSon-small.jpg:</b><br>
<img src="/examples/fatherAndSon-small.jpg">
</p>
```

Results

scalePercent()

Description

The scalePercent() method resizes image data by a percent of the current width and height using bilinear interpolation. Percent values greater than 100 will increase the size of the image. Values less than 100 will decrease the size of the image. The values must be greater than 0. To resize an image to a specific width and height, see scalePixels().

Syntax

scalePercent(widthPercent, heightPercent)

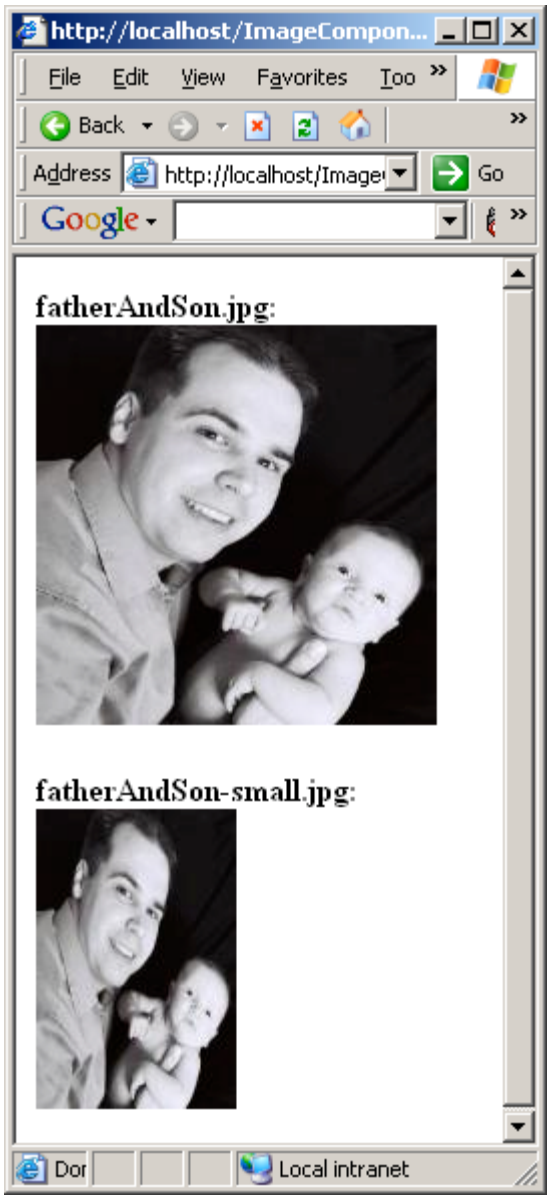
Parameter	Required	Type	Description
widthPercent	Yes	Numeric	The percentage to scale the image on the X axis.
heightPercent	Yes	Numeric	The percentage to scale the image on

			the Y axis.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image to 50% of the current width and 75% of the
current height --->
<cfset myImage.scalePercent(50, 75) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\fatherAndSon-small.jpg", "jpg") />

<!--- the new images --->
<p>
<b>fatherAndSon.jpg:</b><br>
<img src="/examples/fatherAndSon.jpg">
</p>
<p>
<b>fatherAndSon-small.jpg:</b><br>
<img src="/examples/fatherAndSon-small.jpg">
</p>
```

Results



See Also

scalePixels()

scalePixels()

Description

The scalePixels() method resizes image data to a specific pixel width and height using bilinear interpolation. Values greater than the current width or height values will increase the size of the image, values less than the current width or height will decrease the size of the image. The values must be greater than 0. To resize an image to a percentage of the image width and height, see scalePercent().

Syntax

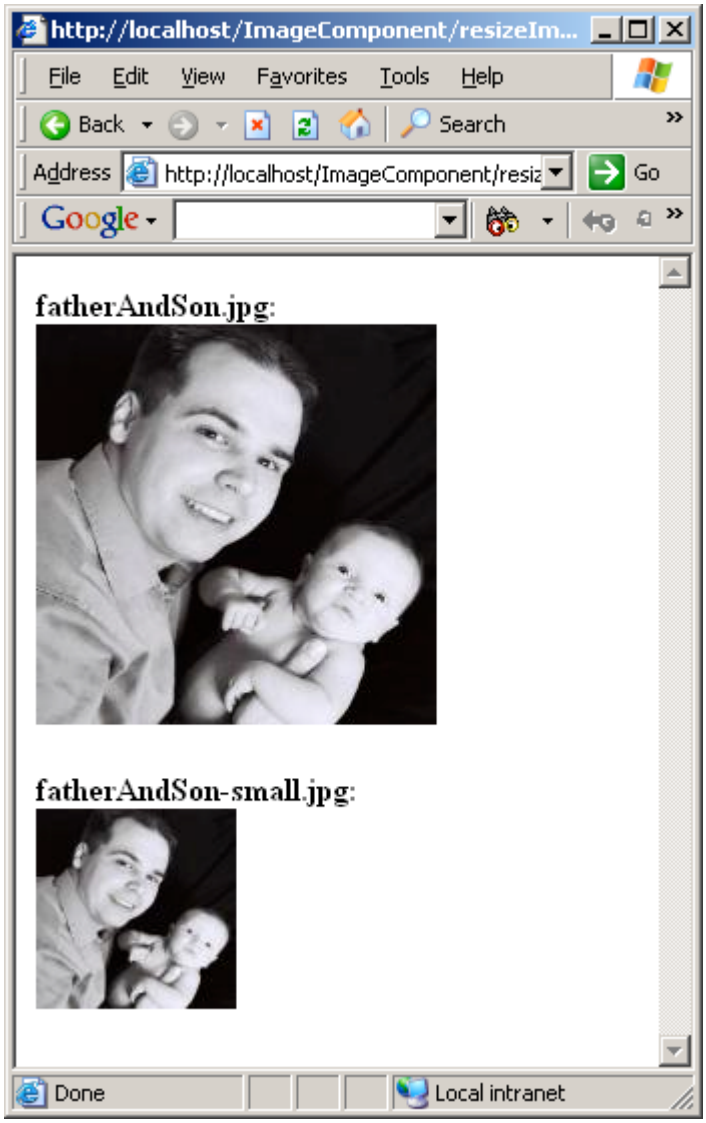
scalePixels(width, height)

Parameter	Required	Type	Description
Width	Yes	Numeric	The pixel width to scale the image on the X axis.
Height	Yes	Numeric	The pixel width to scale the image on the Y axis.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image to a specific width and height --->
<cfset myImage.scalePixels(100, 100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\fatherAndSon-small.jpg", "jpg") />

<!--- the new images --->
<p>
<b>fatherAndSon.jpg:</b><br>
<img src="/examples/fatherAndSon.jpg">
</p>
<p>
<b>fatherAndSon-small.jpg:</b><br>
<img src="/examples/fatherAndSon-small.jpg">
</p>
```

Results



See Also

scalePercent()

scaleToBox()

Description

The scaleToBox() method scales an image proportionally so that it fits within a specific width and height box defined by the parameter values. This is a convenience method which calls either scaleWidth() or scaleHeight() depending on the image proportions.

Syntax

scaleToFit(maxWidth, maxHeight)

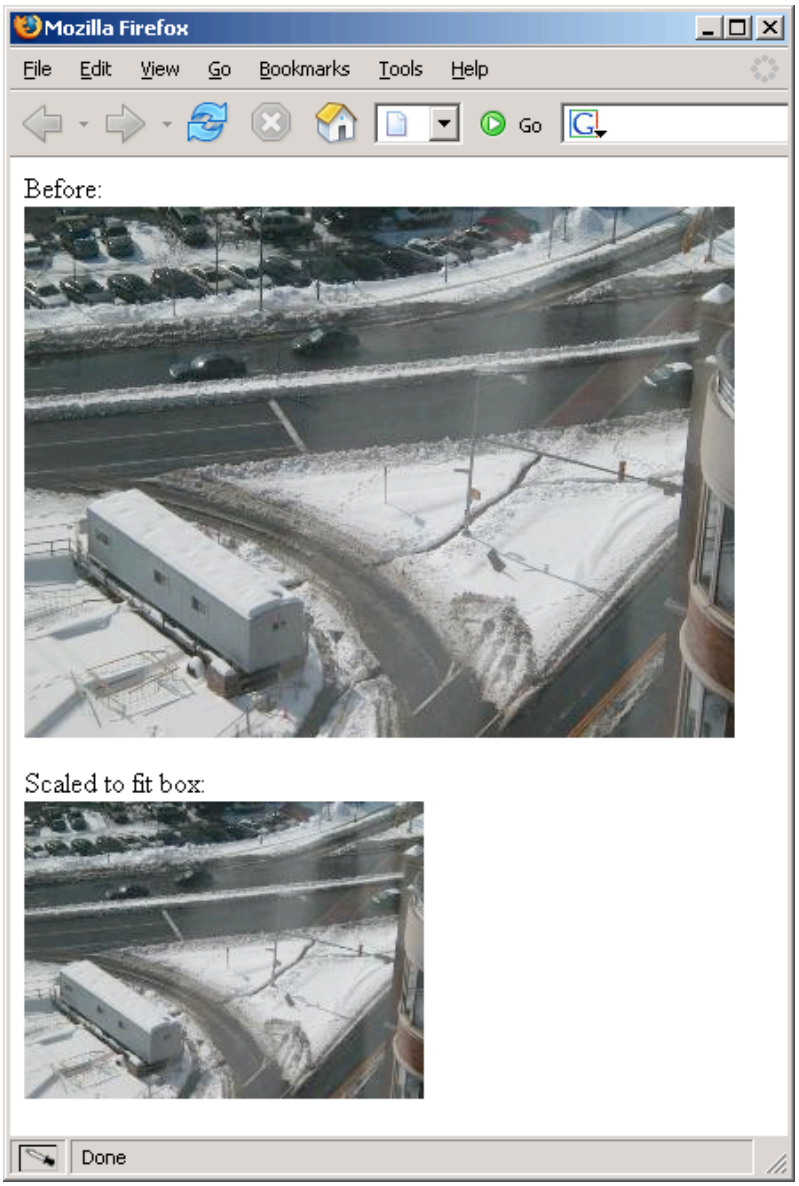
Parameter	Required	Type	Description
maxWidth	Yes	Numeric	The maximum width an image can be

			scaled to fit within.
maxHeight	Yes	Numeric	The maximum height an image can be scaled to fit within.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open image to resize --->
<cfset myImage.readImage(expandPath("snow.jpg"))>

<!--- scale the image to fit in a box --->
<cfset myImage.scaleToBox(200, 300) />

<!--- write the image --->
<cfset myImage.writeImage(expandPath("blueRay.jpg"), "jpg")>

<!--- shows the results --->
<p>Before:<br />
<img src="snow.jpg" /></p>

<p>Scaled to fit box:<br />
<img src="snow.jpg" /></p>
```

Results



scaleToFit()

Description

The scaleToFit() method scales an image so that it fits within a square defined by the parameter value. This is a convenience method which calls either scaleWidth() or scaleHeight() depending on the image proportions.

Syntax

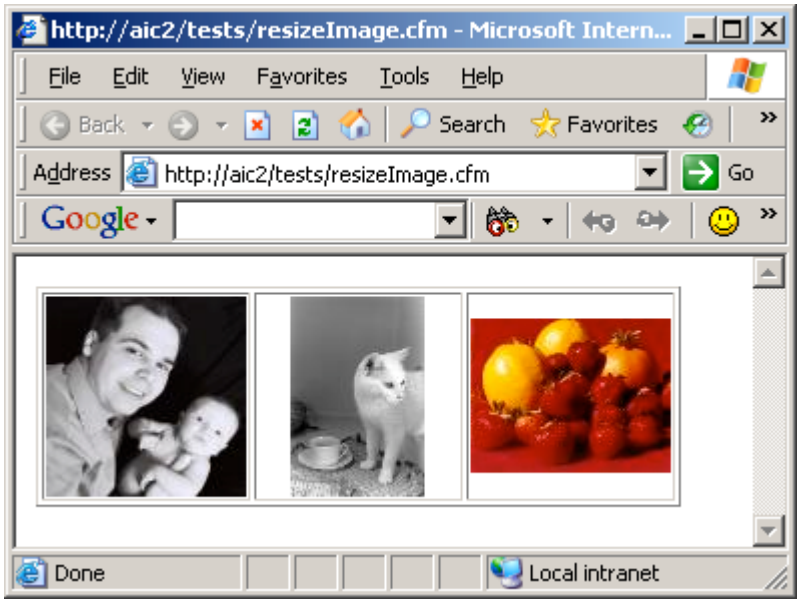
scaleToFit(value)

Parameter	Required	Type	Description
value	Yes	Numeric	The value to scale the image to fit within.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!---// open image to resize (equal width and height) //--->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image to fit a specific width and height --->
<cfset myImage.scaleToFit(100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\equalWidthHeight.jpg", "jpg") />

<!---// open image to resize (taller than wide) //--->
<cfset myImage.readImage("d:\examples\catAndCup.jpg") />

<!--- resize the image to fit a specific width and height --->
<cfset myImage.scaleToFit(100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\tallerThanWide.jpg", "jpg") />

<!---// open image to resize (wider than tall) //--->
<cfset myImage.readImage("d:\examples\strawberries.jpg") />

<!--- resize the image to fit a specific width and height --->
<cfset myImage.scaleToFit(100) />

<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\widerThanTall.jpg", "jpg") />


<!--- display the new images --->
<table border="1">
      <tr>
            <td height="100" width="100" align="center" valign="middle">
                  <img src="/examples/equalWidthHeight.jpg"></td>

            <td height="100" width="100" align="center" valign="middle">
                  <img src="/examples/tallerThanWide.jpg"></td>

            <td height="100" width="100" align="center" valign="middle">
                  <img src="/examples/widerThanTall.jpg"></td>
      </tr>
</table>
```

Results



See Also

scaleWidth(), scaleHeight()

scaleWidth()

Description

The scaleWidth() method resizes image data to a new width using bilinear interpolation. The width can be scaled to a specific pixel value or by a percentage of the current size. Values greater than the current width value will increase the size of the image, values less than the current width will decrease the size of the image. The values must be greater than 0.

Syntax

Parameter	Required	Type	Description
value	Yes	Numeric	The value to scale the image width.
percentOrPixels	No	String	Indicates how the image is scaled. Options are: • percent • *(default)* pixels
Proportional	No	Boolean	Indicates if the image is scaled proportionately. Defaults to true.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />


<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- resize the image Proportionately to 50 pixels wide --->
```

```
<cfset myImage.scaleWidth(100) />
<!--- output the image in JPG format --->
<cfset myImage.writeImage("d:\examples\fatherAndSon-small.jpg", "jpg") />

<!--- the new images --->
<p>
<b>fatherAndSon.jpg:</b><br>
<img src="/examples/fatherAndSon.jpg">
</p>
<p>
<b>fatherAndSon-small.jpg:</b><br>
<img src="/examples/fatherAndSon-small.jpg">
</p>
```

Results

setImageSize()

Description

The setImagesize() method sets the image width and height without resizing the original image data. Any portion of the new image which extends outside the bounds of the original image is set to the current background color.

Syntax

setImageSize(width, height, align)

Parameter	Required	Type	Description
Width	Yes	Numeric	The pixel width to scale the image on the X axis.
Height	Yes	Numeric	The pixel width to scale the image on the Y axis.
Align	No	String	Controls how the image data is aligned within the new image frame. Options are: • *(default)* topLeft • middleLeft • bottomLeft • topCenter • middleCenter • bottomCenter • topRight • middleRight • bottomRight

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to resize --->
<cfset myImage.readImage("d:\examples\bob.gif") />

<!--- set the image background color to cornflowerBlue --->
<cfset cornflowerBlue = myImage.getColorByName("CornflowerBlue") />
<cfset myImage.setBackgroundColor(cornflowerBlue) />

<!--- set the image canvas size to 40 pixels larger in the X and Y
axis.  Align the image in the lower right --->
<cfset myImage.setImageSize(myImage.getWidth() + 40, ¬
myImage.getHeight() + 40, "bottomRight" ) />

<!--- output the image in png format --->
<cfset myImage.writeImage("d:\examples\bobNewSize.png", "png") />

<!--- the new images --->
<p>
<b>bob.gif:</b><br>
<img src="/examples/bob.gif">
```

```
</p>
<p>
<b>bobNewSize.png:</b><br>
<img src="/examples/bobNewSize.png">
</p>
```

Results

See Also

setBackgroundColor()

trimEdges()

Description

The trimEdges() method trims areas of the same color from edges of an image. The color used for trimming is based on a specified pixel. By default all edges are trimmed, but you can easily indicate which edges to trim.

Syntax

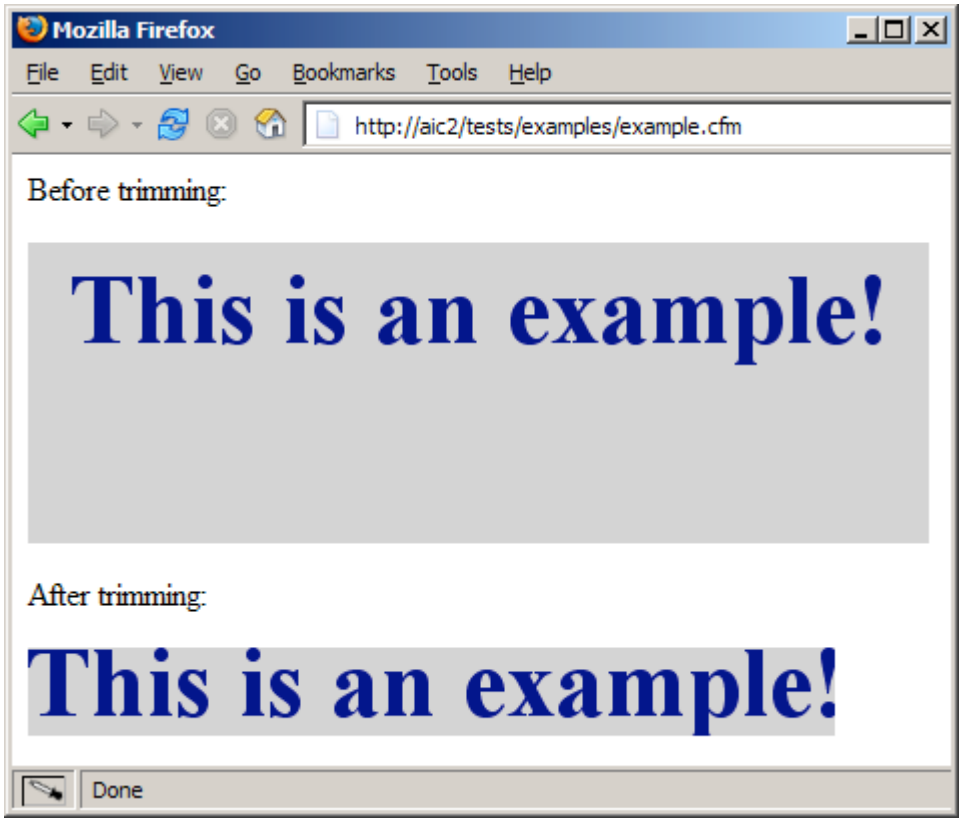
trimEdges(startingPixel, trimLeft, trimTop, trimRight, trimBottom)

Parameter	Required	Type	Description
StartingPixel	Yes	String	Indicates the pixel to base the trim on. Options are: • topLeft • topRight • bottomLeft • bottomRight
trimLeft	No	Boolean	Indicates if the left edge should be trimmed. Defaults to true.
trimTop	No	Boolean	Indicates if the top edge should be trimmed. Defaults to true.
trimRight	No	Boolean	Indicates if the right edge should be trimmed. Defaults to true.
trimBottom	No	Boolean	Indicates if the bottom edge should be trimmed. Defaults to true.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image2.Image") />

<!--- crate a new image --->
<cfset myImage.createImage(450, 150) />

<!--- create a font and color --->
<cfset times = myImage.loadSystemFont("Times New Roman", 50, "bold") />
<cfset lightGray = myImage.getColorByName("lightGray") />
<cfset darkBlue = myImage.getColorByName("darkBlue") />

<!--- fill the background --->
<cfset myImage.setBackgroundColor(lightGray) />
<cfset myImage.clearImage() />

<!--- draw text into the images --->
<cfset myImage.setFill(darkBlue) />
<cfset myImage.drawSimpleString("This is an example!", 20, 50, times)
/>

<!--- write and display the image --->
<cfset myImage.writeImage(expandPath("example1.png"), "png") />
<p>Before trimming:</p>
<img src="example1.png" />

<!--- trim the edges off the image --->
<cfset myImage.trimEdges("topLeft") />

<!--- write and display the trimmed image --->
<cfset myImage.writeImage(expandPath("example2.png"), "png") />
<p>After trimming:</p>
<img src="example2.png" />
```

Results



getHeight()

Description

The getHeight() method returns the image height in pixels.

Syntax

Numeric = getHeight()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\georgetown.jpg") />

<!--- get the width --->
<cfset width = myImage.getWidth() />
<!--- get the height --->
<cfset height = myImage.getHeight() />

<!--- output the image size --->
<p>
<b>georgetown.jpg:</b><br>
<img src="/examples/georgetown.jpg"><br>
<cfoutput>
      Width: #width#<br>
      Height: #height#
</cfoutput>
```

```
</p>
```

getWidth()

Description

The getWidth() method returns the image height in pixels.

Syntax

Numeric = getWidth()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\georgetown.jpg") />

<!--- get the width --->
<cfset width = myImage.getWidth() />
<!--- get the height --->
<cfset height = myImage.getHeight() />

<!--- output the image size --->
<p>
<b>georgetown.jpg:</b><br>
<img src="/examples/georgetown.jpg"><br>
<cfoutput>
      Width: #width#<br>
      Height: #height#
</cfoutput>
</p>
```

Manipulating Image Data

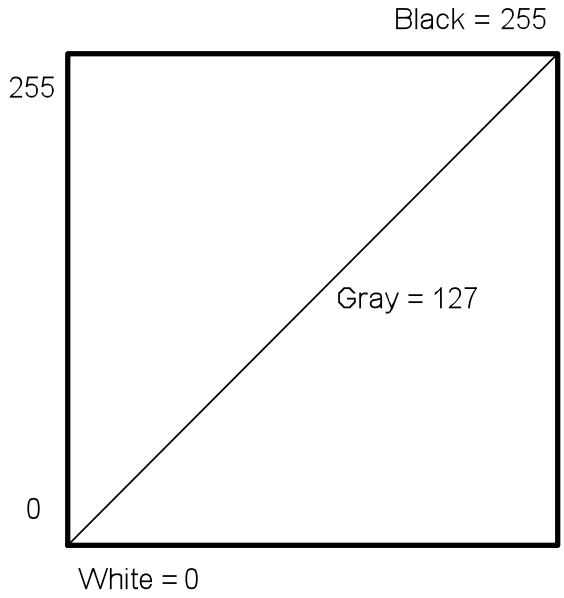
adjustLevels()

Description

The adjustLevels() method is a very versatile method for adjusting the image's white and black points. This method functions similar to adjusting an image's levels or curves in other commercial photo editing software.

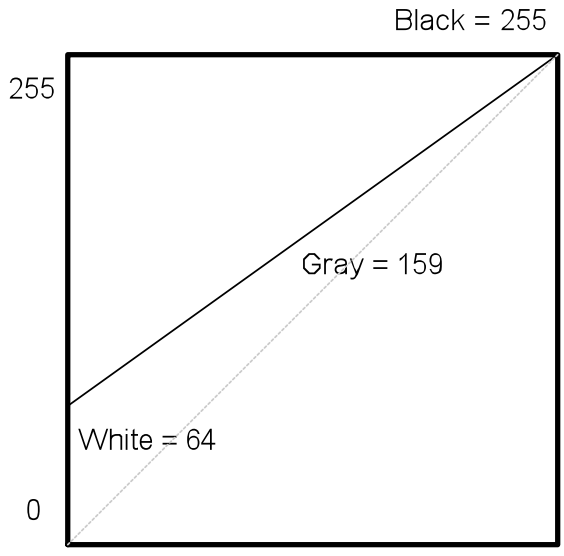
The adjustLevels() method takes two parameters, low and high, both numeric values between 0 and 255. The low parameter corresponds to white colors in the image and the high parameter corresponds to blacks. The parameters allow you to map the white and black points to other shades of gray and will interpolate the values back to black and white, adjusting the other pixels in the image.

The concept can most easily be described in a graph:



The value of white in the lower left corner is 0. The value of black in the upper right corner is 255. Gray would be 127, in the middle of the graph.

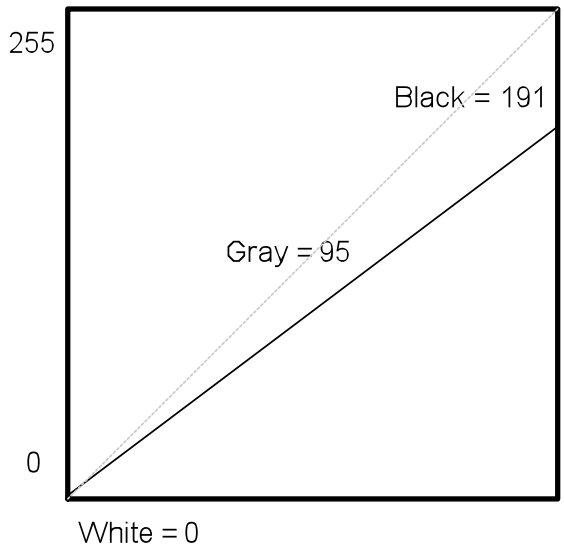
Imagine that you move the white corner from 0 to 64. The resulting graph would describe and image where light gray becomes white, dark gray becomes a lighter gray and black remains black. The new graph would look like this:



By adjusting the low end of the graph you lighten portions of the image. The graph above represents what calling the adjustLevels() method with a low parameter of 64 and a high parameter of 255 would do. For example:

```
<!--- darken the image --->
<cfset myImage.adjustLevels(64, 255) />
```

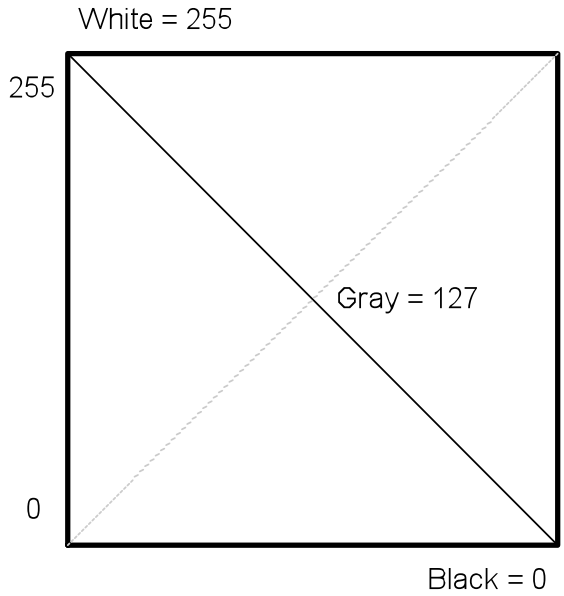
If you were to move the black corner from 255 to 191 as in the following graph you would darken the image by mapping dark gray to black, a lighter gray to gray and white would remain white.



The graph above represents what happens when you call:

```
<!--- lighten the image --->
<cfset myImage.adjustLevels(0, 191) />
```

An extreme version would be represented by the following graph which depicts an image where white has been mapped to black and black to white. The result of this graph is an inverted image.



The graph above represents what happens when you call:

```
<!--- negate the image --->
<cfset myImage.adjustLevels(255, 0) />
```

The Alagad Image Component provides a few shortcut methods for working with adjustLevels(). See lighten(), darken() and negate().

Syntax

adjustLevels(low, high)

Parameter	Required	Type	Description
Low	Yes	Numeric	A numeric value which maps the white point to another value. Accepts values from 0 to 255.
High	Yes	Numeric	A numeric value which maps the black point to another value. Accepts values from 255 to 0.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\shells.jpg") />
<!--- darken the image --->
<cfset myImage.adjustLevels(0, 191) />
<!--- output the lightened version --->
<cfset myImage.writeImage("d:\examples\shells_dark.jpg", "jpg") />

<!--- reopen the image to inspect --->
<cfset myImage.readImage("d:\examples\shells.jpg") />
<!--- lighten the image --->
<cfset myImage.adjustLevels(64, 255) />
<!--- output the lightened version --->
<cfset myImage.writeImage("d:\examples\shells_light.jpg", "jpg") />

<!--- reopen the image to inspect --->
<cfset myImage.readImage("d:\examples\shells.jpg") />
<!--- negate the image --->
<cfset myImage.adjustLevels(255, 0) />
<!--- output the negative version --->
<cfset myImage.writeImage("d:\examples\shells_negative.jpg", "jpg") />

<!--- reopen the image to inspect --->
<cfset myImage.readImage("d:\examples\shells.jpg") />
<!--- lower the image contrast --->
<cfset myImage.adjustLevels(64, 191) />
<!--- output the lowered contrast version --->
<cfset myImage.writeImage("d:\examples\shells_lowContrast.jpg", ¬
"jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/shells.jpg">
	</td>
	<td>
```

```
            <b>Darkened</b><br>
            <img src="/examples/shells_dark.jpg">
      </td>
</tr>
<tr>
      <td>
            <b>Lightened</b><br>
            <img src="/examples/shells_light.jpg">
      </td>
      <td>
            <b>Negated</b><br>
            <img src="/examples/shells_negative.jpg">
      </td>
</tr>
<tr>
      <td colspan="2">
            <b>Low Contrast</b><br>
            <img src="/examples/shells_lowContrast.jpg">
      </td>
</tr>
</table>
```

Results

See Also

lighten() darken() negate()

blur()

Description

The blur() method blurs the image by set pixel values based on the values if pixels within a particular range. Due to the algorithm used, this method introduces a border around the image the color of the background color.

Syntax

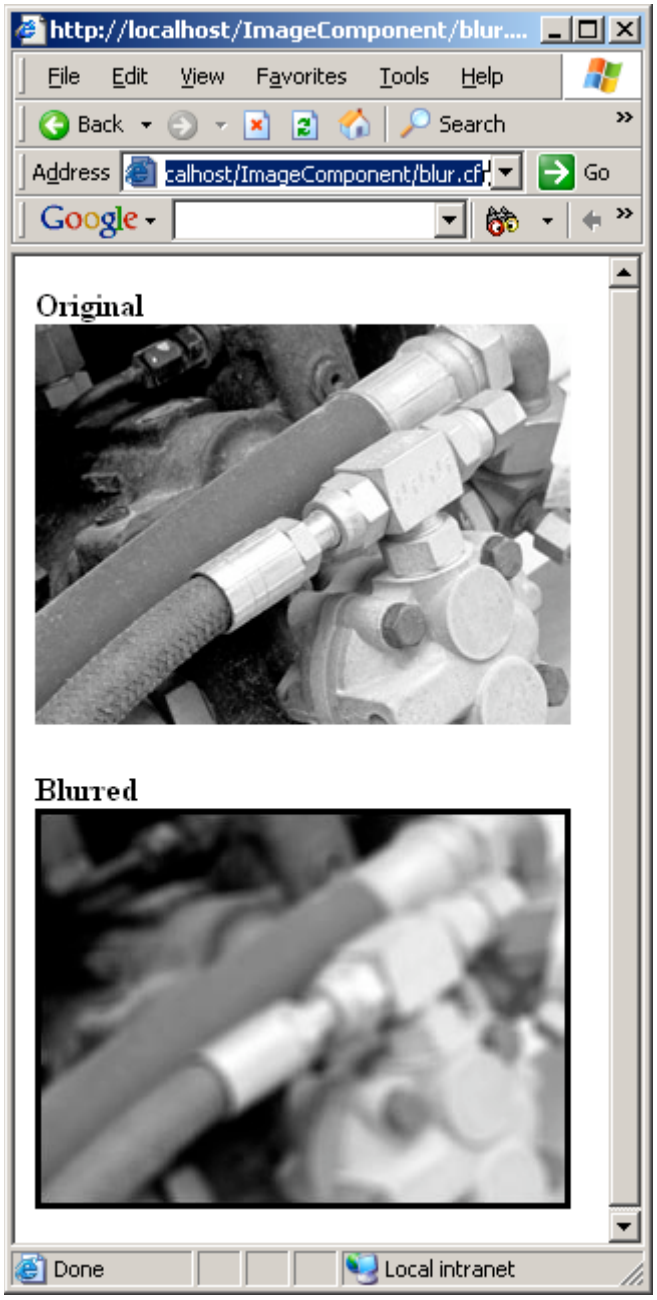
blur(size)

Parameter	Required	Type	Description
Size	Yes	Numeric	The size of the blur. In general, this is the number of surrounding pixels used when blurring the image. The higher the number, the more blurry the image.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\fuelPump.jpg") />

<!--- blur the image --->
<cfset myImage.blur(3) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\fuelPumpBlur.jpg", "jpg") />

<!--- output the images --->
<p>
<b>Original</b><br>
<img src="/examples/fuelPump.jpg">
</p>
<p>
<b>Blurred</b><br>
<img src="/examples/fuelPumpBlur.jpg">
</p>
```

Results



clearImage()

Description

The clearImage() method clears the entire area of the image and sets it uniformly to the current background color.

Syntax

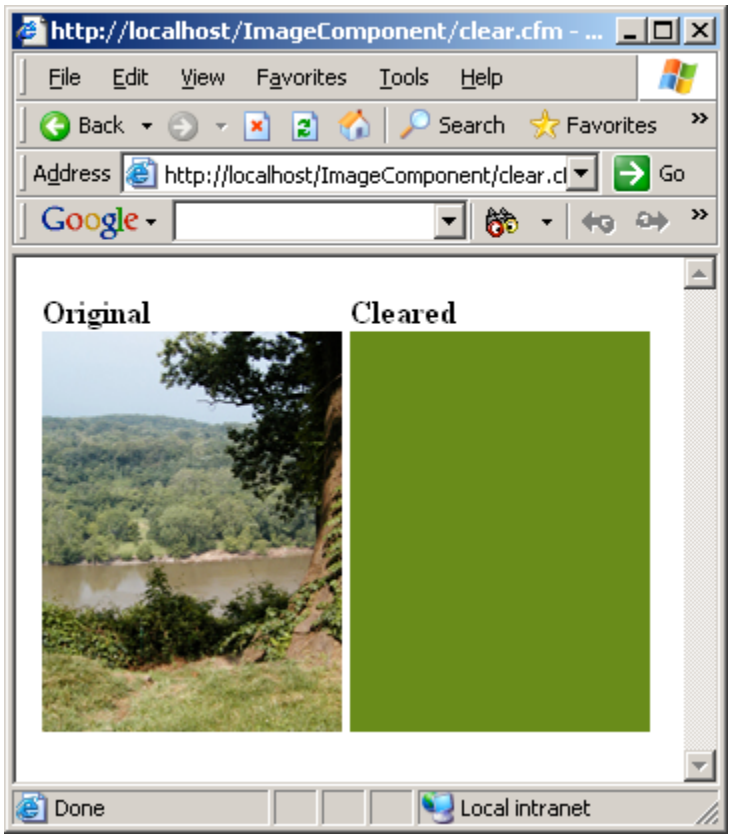
clearImage()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
```

```
<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\river.jpg") />

<!--- set the background color ----->
<cfset oliveDrab = myImage.getColorByName("OliveDrab") />
<cfset myImage.setBackgroundColor(oliveDrab) />

<!--- clear the image --->
<cfset myImage.clearImage() />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\clear.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/river.jpg">
	</td>
	<td>
		<b>Cleared</b><br>
		<img src="/examples/clear.jpg">
	</td>
</tr>
</table>
```

Results



See Also

setBackgroundColor()

clearRectangle()

Description

The clearRectangle() method clears a rectangular area of the image and sets it to the current background color.

Syntax

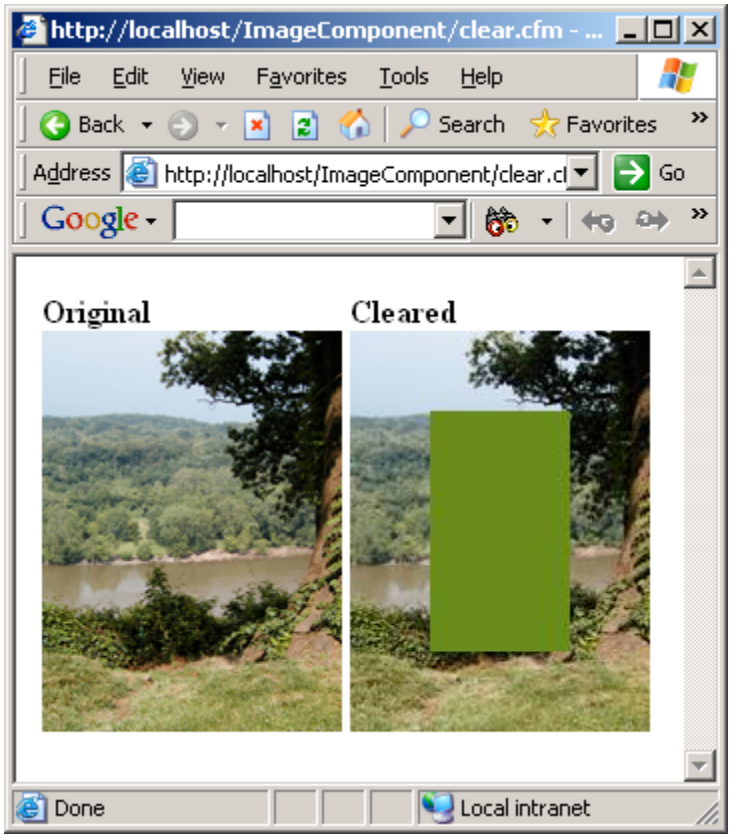
clearRectangle(x, y, width, height)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the rectangular area to clear.
Y	Yes	Numeric	The Y location of the upper left corner of the rectangular area to clear.
Width	Yes	Numeric	The width of the area to clear.
Height	Yes	Numeric	The height of the area to clear.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\river.jpg") />

<!--- set the background color ----->
<cfset oliveDrab = myImage.getColorByName("OliveDrab") />
<cfset myImage.setBackgroundColor(oliveDrab) />

<!--- clear an area --->
<cfset myImage.clearRectangle(40, 40, 70, 120) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\clear.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/river.jpg">
	</td>
	<td>
		<b>Cleared</b><br>
		<img src="/examples/clear.jpg">
	</td>
</tr>
</table>
```

Results



See Also

setBackgroundColor()

copyRectangleTo()

Description

The copyRectangleTo() method copies a rectangular area of the image to another location within the image.

Syntax

copyRectangleTo(x, y, width, height, distanceX, distanceY)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the rectangular area to copy.
Y	Yes	Numeric	The Y location of the upper left corner of the rectangular area to copy.
Width	Yes	Numeric	The width of the area to copy.
Height	Yes	Numeric	The height of the area to copy.
DistanceX	Yes	Numeric	The distance to move the copied area on the X axis.
DistanceY	Yes	Numeric	The distance to move the copied area on the Y axis.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\fatherAndSon.jpg") />

<!--- copy a region of the image and move it elsewhere ----->
<cfset myImage.copyRectangleTo(125, 100, 65, 65, -65, -65) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\fatherAndSon2.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/fatherAndSon.jpg">
	</td>
	<td>
		<b>Area Copied</b><br>
		<img src="/examples/fatherAndSon2.jpg">
	</td>
</tr>
</table>
```

Results

darken()

Description

The darken() method darkens the image by adjusting the images levels. This method is a convenience method which calls adjustLevels().

Syntax

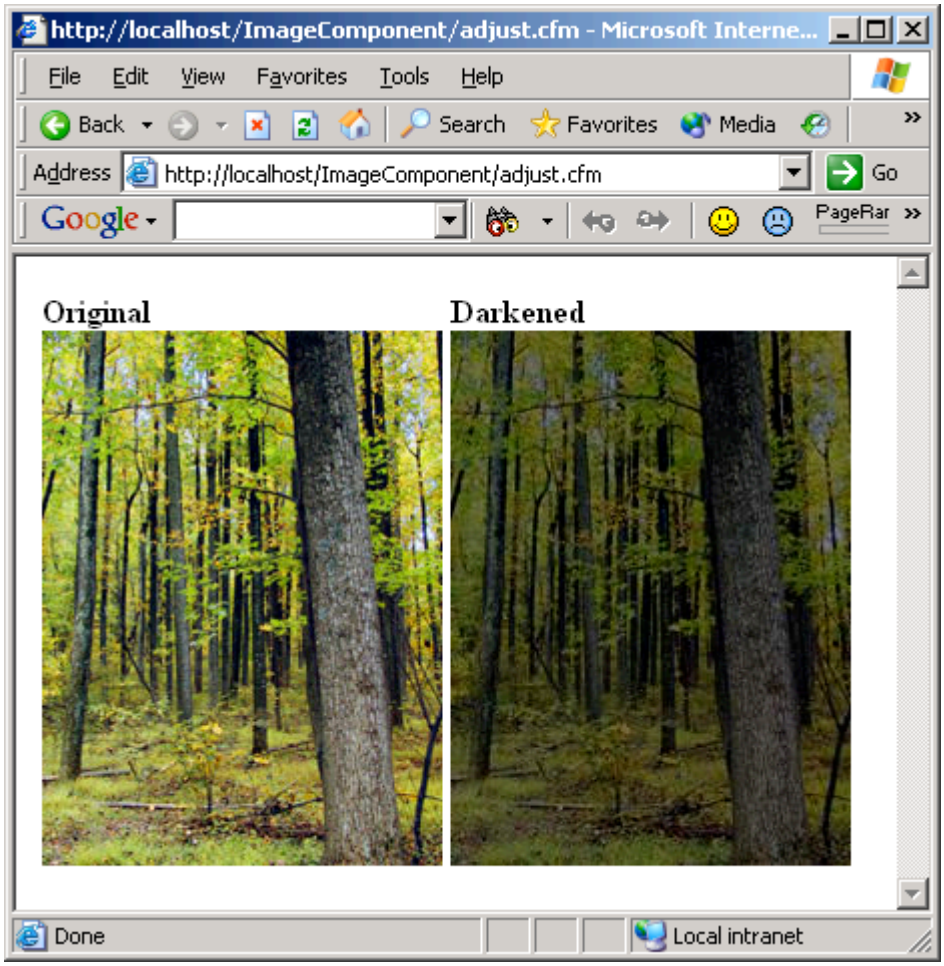
darken(percent)

Parameter	Required	Type	Description
Percent	Yes	Numeric	The percent to darken the image. This must be a value from 0 to 100.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\forest1.jpg") />

<!--- darken the image 50% ----->
<cfset myImage.darken(50) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\darkForest.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/forest1.jpg">
	</td>
	<td>
		<b>Darkened</b><br>
		<img src="/examples/darkForest.jpg">
	</td>
</tr>
</table>
```

Results



See Also

adjustLevels() lighten() negate()

emboss()

Description

The emboss() method emboss the image.

Syntax

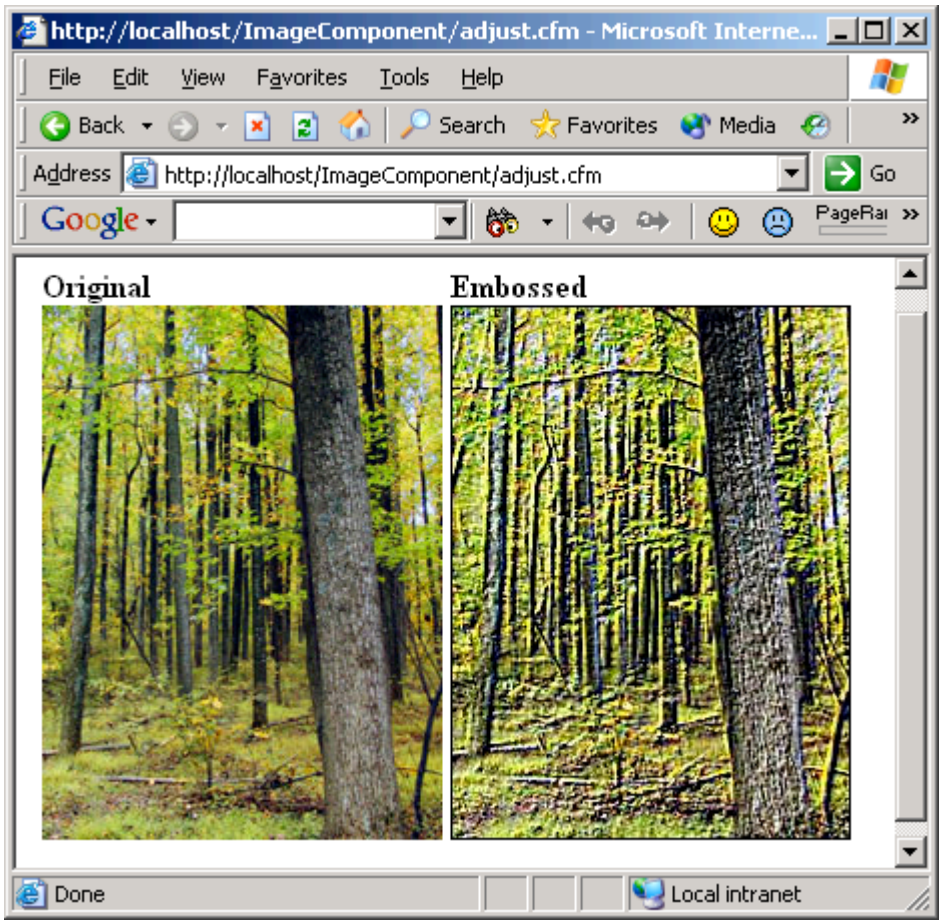
emboss(size, strength)

Parameter	Required	Type	Description
Size	Yes	Numeric	The size of the emboss. In general, this is the number of surrounding pixels used when embossing the image.
Strength	Yes	Numeric	The amount the image is embossed. The higher this value the more apparent the emboss.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\forest1.jpg") />

<!--- emboss the image ----->
<cfset myImage.emboss(1, 2) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\embossForest.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/forest1.jpg">
	</td>
	<td>
		<b>Embossed</b><br>
		<img src="/examples/embossForest.jpg">
	</td>
</tr>
</table>
```

Results



findEdges()

Description

The findEdges() method highlights areas of the image with significant changes between light and dark.

Syntax

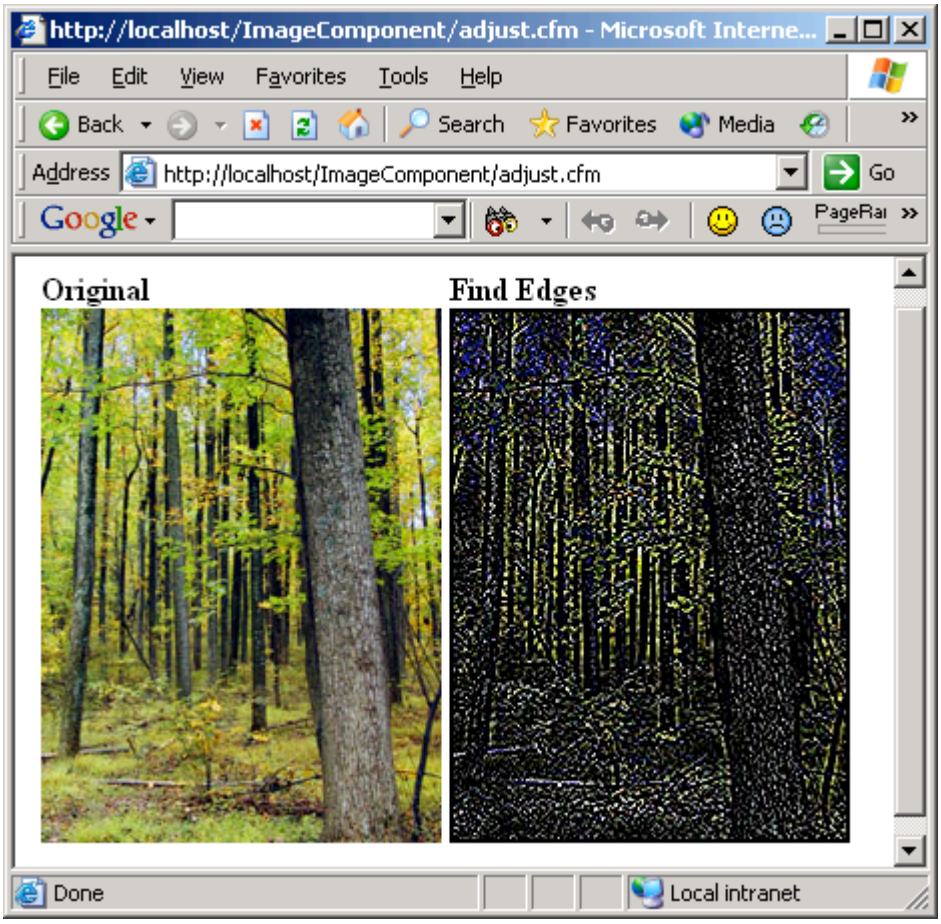
findEdges(size, strength)

Parameter	Required	Type	Description
Size	Yes	Numeric	The size of the edge. In general, this is the number of pixels wide a transition between light and dark is.
Strength	Yes	Numeric	The amount the image is affected by the findEdges() method. The higher this value the more extreme the effect.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\forest1.jpg") />

<!--- find edges ----->
<cfset myImage.findEdges(2, 5) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\forestEdges.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/forest1.jpg">
	</td>
	<td>
		<b>Find Edges</b><br>
		<img src="/examples/forestEdges.jpg">
	</td>
</tr>
</table>
```

Results



flipHorizontal()

Description

The flipHorizontal() method flips the image along it's horizontal axis.

Syntax

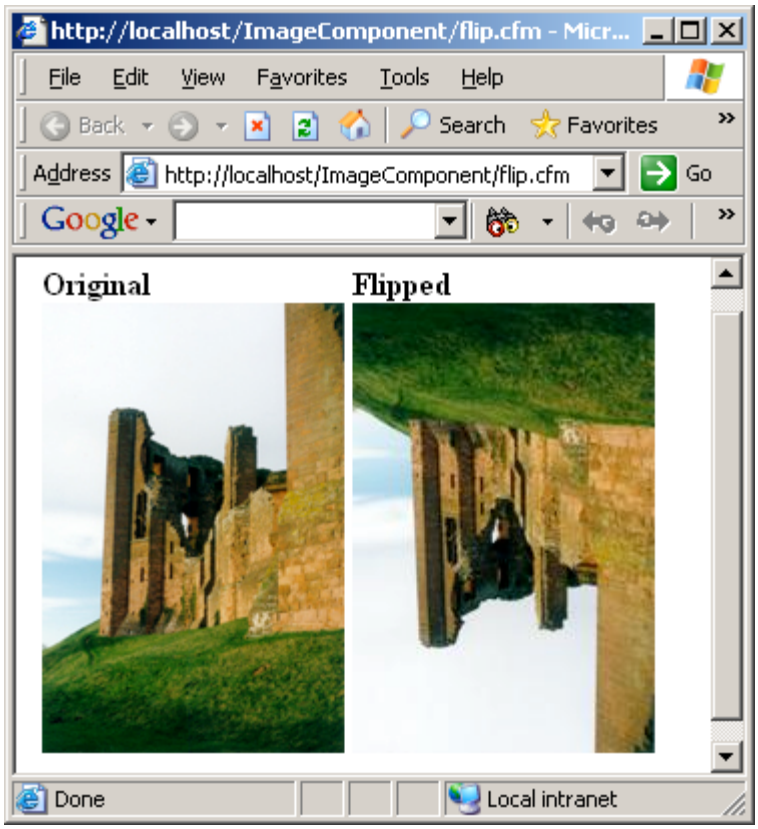
flipHorizontal()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\kennelworth3.jpg") />

<!--- flip the image ---->
<cfset myImage.flipHorizontal() />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\kennelworth3Flip.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
```

```
	<td>
		<b>Original</b><br>
		<img src="/examples/kennelworth3.jpg">
	</td>
	<td>
		<b>Flipped</b><br>
		<img src="/examples/kennelworth3Flip.jpg">
	</td>
</tr>
</table>
```

Results



See Also

flipVertical()

flipVertical()

Description

The flipVertical() method flips the image along it's vertical axis.

Syntax

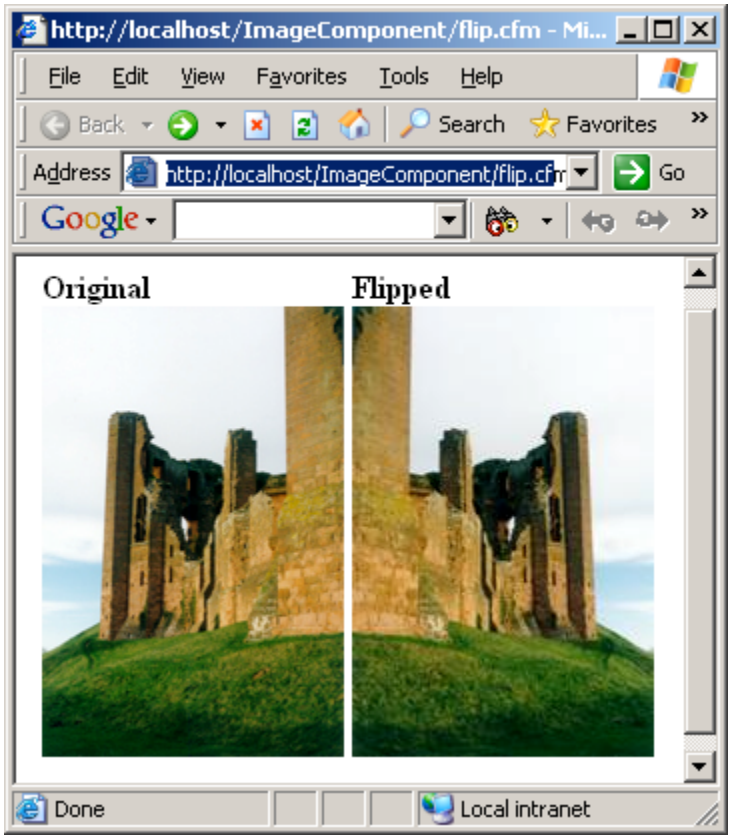
flipVertical()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
```

```
<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\kennelworth3.jpg") />

<!--- flip the image ---->
<cfset myImage.flipVertical() />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\kennelworth3Flip.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/kennelworth3.jpg">
	</td>
	<td>
		<b>Flipped</b><br>
		<img src="/examples/kennelworth3Flip.jpg">
	</td>
</tr>
</table>
```

Results



See Also

flipHorizontal()

getImageMode()

Description

The getImageMode() method returns the image mode. A mode defines the capabilities of the image. Possible options are:

- Grayscale – Indicates that the image is composed of shades of gray.
- RGB – Indicates that the image is composed of red, green and blue colors.
- ARGB – Indicates that the image is composed of red, green and blue colors and has an alpha channel.
- BGR – Indicates that the image is composed of blue, green and red colors.
- INDEXED – Indicates that the image is based on a pallet of colors. This is always the case with GIF images.
- BINARY – Indicates the color is composed only of black and white.
- Unrecognized – Indicates the Image Component doesn't recognize the image mode.

Syntax

String = getImageMode()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image") />

<!--- read an image --->
<cfset myImage.readImage(expandPath("forest1.jpg")) />

<!--- display the new image and output it's mode --->
<img src="forest1.jpg">
<cfoutput>
      <p>The mode of this image is: #myImage.getImageMode()#</p>
</cfoutput>
```

Results

See Also

setImageMode()

grayScale()

Description

The grayscale() method converts an image to a Gray Scale color mode.

Syntax

grayscale()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\strawberries.jpg") />

<!--- set to grayscale ----->
<cfset myImage.grayscale() />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\strawberriesBW.jpg", "jpg") />

<!--- output the images --->
```

```
<b>Original</b><br>
<img src="/examples/strawberries.jpg"><Br>
<b>Gray Scale</b><br>
<img src="/examples/strawberriesBW.jpg">
```

Results

lighten()

Description

The lighten() method lightens the image by adjusting the images levels. This method is a convenience method which calls adjustLevels().

Syntax

lighten(percent)

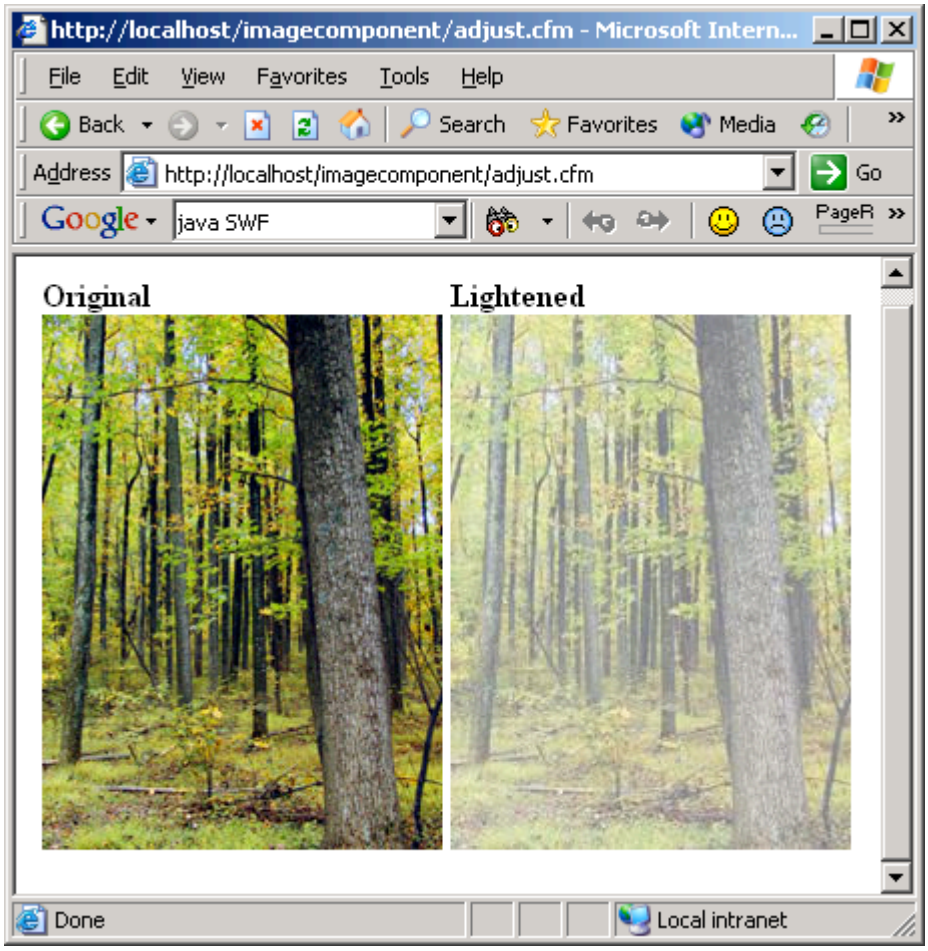
Parameter	Required	Type	Description
Percent	Yes	Numeric	The percent to lighten the image. This

			must be a value from 0 to 100.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\forest1.jpg") />

<!--- lighten the image 50% ---->
<cfset myImage.lighten(50) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\lightForest.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/forest1.jpg">
	</td>
	<td>
		<b>Lightened</b><br>
		<img src="/examples/lightForest.jpg">
	</td>
</tr>
</table>
```

Results



See Also

adjustLevels() darken() negate()

negate()

Description

The negate() method inverts all of the colors in the image similar to a photographic negative. This method is a convenience method which calls adjustLevels().

Syntax

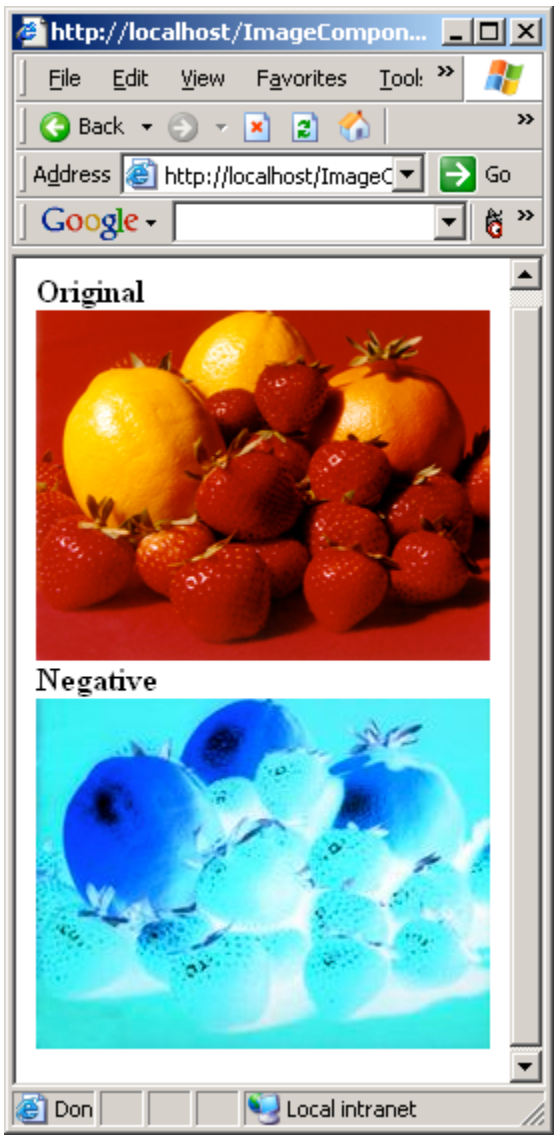
negate()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\strawberries.jpg") />

<!--- set to negative ----->
<cfset myImage.negate() />
```

```
<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\strawberriesNeg.jpg", "jpg") />

<!--- output the images --->
<b>Original</b><br>
<img src="/examples/strawberries.jpg"><Br>
<b>Negative</b><br>
<img src="/examples/strawberriesNeg.jpg">
```

Results



See Also

adjustLevels() lighten() negate()

rotate()

Description

The rotate() method rotates an image by an arbitrary angle. Any portion of the rotated image which exposes the background is set to the current background color.

Syntax

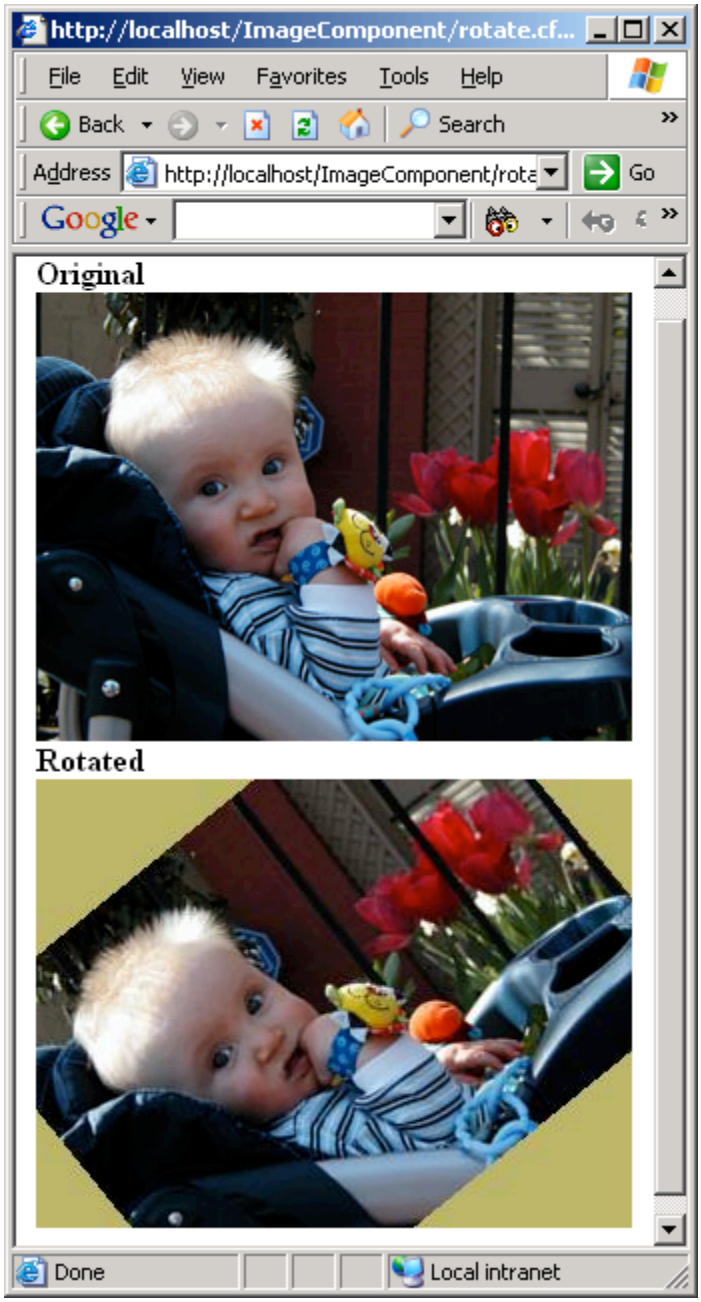
rotate(angle, resizeImage)

Parameter	Required	Type	Description
Angle	Yes	Numeric	The angle to rotate the image. Positive values rotate the image clockwise. Negative values rotate the image counterclockwise.
ResizeImage	No	Boolean	Indicates if the image should be resized to fit the rotated image. If false, the image may be cropped. If true, the image may expand to fit the entire rotated image.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\georgetown.jpg") />

<!--- set the background color --->
<cfset darkKhaki = myImage.getColorByName("DarkKhaki") />
<cfset myImage.setBackgroundColor(darkKhaki) />

<!--- rotate the image couterclockwise 39 degrees ---->
<cfset myImage.rotate(-39, false) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\georgetownRotate.jpg", "jpg") />

<!--- output the images --->
<b>Original</b><br>
<img src="/examples/georgetown.jpg"><Br>
<b>Rotated</b><br>
<img src="/examples/georgetownRotate.jpg">
```

Results



See Also

setBackgroundColor()

setImageMode()

Description

The setImageMode() method is used to change the image mode. A mode defines the capabilities of the image. Possible options are:

- Grayscale – Indicates that the image is composed of shades of gray.

- RGB – Indicates that the image is composed of red, green and blue colors.
- ARGB – Indicates that the image is composed of red, green and blue colors and has an alpha channel.
- BGR – Indicates that the image is composed of blue, green and red colors.
- INDEXED – Indicates that the image is based on a pallet of colors. This is always the case with GIF images.
- BINARY – Indicates the color is composed only of black and white.
- Unrecognized – Indicates the Image Component doesn't recognize the image mode.

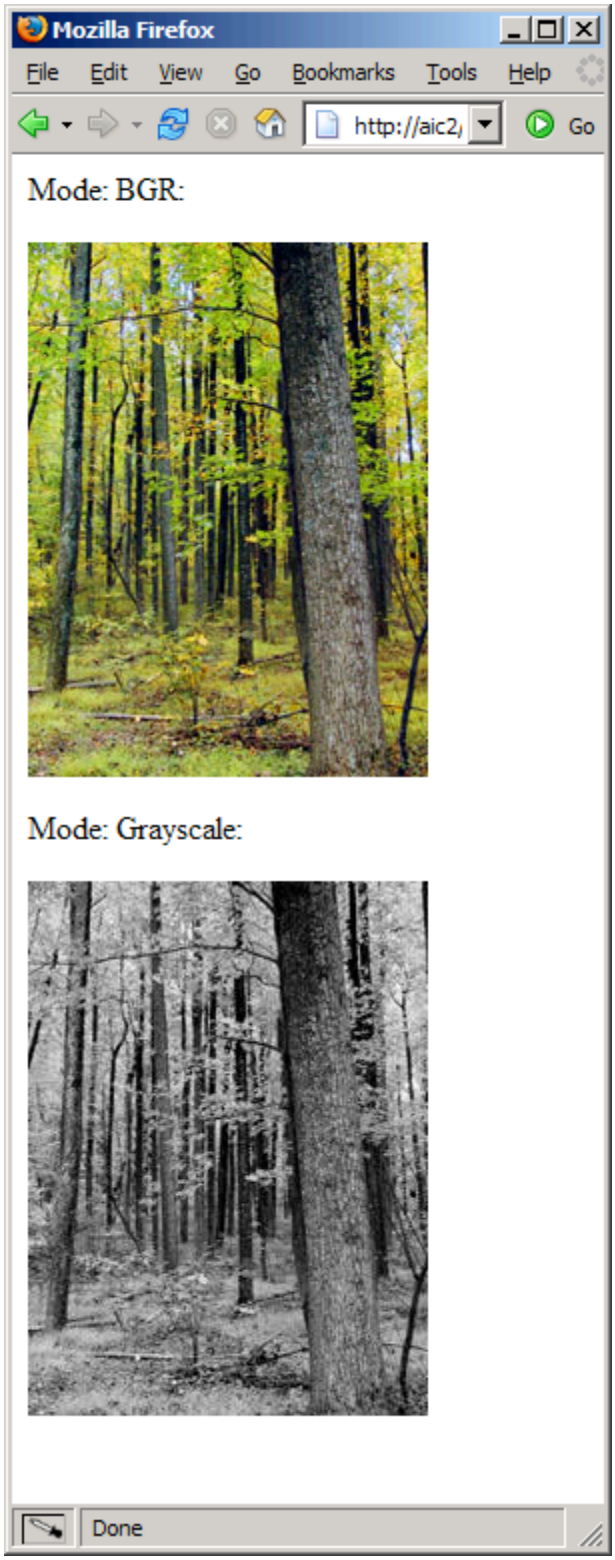
Syntax

getImageMode(Mode)

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image2.Image") />

<!--- read an image --->
<cfset myImage.readImage(expandPath("forest1.jpg")) />

<!--- display the old Image --->
<cfoutput>
<p>Mode: #myImage.getImageMode()#:</p>
</cfoutput>
<img src="forest1.jpg">

<!--- change the image mode --->
<cfset myImage.setImageMode("grayscale") />

<!--- write the new image --->
<cfset myImage.writeImage(expandPath("forest2.jpg"), "jpg") />

<!--- display the old Image --->
<cfoutput>
<p>Mode: #myImage.getImageMode()#:</p>
</cfoutput>
<img src="forest2.jpg">
```

Results



See Also

getImageMode()

sharpen()

Description

The sharpen() method applies a sharpening filter to the image.

Syntax

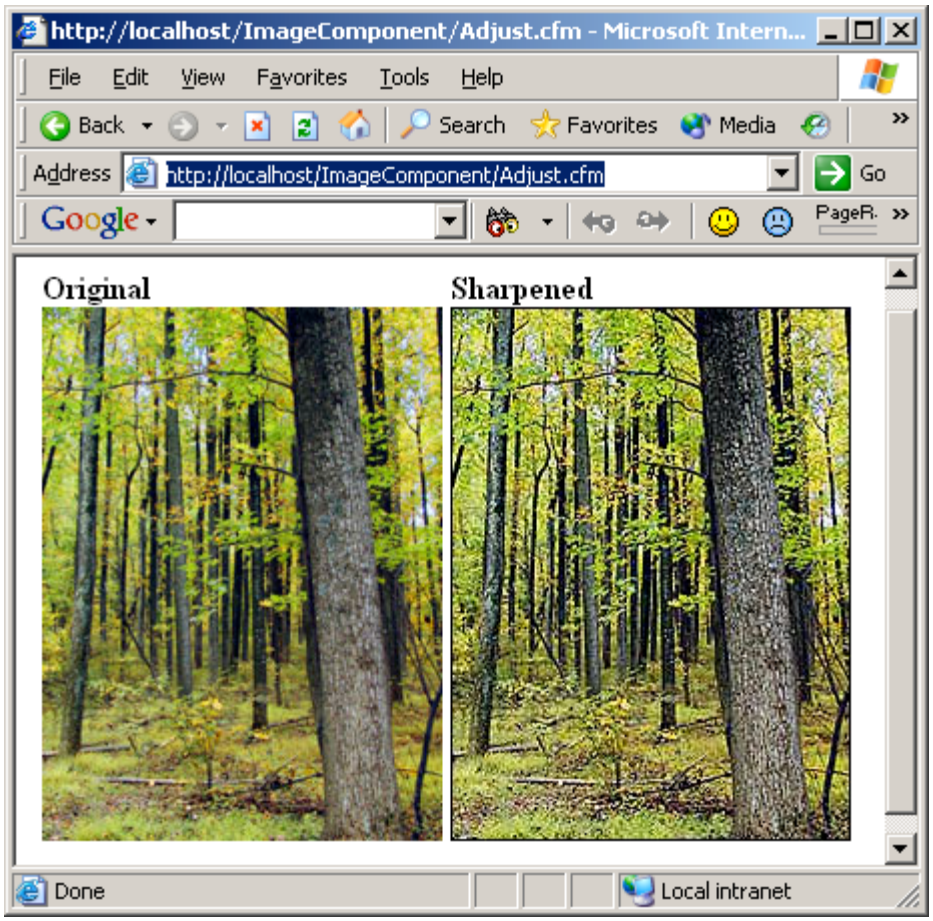
sharpen(size, strength)

Parameter	Required	Type	Description
Size	Yes	Numeric	The size of the sharpening filter. In general, this is the number of surrounding pixels used when sharpening the image.
Strength	Yes	Numeric	The amount the image is sharpened. The higher this value the more apparent the sharpening filter.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\forest1.jpg") />

<!--- sharpen the image ---->
<cfset myImage.sharpen(1, 2) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\SharpForest.jpg", "jpg") />

<!--- output the images --->
<table>
<tr>
	<td>
		<b>Original</b><br>
		<img src="/examples/forest1.jpg">
	</td>
	<td>
		<b>Sharpened</b><br>
		<img src="/examples/SharpForest.jpg">
	</td>
</tr>
</table>
```

Results



Drawing on Images

Overview

The Alagad Image Component provides a number of methods for drawing shapes, lines and other images into your image. Each of these methods is affected by various settings, in particular setFill() and setStroke(). For more information on methods which change the results of these drawing methods, see the Component Properties section.

An example which draws several shapes into an image with different lines and fills can be seen in the Creating and Drawing into a New Image example.

For information on drawing polygons see Drawing Polygons on Images. For information on drawing text in to your image see Drawing Text on Images.

drawArc()

Description

The drawArc() method draws an arc into the image.

Syntax

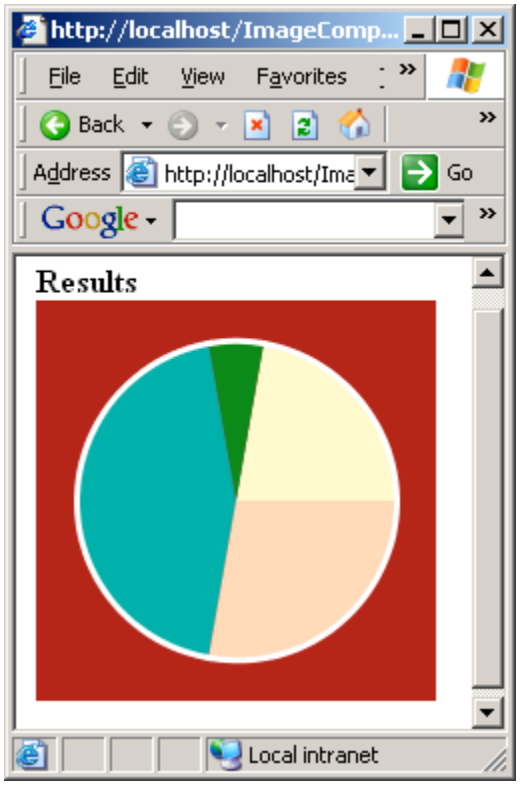
drawArc(x, y, width, height, startAngle, endAngle)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the arc to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the arc to draw.
Width	Yes	Numeric	The width of the arc to draw.
Height	Yes	Numeric	The height of the arc to draw.
StartAngle	Yes	Numeric	The angle at which to start the arc.
EndAngle	Yes	Numeric	The angle at which to end the arc.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- get some colors --->
<cfset fireBrick = myImage.getColorByName("FireBrick") />
<cfset lemonChiffon = myImage.getColorByName("LemonChiffon") />
<cfset forestGreen = myImage.getColorByName("ForestGreen") />
<cfset lightSeaGreen = myImage.getColorByName("LightSeaGreen") />
<cfset peachPuff = myImage.getColorByName("PeachPuff") />

<!--- set the background color ----->
<cfset myImage.setBackgroundColor(fireBrick) />

<!--- create a new image --->
<cfset myImage.createImage(200, 200, "argb") />

<!--- set the stroke for all of the following shapes --->
<cfset myImage.setStroke(3) />

<!--- draw some arcs with various fills --->
<cfset myImage.setFill(lemonChiffon) />
<cfset myImage.drawArc(20, 20, 160, 160, 0, 80) />

<cfset myImage.setFill(forestGreen) />
<cfset myImage.drawArc(20, 20, 160, 160, 80, 20) />

<cfset myImage.setFill(lightSeaGreen) />
<cfset myImage.drawArc(20, 20, 160, 160, 100, 160) />

<cfset myImage.setFill(peachPuff) />
<cfset myImage.drawArc(20, 20, 160, 160, 260, 100) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\shape.png", "png") />

<!--- output the images --->
<b>Result</b><br>
<img src="/examples/shape.png">
```

Results



See Also

setFill() setStroke()

drawImage()

Description

The drawImage() method draws a new image into your image. If the new image has an alpha channel it will be used to composite the two images. You can change how the two image's alpha channels interact using setComposite().

Syntax

drawImage(path, x, y, width, height)

Parameter	Required	Type	Description
Path	Yes	String	The path to the image to draw into your image.
X	Yes	Numeric	The X location of the upper left corner of the image to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the image to draw.
Width	No	Numeric	The width of the image to draw. Defaults to the width of the image being drawn.
Height	No	Numeric	The height of the image to draw. Defaults to the height of the image

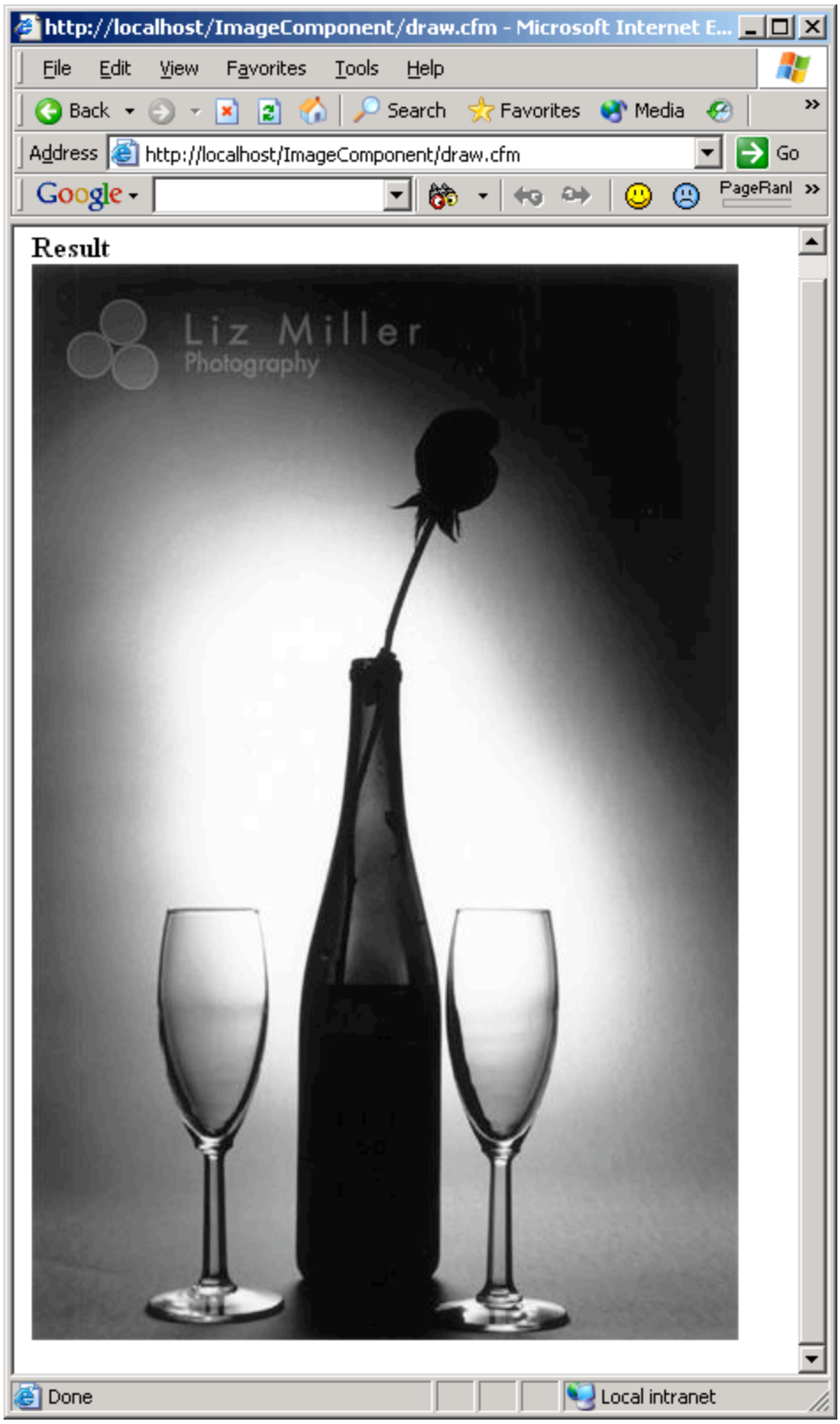
			being drawn.

Example

The following example draws a company logo into the upper left corner of an image. The logo has an alpha channel which is used to composite the two images so that the image shows through the transparent portions of the logo. Additionally, the logo is made to be slightly transparent.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open a new image --->
<cfset myImage.readImage("d:\examples\wineRose.jpg") />

<!--- set the transparency used when drawing into the image --->
<cfset myImage.setTransparency(25) />

<!--- draw an image into the image --->
<cfset myImage.drawImage("d:\examples\logo.png", 20, 20) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\wineRoseLogo.jpg", "jpg") />

<!--- output the images --->
<b>Result</b><br>
<img src="/examples/wineRoseLogo.jpg">
```

Results



See Also

setFill() setStroke() setTransparency()

drawLine()

Description

The drawLine() method draws a line into the image.

Syntax

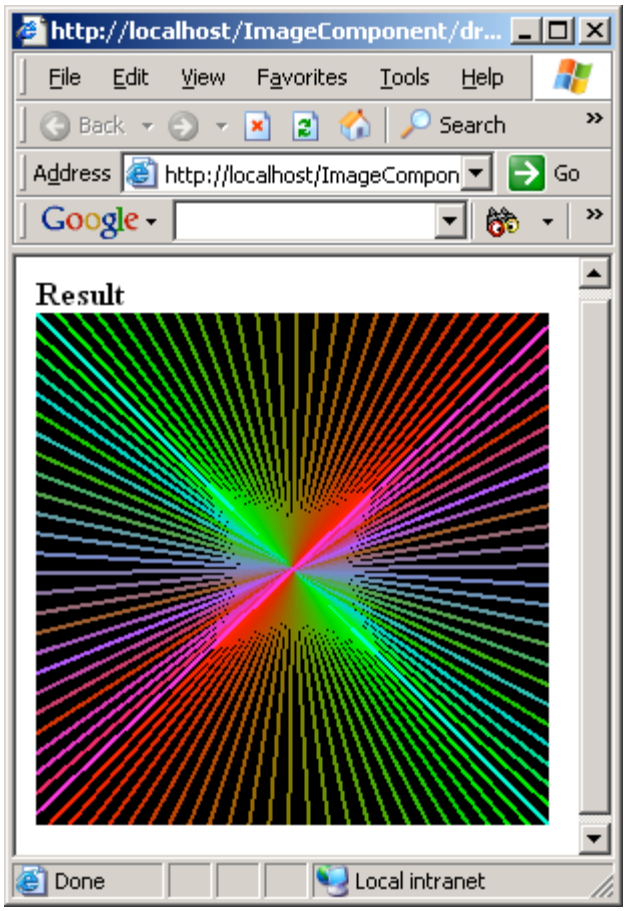
drawLine(x1, y1, x2, y2)

Parameter	Required	Type	Description
X1	Yes	Numeric	The X value of the first point of the line.
Y1	Yes	Numeric	The Y value of the first point of the line.
X2	Yes	Numeric	The X value of the second point of the line.
Y2	Yes	Numeric	The Y value of the second point of the line.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create a new image --->
<cfset myImage.createImage(256, 256, "rgb") />

<!--- loop from 0 to 300 and draw various lines --->
<cfloop from="0" to="255" index="x" step="10">
      <!--- create a color based on the value of x --->
      <cfset myImage.setStroke(2, myImage.createColor(x, 255 - x, 0)) />
      <!--- draw a line --->
      <cfset myImage.drawLine(0 + x, 0, 256 - x, 256) />
      <!--- create a random color based on the value of x --->
      <cfset myImage.setStroke(2, myImage.createColor(x, 255 - x, ¬
randrange(0, 255))) />
      <!--- draw a line --->
      <cfset myImage.drawLine(0, 0 + x, 256, 256 - x) />
</cfloop>

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\lines.png", "png") />

<!--- output the images --->
<b>Result</b><br>
<img src="/examples/lines.png">
```

Results



See Also

setStroke()

drawOval()

Description

The drawOval() method draws an oval into the image.

Syntax

drawOval(x, y, width, height)

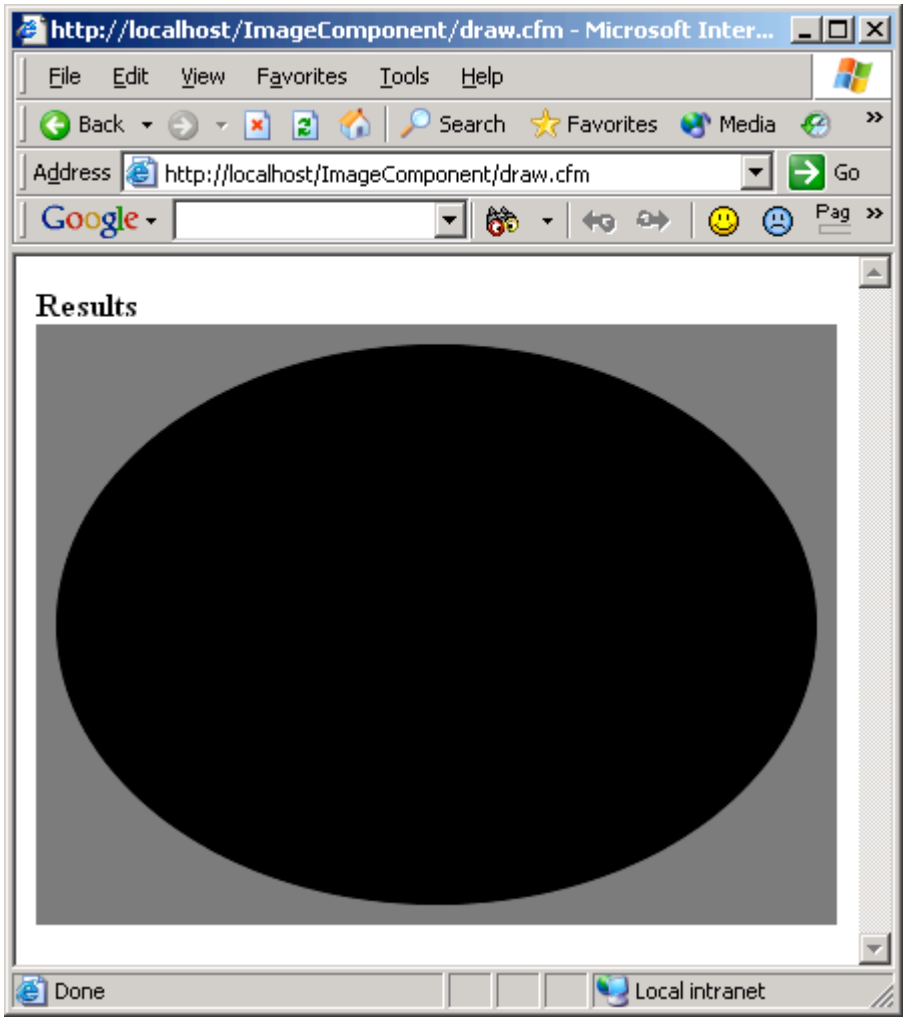
Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the oval to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the oval to draw.
Width	Yes	Numeric	The width of the oval to draw.
Height	Yes	Numeric	The height of the oval to draw.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />
```

```
<!--- create some colors --->
<cfset gray = myImage.getColorByName("gray") />
<cfset black = myImage.getColorByName("black") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(gray) />

<!--- create a new image --->
<cfset myImage.createImage(400, 300) />

<!--- draw an oval --->
<cfset myImage.setFill(black) />
<cfset myImage.drawOval(10, 10, 380, 280) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>Results</b><br>
<img src="/examples/newImage.png">
</p>
```

Results



See Also
setFill() setStroke()

drawRectangle()

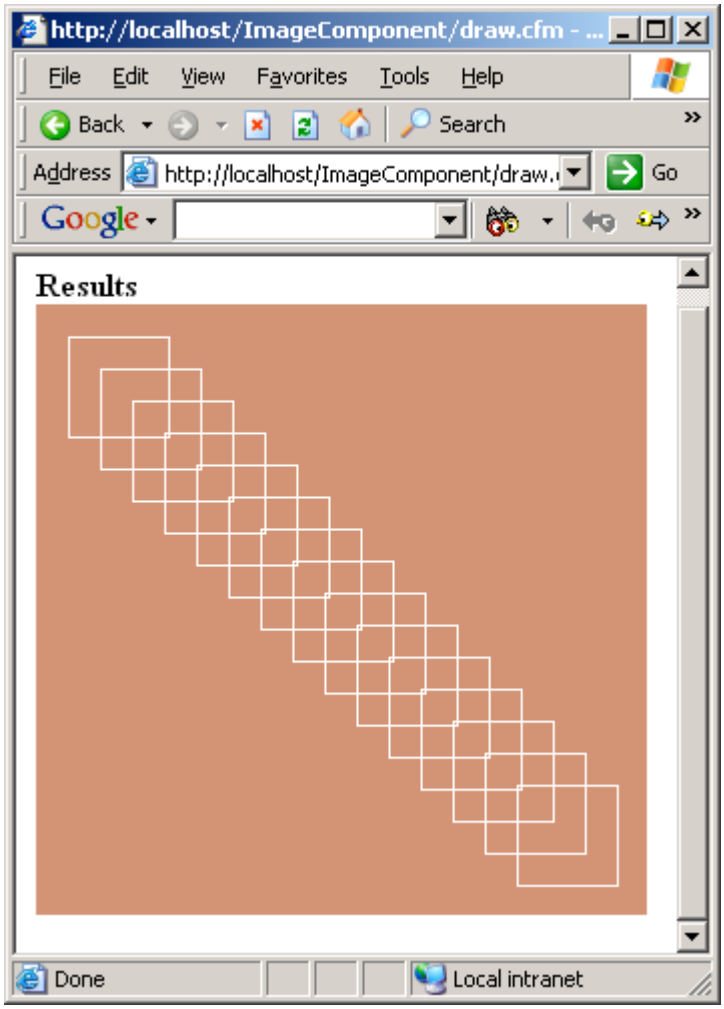
Description
The drawRectangle() method draws a rectangle into the image.

Syntax
drawRectangle(x, y, width, height)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the rectangle to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the rectangle to draw.
Width	Yes	Numeric	The width of the oval to draw.
Height	Yes	Numeric	The height of the oval to draw.

Example
```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset feldspar = myImage.getColorByName("feldspar") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(feldspar) />

<!--- create a new image --->
<cfset myImage.createImage(305, 305) />

<!--- draw a series of rectangles --->
<cfloop from="16" to="255" index="x" step="16">
      <cfset myImage.drawRectangle(x, x, 50, 50) />
</cfloop>

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>Results</b><br>
<img src="/examples/newImage.png">
</p>
```

Results



See Also

setFill() setStroke()

drawRoundRectangle()

Description

The drawRoundRectangle() method draws a rectangle with rounded corners into the image.

Syntax

drawRectangle(x, y, width, height, arcWidth, arcHeight)

Parameter	Required	Type	Description
X	Yes	Numeric	The X location of the upper left corner of the rectangle to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the rectangle to draw.
Width	Yes	Numeric	The width of the oval to draw.
Height	Yes	Numeric	The height of the oval to draw.

ArcWidth	Yes	Numeric	This is the width of the arcs at the corners of the rounded rectangle.
ArcHeight	Yes	Numeric	This is the height of the arcs at the corners of the rounded rectangle.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset aliceBlue = myImage.getColorByName("AliceBlue") />
<cfset deepPink = myImage.getColorByName("DeepPink") />
<cfset gold = myImage.getColorByName("Gold") />
<cfset indigo = myImage.getColorByName("Indigo") />
<cfset lawnGreen = myImage.getColorByName("LawnGreen") />
<cfset lightSalmon = myImage.getColorByName("LightSalmon") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(aliceBlue) />

<!--- create a new image --->
<cfset myImage.createImage(200, 200) />

<!--- draw four rounded rectanges into the new image --->
<!--- round rectangle 1 --->
<cfset myImage.setFill(deepPink) />
<cfset myImage.setStroke(2, gold) />
<cfset myImage.drawRoundRectangle(10, 10, 80, 80, 50, 50) />

<!--- round rectangle 2 --->
<cfset myImage.setFill(gold) />
<cfset myImage.setStroke(2, indigo) />
<cfset myImage.drawRoundRectangle(110, 10, 80, 80, 50, 30) />

<!--- round rectangle 3 --->
<cfset myImage.setFill(indigo) />
<cfset myImage.setStroke(2, lawnGreen) />
<cfset myImage.drawRoundRectangle(10, 110, 80, 80, 30, 50) />

<!--- round rectangle 4 --->
<cfset myImage.setFill(lawnGreen) />
<cfset myImage.setStroke(2, lightSalmon) />
<cfset myImage.drawRoundRectangle(110, 110, 80, 80, 30, 30) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>newImage.png:</b><br>
<img src="/examples/newImage.png">
</p>
```

Results

See Also

setFill() setStroke()

Drawing Polygons on Images

Overview

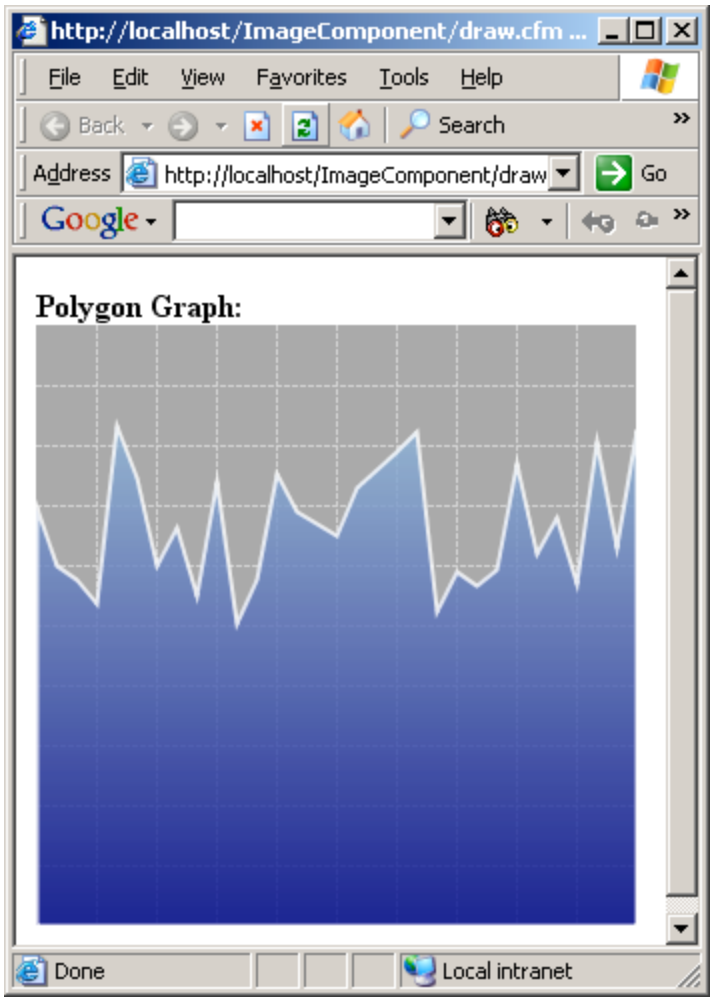
The Alagad Image Component provides a mechanism for drawing polygons into your image. Unlike other drawing methods where the entire shape is composed and drawn with one method call, to draw a polygon you must combine a series of method calls to create a polygon, add points to your polygon and then to draw the polygon.

Example

```
<!---
This example creates a simple graph of the following
list of random data.
--->
<cfset data = "" />
<cfloop from="0" to="300" index="x" step="10">
      <cfset data = ListAppend(data, RandRange(50, 150)) />
</cfloop>

<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset darkGray = myImage.getColorByName("DarkGray") />
```

```
<cfset lightblue = myImage.getColorByName("lightblue") />
<cfset darkBlue = myImage.getColorByName("DarkBlue") />
<cfset white = myImage.getColorByName("White") />
<cfset lightGray = myImage.getColorByName("LightGray") />

<!--- set the image's background color --->
<cfset myImage.setBackgroundColor(darkGray) />

<!--- create a new image --->
<cfset myImage.createImage(300, 300) />

<!--- set the fill and transparency --->
<cfset myImage.setFill(myImage.createGradient(0,0, 0, 300, lightblue, ¬
darkblue)) />
<cfset myImage.setTransparency(80) />

<!--- create a new polygon --->
<cfset dataPolygon = myImage.createPolygon() />

<!--- add horizontal grid lines to the graph --->
<cfset myImage.setStroke(1, lightGray, "square", "bevel", 0, 2) />
<cfloop from="30" to="300" index="x" step="30">
      <cfset myImage.drawLine(0, x, 300, x) />
</cfloop>
<!--- add vertical grid lines to the graph --->
<cfloop from="30" to="300" index="x" step="30">
      <cfset myImage.drawLine(x, 0, x, 300) />
</cfloop>

<!--- manually set the starting point --->
<cfset myImage.addPolygonPoint(dataPolygon, 0, 300) />

<!--- loop over the data and create a chart --->
<cfset myImage.setStroke(2, white) />
<cfset x = 0 />
<cfloop list="#data#" index="y">
      <!--- add this point to the polygon --->
      <cfset myImage.addPolygonPoint(dataPolygon, x, y) />
      <cfset x = x + 10 />
</cfloop>

<!--- manually set the ending point --->
<cfset myImage.addPolygonPoint(dataPolygon, 300, 300) />

<!--- draw the polygon --->
<cfset myImage.drawPolygon(dataPolygon) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>Polygon Graph:</b><br>
<img src="/examples/newImage.png">
</p>
```

Results



addPolygonPoint()

Description

The addPolygonPoint() method adds a point to a polygon created by the createPolygon() method.

Syntax

addPolygonPoint(Polygon, x, y)

Parameter	Required	Type	Description
Polygon	Yes	Polygon	The Polygon object to add points to.
X	Yes	Numeric	The X location of the upper left corner of the rectangle to draw.
Y	Yes	Numeric	The Y location of the upper left corner of the rectangle to draw.

See Also

Polygon Overview createPolygon() drawPolygon()

createPolygon()

Description

The createPolygon() method creates and returns a new polygon object. The new polygon has no points. Call the addPolygonPoints() method and pass in your polygon object to add points to the polygon. Once you have added points to your polygon it can be draws by calling drawPolygon().

Syntax

Polygon = createPolygon()

See Also

Polygon Overview addPolygonPoint() drawPolygon()

drawPolygon()

Description

The drawPolygon() method is used to draw polygons created by the createPolygon() method. Polygons are automatically closed by drawing a line from the last point back to the first point.

If you attempt to draw a polygon with no points you will receive an error message. You must add at least one point to the polygon with the addPolygonPoint() method.

Polygons with only one point can be drawn but do not appear. The reason is that one point can not be displayed because it takes up no space.

Polygons with two points can be drawn but appear as a line. Be aware that a line is drawn from point one to point two then another line is drawn from point two back to point one.

Syntax

drawPolygon()

Parameter	Required	Type	Description
Polygon	Yes	Polygon	The Polygon object to draw.

See Also

Polygon Overview addPolygonPoint() createPolygon()

Drawing Paths on Images

Overview

The Alagad Image Component provides a mechanism for drawing complex paths into your image. Unlike other drawing methods where the entire shape is composed and drawn with one method call, to draw a path you must combine a

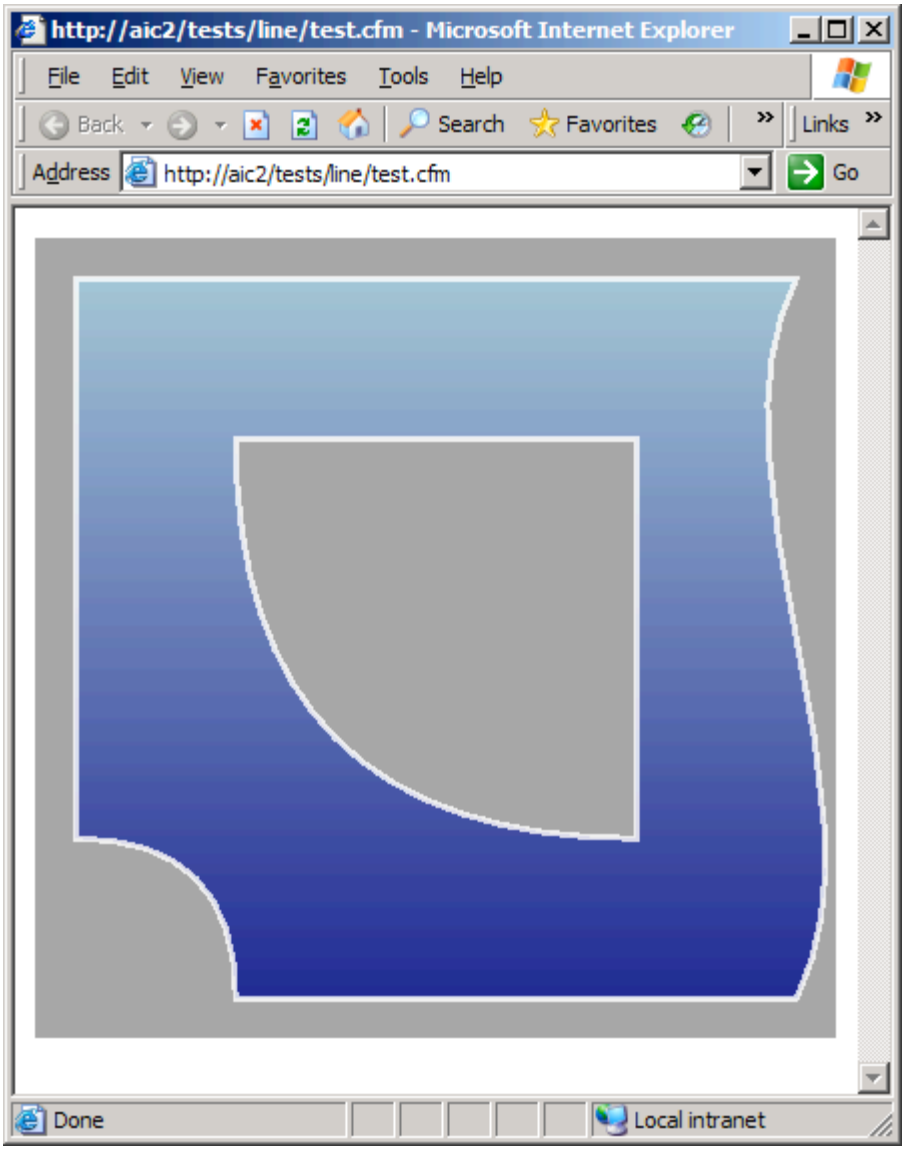
series of method calls to create a path, add lines to your path, and set path properties before drawing the path. Drawing Paths on Images functions behave similar to Drawing Polygons on Images.

Example

```
<!---
This example demonstrates some of the path features.
--->
<cfset myImage = CreateObject("Component", "Image") />

<!--- create some colors --->
<cfset darkGray = myImage.getColorByName("DarkGray") />
<cfset lightblue = myImage.getColorByName("lightblue") />
<cfset darkBlue = myImage.getColorByName("DarkBlue") />
<cfset white = myImage.getColorByName("White") />

<!--- set the image's background color --->
<cfset myImage.setBackgroundColor(darkGray) />

<!--- create a new image --->
<cfset myImage.createImage(400, 400) />

<!--- set the fill and transparency for the shape --->
<cfset myImage.setFill(myImage.createGradient(0,0, 0, 400, lightblue, ¬
darkblue)) />
<cfset myImage.setTransparency(80) />

<!--- make a nice heavy stroke --->
<cfset myImage.setStroke(3) />

<!--- create a new path object, setting it's initial point --->
<cfset myPath = myImage.createPath(20, 20) />

<!--- add a strait line --->
<cfset myImage.addPathLine(myPath, 380, 20) />

<!--- add a bezier curve --->
<cfset myImage.addPathBezierCurve(myPath, 380,380, 330,120, 430,280) />

<!--- add another strait line --->
<cfset myImage.addPathLine(myPath, 100, 380) />

<!--- draw a quadratic curve --->
<cfset myImage.addPathQuadraticCurve(myPath, 20,300, 100, 300) />

<!--- add another strait line --->
<cfset myImage.addPathLine(myPath, 20, 20) />

<!--- jump to the center of the shape --->
<cfset myImage.addPathJump(myPath, 100, 100) />

<!--- draw a few more lines --->
<cfset myImage.addPathQuadraticCurve(myPath, 300,300, 100, 300) />
<cfset myImage.addPathLine(myPath, 300, 100) />

<!--- close the path --->
<cfset myImage.closePath(myPath) />

<!--- draw the path --->
```

```
<cfset myImage.drawPath(myPath) />

<!--- write the image --->
<cfset myImage.writeImage(expandPath("line.png"), "png") />

<img src="line.png">
```

Results



addPathBezierCurve()

Description

The addPathBezierCurve() method appends a bezier curve to a path created with the createPath() method. A bezier curve is a curve controlled by two control points along its access. The curve is drawn from the last point in the shape to the target X and Y coordinates.

Syntax

addPathBezierCurve(Path, x, y, controlX1, controlY1, controlX2, controlY2)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to add the bezier curve to.
X	Yes	Numeric	I am the target X coordinate
Y	Yes	Numeric	I am the target Y coordinate
controlX1	Yes	Numeric	I am the X coordinate for the first control point.
controlY1	Yes	Numeric	I am the Y coordinate for the first control point.
controlX2	Yes	Numeric	I am the X coordinate for the second control point.
controlY2	Yes	Numeric	I am the Y coordinate for the second control point.

See Also

Path Overview createPath() drawPath()

addPathJump()

Description

The addPathJump() method moves adds a point to the path without drawing a line from the last point. When addPathJump() is used the point specified becomes the new target point for closing the path with the closePath() method.

For example, if you add three lines to the path and call the closePath() method, the last point will be connected to the first point specified when you created the path with the createPath() method. However, if you add three lines to the path then call addPathJump(), then draw more lines, subsequent calls to closePath() will use the point specified with the last addPathJump() call.

The example in the Path Overview demonstrates how to draw complex shapes with negative space using this technique.

Syntax

addPathJump(Path, x, y)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to add the jump to.
X	Yes	Numeric	I am the target X coordinate
Y	Yes	Numeric	I am the target Y coordinate

See Also

Path Overview createPath() drawPath()

addPathLine()

Description

The addPathLine() method adds a strait line to the path. The line is drawn from the last point to the specified point.

Syntax

addPathLine(Path, x, y)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to add the strait line to.
X	Yes	Numeric	I am the target X coordinate
Y	Yes	Numeric	I am the target Y coordinate

See Also

Path Overview createPath() drawPath()

addPathQuadraticCurve()

Description

The addPathQuadraticCurve() method appends a quadratic curve to a path created with the createPath() method. A quadratic curve is a curve controlled by single control point. The curve is drawn from the last point in the shape to the target X and Y coordinates.

Syntax

addPathQuadraticCurve(Path, x, y, controlX1, controlY1)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to add the quadratic curve to.
X	Yes	Numeric	I am the target X coordinate
Y	Yes	Numeric	I am the target Y coordinate
controlX1	Yes	Numeric	I am the X coordinate for the first control point.
controlY1	Yes	Numeric	I am the Y coordinate for the first control point.

See Also

Path Overview createPath() drawPath()

closePath()

Description

The closePath() method draws a line from the current point to the first point added with the createPath() method or the last point specified with addPathJump() if it has been called.

For example, if you add three lines to the path and call the closePath() method, the last point will be connected to the first point specified when you created the path with the createPath() method. However, if you add three lines to the path then call addPathJump(), then draw more lines, subsequent calls to closePath() will use the point specified with the last addPathJump() call.

The example in the Path Overview demonstrates how to draw complex shapes with negative space using this technique.

Syntax

closePath(Path)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to close.

See Also

Path Overview createPath() drawPath() addPathJump()

createPath()

Description

The createPath() method creates and returns a new Path object. The new path object has only one point which is specified when you call createPath(). You can add lines, curves and more to that path by calling the addPathBezierCurve(), addPathJump(), addPathLine(), addPathQuadraticCurve(), closePath() methods.

Once you are done creating your path you can draw it using the drawPath() method.

Syntax

createPath(x, y)

Parameter	Required	Type	Description
X	Yes	Numeric	I am the starting X coordinate
Y	Yes	Numeric	I am the starting Y coordinate

See Also

Path Overview createPath() drawPath()

drawPath()

The drawPath() method is used to draw Paths created by the createPath() method. You can add lines, curves and more to that path by calling the addPathBezierCurve(), addPathJump(), addPathLine(), addPathQuadraticCurve(), closePath() methods.

Polygons with only the starting point can be drawn but do not appear. The reason is that one point can not be displayed because it takes up no space

Paths with two or more points can be drawn. Depending on the method used to add points, the path may appear as a line.

Syntax

drawPath(Path)

Parameter	Required	Type	Description
Path	Yes	Path	The Path object to draw.

See Also

Path Overview createPath()

Drawing Simple Text on Images

Overview

The Alagad Image Component provides two ways to draw text into your image, Simple Strings which are quick and easy to use, and Advanced String which are more complicated but more powerful.

Simple Strings allow you to draw text in a particular font and style into the image. The entire string is formatted in the same font. You can use the getSimpleStringMetrics() method to find of the string size information, such as its height and width.

Advanced Strings allow you more control over the formatting of your text. You have control over the formatting of every character in your string. For more information see the Drawing Advanced Text on Images section.

drawSimpleString()

Description

The drawSimpleString() method draws a Simple String into the image at a specific location. You can optionally specify the font of the text. See the Fonts section for more information on creating fonts.

Syntax

drawSimpleString(string, x, y, Font)

Parameter	Required	Type	Description
String	Yes	String	The string to draw into the image.
X	Yes	Numeric	The X location of the start of the baseline to draw the text on.
Y	Yes	Numeric	The Y location of the start of the baseline to draw the text on.
Font	No	Font	This is the Font Object to use when drawing the image. If not provided, the default font will be used.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\catAndCup.jpg") />

<!--- set the font --->
<cfset timesNewRoman = myImage.loadSystemFont("Times New Roman", 20,
"boldItalic") />

<!--- create a string to write into the image --->
<cfset myString = "Where the heck is my milk?" />

<!---
      Find the metrics (width, height, etc) for the string.
      We will use this to center the string in the image.
--->
<cfset metrics = myImage.getSimpleStringMetrics(myString, ¬
timesNewRoman) />

<!--- determine the X coordinate so we can center the text ¬
in the image --->
<cfset x = (myImage.getWidth() - metrics.width) / 2 />

<!--- draw the text, centered at the top of the image --->
<cfset myImage.drawSimpleString(myString, x, 30, timesNewRoman) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\catCupAndText.jpg", "jpg") />

<!--- the new images --->
<p>
<b>catCupAndText.jpg:</b><br>
<img src="/examples/catCupAndText.jpg"><br>
</p>
```

Results

See Also

loadSystemFont() loadTTFFile() getSimpleStringMetrics()

getSimpleStringMetrics()

Description

The getSimpleStringMetrics() method returns a structure containing information about the size of the string. The structure of information returned is:

- Height – The height of the string in totality. This equals the accent + decent + leading.
- Width – The width of the string in totality.
- Ascent – The height of the text above the baseline to the top of the characters.

- Descent – The height of the text from the baseline to the bottom of the characters.
- Leading – The line spacing of the text. *Note: The Image Component does not wrap text so this is not used by the Image Component.*

Syntax

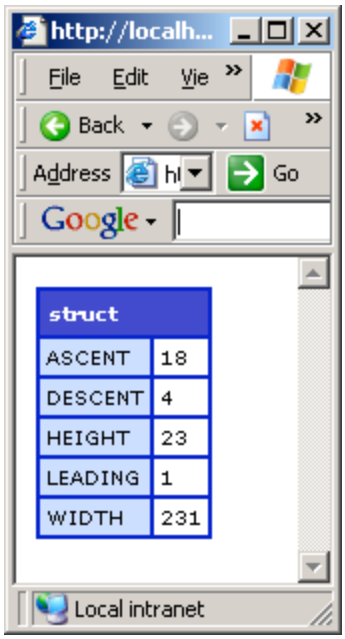
Struct = getSimpleStringMetrics(string, Font)

Parameter	Required	Type	Description
String	Yes	String	The string to get the metrics of.
Font	No	Font	This is the Font Object to use when drawing the image. If not provided, the default font will be used.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open the image to inspect --->
<cfset myImage.readImage("d:\examples\catAndCup.jpg") />

<!--- set the font --->
<cfset timesNewRoman = myImage.loadSystemFont("Times New Roman", 20, ¬
"boldItalic") />

<!--- create a string to write into the image --->
<cfset myString = "Where the heck is my milk?" />

<!--- Find the metrics (width, height, etc) for the string. --->
<cfset metrics = myImage.getSimpleStringMetrics(myString,  ¬
timesNewRoman) />

<!--- the resulting metrics --->
<cfdump var="#metrics#">
```

Results



Drawing Advanced Text on Images

Overview

The Alagad Image Component provides two ways to draw text into your image, Simple Strings which are quick and easy to use, and Advanced String which are more complicated but more powerful.

Advanced Strings give you lots of control over the formatting of your text. You have control of the formatting of every character in your string, including its font and styling. You can use the getStringMetrics() method to find of the string size information, such as its height and width.

Be aware that behaviors of the Advanced String methods vary, depending on fonts, locales and other variables. Be sure to test your usage of these methods thoroughly. Not all methods work under all circumstances.

Simple Strings allow you to draw a text in a particular font and style into the image. The entire string is formatted in the same font. For more information see the Drawing Simple Text on Images section.

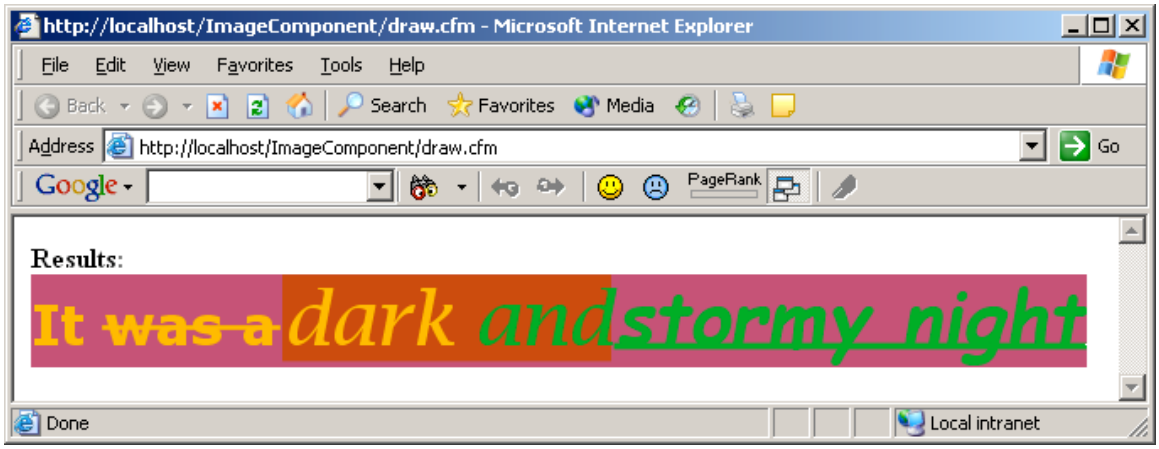
Example

The following example demonstrates the usage of some of the most common Advanced String formatting methods.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
```

```
<cfset randomColor1 = myImage.createColor(randrange(0,255), ¬
randrange(0,255), randrange(0,255)) />
<cfset randomColor2 = myImage.createColor(randrange(0,255), ¬
randrange(0,255), randrange(0,255)) />
<cfset randomColor3 = myImage.createColor(randrange(0,255), ¬
randrange(0,255), randrange(0,255)) />
<cfset randomColor4 = myImage.createColor(randrange(0,255), ¬
randrange(0,255), randrange(0,255)) />

<!--- create three random fonts --->
<cfset fontArray = myImage.getSystemFonts() />
<cfset fontIndex = randRange(1, ArrayLen(fontArray)) />
<cfset randomFont1 = myImage.loadsystemFont(fontArray[fontIndex], ¬
randRange(25, 45), "bold") />
<cfset fontIndex = randRange(1, ArrayLen(fontArray)) />
<cfset randomFont2 = myImage.loadsystemFont(fontArray[fontIndex], ¬
randRange(25, 45), "italic") />
<cfset fontIndex = randRange(1, ArrayLen(fontArray)) />
<cfset randomFont3 = myImage.loadsystemFont(fontArray[fontIndex], ¬
randRange(25, 45), "boldItalic") />

<!--- create a new image --->
<cfset myImage.setBackgroundColor(randomColor1) />

<!--- create an advanced string --->
<cfset myString = "It was a dark and stormy night." />
<cfset advancedString = myImage.createString(myString) />

<!--- set some font and size settings --->
<cfset myImage.setStringFont(advancedString, randomFont1, 0, 8) />
<cfset myImage.setStringFont(advancedString, randomFont2, 9, 17) />
<cfset myImage.setStringFont(advancedString, randomFont3, 18, 30) />

<!--- set some font colors --->
<cfset myImage.setStringForeground(advancedString, randomColor3, 0, ¬
14) />
<cfset myImage.setStringForeground(advancedString, randomColor4, 14, ¬
30) />

<!--- get the string's metrics --->
<cfset metrics = myImage.getStringMetrics(advancedString) >

<!--- create an image the width and height of the text --->
<cfset myImage.createImage(metrics.width, metrics.height) />

<!--- draw the text into the image --->
<cfset myImage.drawString(advancedString, 0, metrics.height - ¬
metrics.descent) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\textExample.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/textExample.png"><br>
</p>
```

Results



createString()

Description

The createString() method creates and returns an unformatted Advanced String Object. Once you have created an Advanced String Object you can format it using a number of formatting methods such as setStringSize(), setStringUnderline(), and setStringBackground(). At any time you can call drawString() to draw your Advanced String Object into your image.

Note: Once the text in the Advanced String has been defined it can not be changed.

Syntax

AdvancedString = createString(string)

Parameter	Required	Type	Description
String	Yes	String	The string used to create the Advanced String Object.

Example

See the Drawing Advanced Text on Images Example

See Also

drawString() setStringSize() setStringUnderline() setStringBackground()

drawString()

Description

The drawString() method draws the Advanced String Object into your image.

Syntax

drawString(advancedString, x, y)

Parameter	Required	Type	Description

AdvancedString	Yes	Advanced String Object	This is the Advanced String Object to draw into your image
X	Yes	Numeric	The X location of the start of the baseline to draw the text on.
Y	Yes	Numeric	The Y location of the start of the baseline to draw the text on.

Example

See the Drawing Advanced Text on Images Example

See Also

getStringMetrics()

getStringMetrics()

The getStringMetrics() method returns a structure containing information about the size of the Advanced String Object. The structure of information returned is:

- Height – The height of the string in totality. This equals the accent + decent + leading.
- Width – The width of the string in totality.
- Ascent – The height of the text above the baseline to the top of the characters.
- Descent – The height of the text from the baseline to the bottom of the characters.
- Leading – The line spacing of the text. *Note: The Image Component does not wrap text so this is not used by the Image Component.*

Syntax

Struct = getStringMetrics(advancedString)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	This is the Advanced String Object to get the metrics of.

Example

See the Drawing Advanced Text on Images Example

See Also

drawString()

setStringBackground()

Description

The setStringBackground() method sets the background color for the provided Advanced String Object. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringBackground(advancedString, color, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Color	Yes	Color Object	The color to set the background to.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

Example

See the Drawing Advanced Text on Images Example

See Also

createColor() getColorByName()

setStringDirection()

Description

In general, the setStringDirection() method sets the direction of a font. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

The results of using this method may vary.

Syntax

setStringDirection(advancedString, direction, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Direction	Yes	String	Indicates the direction option to use. Options are: • ltr – Left to Right • rtl – Right to Left
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

setStringFamily()

Description

The setStringFamily() method sets the font family used for the provided Advanced String Object. The family can be any system font. This functions

similar to setStringFont(). The main difference being that setStringFont() allows you to specify a particular font which has been loaded with loadTTFFile() or loadSystemFont() and set to a specific size and style, setStringFont() only sets the font. To set the size and posture use setStringSize() and setStringPosture() respectively. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringFamily(advancedString, family, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the font family of.
Family	Yes	String	A name of a system font to use.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

See Also

getSystemFonts() setStringFont() setStringSize() setStringPosture()

setStringFont()

Description

The setStringFont() method sets the font of the provided Advanced String Object. This method functions similarly to setStringFamily() but allows you to provide a font loaded using loadSystemFont() or loadTTFFile(). Because both of these methods set the size and style of the font these settings usually override other Advanced String Object formatting methods. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringFont(advancedString, family, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the font family of.
Font	Yes	Font Object	This is the Font Object to use when drawing the text.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

See Also

loadTTFFile() loadSystemFont() setStringFont()

setStringForeground()

Description

The setStringForeground() method sets the foreground color for the provided Advanced String Object. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringForeground(advancedString, color, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Color	Yes	Color Object	The color to set the foreground to.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

Example

See the Drawing Advanced Text on Images Example

See Also

createColor() getColorByName()

setStringSize()

Description

The setStringSize() method sets the font size for the Advanced String Object. This may be overridden by font sizes specified in setStringFont(). By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringSize(advancedString, size, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Size	Yes	Numeric	The font size to set the string to.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

See Also

setStringFont()

setStringStrikeThrough()

Description

The setStringStrikeThrough() method turns on and off strikethrough for the provided Advanced String Object. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringStrikeThrough(advancedString, state, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
State	Yes	Boolean	Indicates if strikethrough is turned on or off.
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

Example

See the Drawing Advanced Text on Images Example

setStringUnderline()

Description

The setStringUnderline() method sets the underline options for the provided Advanced String Object. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

Syntax

setStringUnderline(advancedString, underline, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Underline	Yes	String	Indicates the underline option to use. Options are: • on • off • lowOnePixel • lowTwoPixel • lowDotted • lowGray • lowDashed
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the

			style to.

Example

See the Drawing Advanced Text on Images Example

setStringPosture()

Description

In general, the setStringPosture() method sets the posture, or skew, of a font. This gives the font an italic appearance. If the current font has an italic face the italic face may be used. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

The results of using this method may vary.

Syntax

setStringPosture(advancedString, posture, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Posture	Yes	String	Indicates the posture option to use. Options are: • oblique • regular
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

setStringWeight()

Description

In general, the setStringWeight() method sets the weight of a font. This sets the bold appearance of the font. The current font may need to have a corresponding face for the selected style. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

The results of using this method may vary.

Syntax

setStringWeight(advancedString, weight, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Weight	Yes	String	Indicates the weight option to use. Options are:

			• regular • bold • demiBold • demiLight • extraLight • extrabold • heavy • light • medium • semiBold • ultraBold
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

setStringWidth()

Description

In general, the setStringWidth() method sets the width of a font. The current font may need to have a corresponding face for the selected style. By setting the start and end parameters of this method you can limit the portion of the Advanced String Object it affects.

The results of using this method may vary.

Syntax

setStringWidth(advancedString, width, x, y)

Parameter	Required	Type	Description
AdvancedString	Yes	Advanced String Object	The advanced string to set the style of.
Width	Yes	String	Indicates the width option to use. Options are: • regular • condensed • extended • semiCondensed • semiExtended
Start	No	Numeric	The starting character to apply the style to.
End	No	Numeric	The ending character to apply the style to.

Fonts

getSystemFonts()

Description

The getSystemFonts() method returns an array of system fonts installed and available for use. The provided names can be used in conjunction with the loadSystemFont() and setStringFont() methods.

Syntax

Array = getSystemFonts()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- dump the system fonts --->
<cfdump var="#myImage.getSystemFonts()#">
```

Results

See Also

loadSystemFont() setStringFont()

loadSystemFont()

Description

The loadSystemFont() method loads and returns a font based on the provided system font name. If you provide a valid font name, such as “Lucidia Sans Typewriter,” and your system has this font installed the method will instantiate and return this font as a Font Object.

You can also provide the following generic names which are mappings to system fonts provided by Java:

- Serif
- SansSerif
- Monospaced
- Dialog
- DialogInput

Syntax

Font = loadSystemFont(systemFontName, size, style)

Parameter	Required	Type	Description
SystemFontName	Yes	String	This is a system font name as returned by getSystemFonts() or a logical font name as outlined above.
Size	Yes	Numeric	The font size to use for this font.
Style	No	String	The style to apply to this font. Options are: • Plain *(default)* • Bold • Italic • BoldItalic

See Also

loadTTFFile()

loadTTFFile()

Description

The loadTTFFile() method loads and returns a font directly from a .TTF True Type Font file. If you provide a valid path to a .TTF font file the method will instantiate and return the font as a Font Object.

Syntax

Font = loadTTFFile(ttfFilePath, size, style)

Parameter	Required	Type	Description
TtfFilePath	Yes	String	This is a path to a .TTF True Type

			Font file.
Size	Yes	Numeric	The font size to use for this font.
Style	No	String	The style to apply to this font. Options are: • Plain *(default)* • Bold • Italic • BoldItalic

See Also

loadSystemFont()

Component Properties

Overview

All of the various methods for drawing and writing text into an image are affected by the Image Component's current settings. For instance, to set the fill color of a rectangle as you draw it into your image, you must first call the setFill() method.

The methods listed below affect the results of calling other Image Component methods.

reset()

Description

The reset() method resets the Image Component back to it's initial settings. This method is a shortcut method which calls all of the reset methods listed in this section.

Syntax

Reset()

resetAntialias()

Description

The resetAntialias() method sets the antialiasing setting back to it's default setting. The antialias setting is on by default.

Syntax

resetAntialias()

resetBackgroundColor()

Description

The resetBackgroundColor() method sets the background color back to the default color. Typically this color is Black.

Syntax
resetBackgroundColor()

resetComposite()

Description
The resetComposite() method sets the composite method back to the default setting.

Syntax
resetComposite()

resetFill()

Description
The resetFill() method sets the current fill back to the default. Typically the default fill is none.

Syntax
resetFill()

resetRotation()

Description
The resetRotation() method sets the current rotation used when drawing into the image back to 0 degrees.

Syntax
resetRotation()

resetShear()

Description
The resetShear() method resets the shear used when drawing into the image back to the default.

Syntax
resetShear()

resetStroke()

Description
The resetStroke() method resets the stroke used to draw shapes into the image. The default is usually a one pixel white line.

Syntax
resetStroke()

resetTransparency()

The resetTransparency() method resets the transparency used when drawing into the image back to the default. The default is usually completely opaque.

Syntax

resetTransparency()

setAntialias()

Description

The setAntialias() method is used to turn antialiasing on and off when drawing shapes and text into the image. Antialiasing is a technique used to soften jagged edges apparent when drawing lines and shapes into images. The default setting is for antialiasing to be on.

Syntax

setAntialiasing(antialiasing)

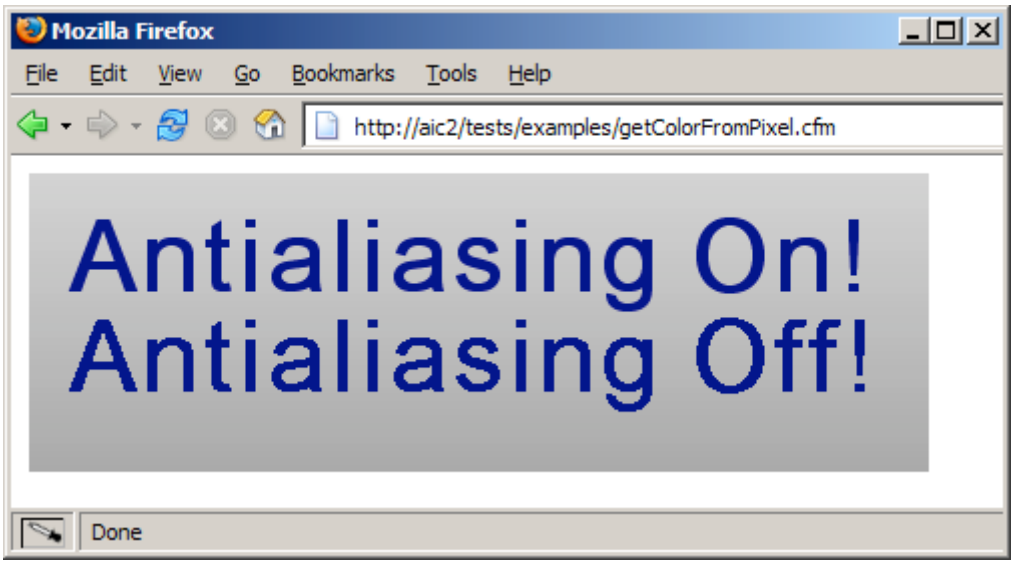
Parameter	Required	Type	Description
Antialiasing	Yes	Boolean	True / False value indicating if antialiasing should be turned on or off.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image2.Image") />

<!--- crate a new image --->
<cfset myImage.createImage(450, 150) />

<!--- create a font and color --->
<cfset comic = myImage.loadSystemFont("Comic Sans", 50, "bold") />
<cfset lightGray = myImage.getColorByName("lightGray") />
<cfset darkGray = myImage.getColorByName("darkGray") />
<cfset darkBlue = myImage.getColorByName("darkBlue") />
<cfset gradient = myImage.createGradient(0,0, 0,150, lightGray, ¬
darkGray) />

<!--- fill the background --->
<cfset myImage.setFill(gradient) />
<cfset myImage.drawRectangle(0, 0, 450, 150) />

<!--- draw antialiased text into the image --->
<cfset myImage.setFill(darkBlue) />
<cfset myImage.drawSimpleString("Antialiasing On!", 20, 60, comic) />

<!--- turn antialiasing off and draw another string --->
<cfset myImage.setAntialias(false) />
<cfset myImage.drawSimpleString("Antialiasing Off!", 20, 110, comic) />

<!--- draw the image --->
<cfset myImage.writeImage(expandPath("example.png"), "png") />

<!--- show the image --->
```

```
<img src="example.png">
```

Results



See Also

resetAntialias()

setBackgroundColor()

Description

The setBackgroundColor() method sets the background color for the image. The background color is used when any methods result in the image canvas being displayed. For example, clearImage().

If the image is an ARGB image and the color specified for the background has a transparent alpha value specified the resulting image's background will be accordingly transparent if the output format supports that type of transparency.

Syntax

setBackgroundColor(color)

Parameter	Required	Type	Description
Color	Yes	Color Object	The color object to set the background to.

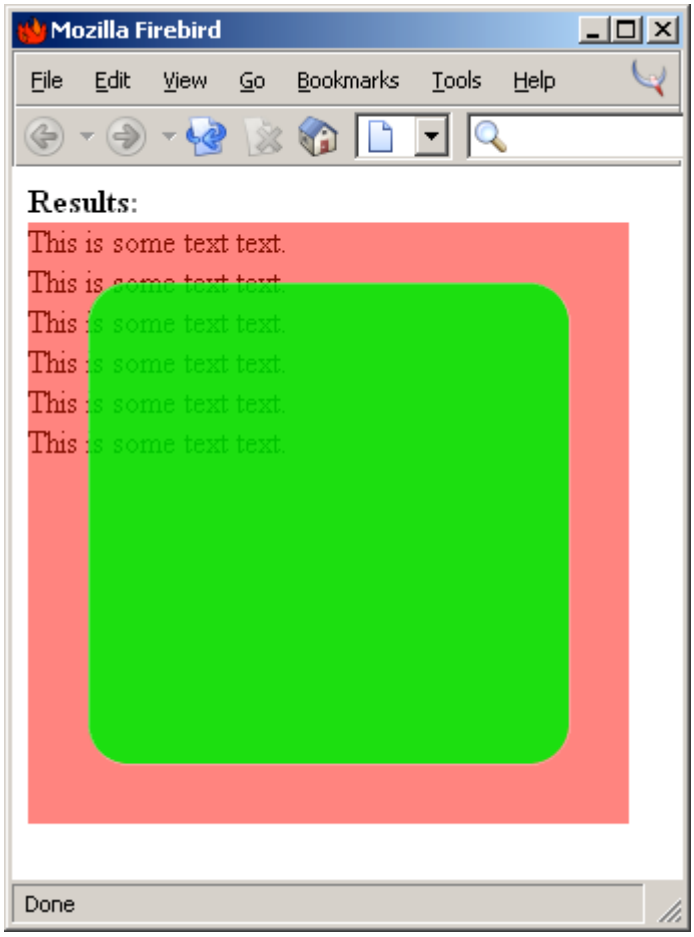
Example

The following example demonstrates creating an image with a partially transparent background color and displaying it over text in a web browser.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset green = myImage.createColor(0,255,0,200) />
```

```
<cfset red = myImage.createColor(255,0,0,128) />

<!--- create a new image --->
<cfset myImage.setBackgroundColor(red) />

<!--- create a new image --->
<cfset myImage.createImage(300, 300, "argb") />

<!--- draw a round rectangle into the image --->
<cfset myImage.setFill(green) />
<cfset myImage.setStroke(0) />
<cfset myImage.drawRoundRectangle(30, 30, 240, 240, 40, 40) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\setBackground.png", "png") />

<!--- the new images --->
<b>Results:</b><br>

<div style="position: absolute;">
<img src="/examples/setBackground.png">
</div>

<div>
This is some test text.<Br>
This is some test text.<Br>
This is some test text.<Br>
This is some test text.<Br>
This is some test text.<Br>
This is some test text.<Br>
</div>
```

Results



See Also

clearImage() clearRectangle()

setComposite()

Description

The setComposite() method sets the algorithm used for compositing images drawn with drawImage(). The algorithms control how the alpha channels for the two images are combined.

Syntax

setComposite(composite)

Parameter	Required	Type	Description
Composite	Yes	String	The composite algorithm to use. Options are: • dstAtop • dstIn • dstOut • dstOver • srcAtop

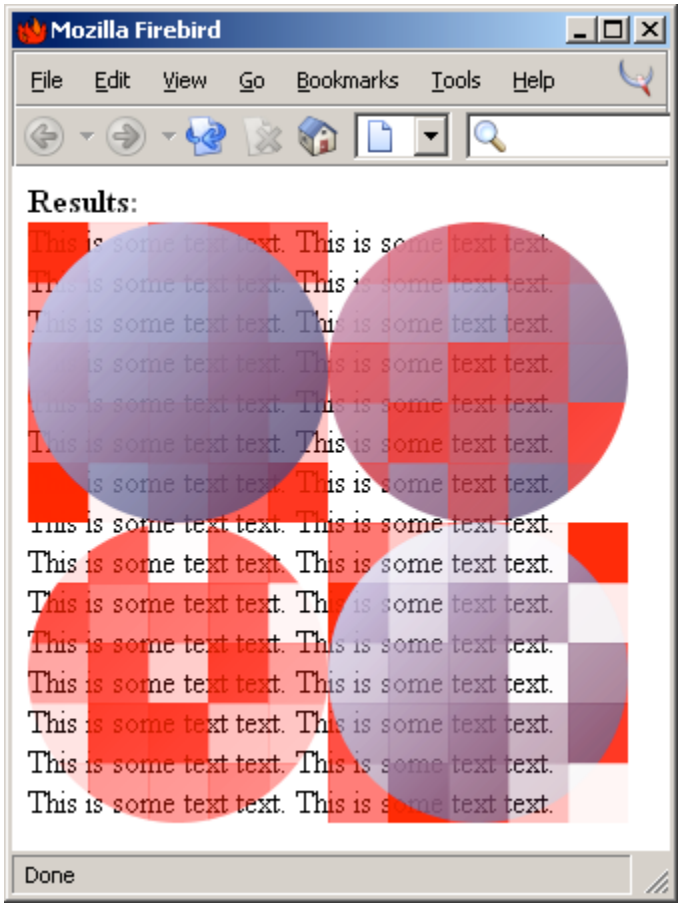
			• srcIn • srcOut • srcOver

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset transparent = myImage.createColor(0,0,0,0) />
<cfset lightblue = myImage.createColor(220,220,255,255) />
<cfset darkblue = myImage.createColor(50,50,100,180) />

<!--- create a gradient --->
<cfset myGrad = myImage.createGradient(0,0, 150,150, lightblue, ¬
darkblue) />

<!--- set the image background color to transparent --->
<cfset myImage.setBackgroundColor(transparent) />

<!--- create a new image --->
<cfset myImage.createImage(150, 150, "argb") />

<!--- draw a blue circle into the image --->
<cfset myImage.setFill(myGrad) />
<cfset myImage.drawOval(0,0,150,150) />

<!--- output the circle image --->
<cfset myImage.writeImage("d:\examples\circle.png", "png") />

<!--- reset/create another new image --->
<cfset myImage.createImage(300, 300, "argb") />

<!---
Loop over the image and create a checkerboard pattern.
Give each square a different level of transparency
--->
<cfloop from="0" to="300" index="x" step="30">
      <cfloop from="0" to="300" index="y" step="30">
            <!--- create a red color with  a random transparency --->
            <cfset newTransparentColor = myImage.CreateColor(255, 0, ¬
0, randrange(0,255)) />
            <cfset myImage.setStroke(0, newTransparentColor) />
            <cfset myImage.setFill(newTransparentColor) />
            <!--- draw a square into the image --->
            <cfset myImage.drawRectangle(x, y, 30, 30) />
      </cfloop>
</cfloop>

<!--- composite the circle image on top of the square image four times
--->
<!--- use the default composite --->
<cfset myImage.drawImage("d:\examples\circle.png", 0,0,150,150) />

<cfset myImage.setComposite("dstAtop") />
<cfset myImage.drawImage("d:\examples\circle.png", 150,0,150,150) />

<cfset myImage.setComposite("srcAtop") />
```

```
<cfset myImage.drawImage("d:\examples\circle.png", 150,150,150,150) />

<cfset myImage.setComposite("dstIn") />
<cfset myImage.drawImage("d:\examples\circle.png", 0,150,150,150) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\composite.png", "png") />

<!--- the new images --->
<b>Results:</b><br>
<div style="position: absolute;">
<img src="/examples/composite.png">
</div>
<div>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
This is some test text. This is some test text. <Br>
</div>
```

Results



setFill()

Description

The setFill() method sets how the interior of drawn shapes is filled. The fill object can accept three different types of objects to fill shapes with:

- **Color Object** – A Color Object will fill the shape with one solid color. A color can be created with either the createColor() or getColorByName() methods.
- **Gradient Object** – A Gradient Object will fill the shape with a gradient color pattern. A Gradient Object can be created with createGradient().
- **Texture Object** – A texture object will fill the shape with a repeating pattern of another image. A Texture Object can be created with createTexture().

Syntax

setFill(fill)

Parameter	Required	Type	Description
Fill	Yes	Color, Gradient,	Defines how drawn shapes will be filled.

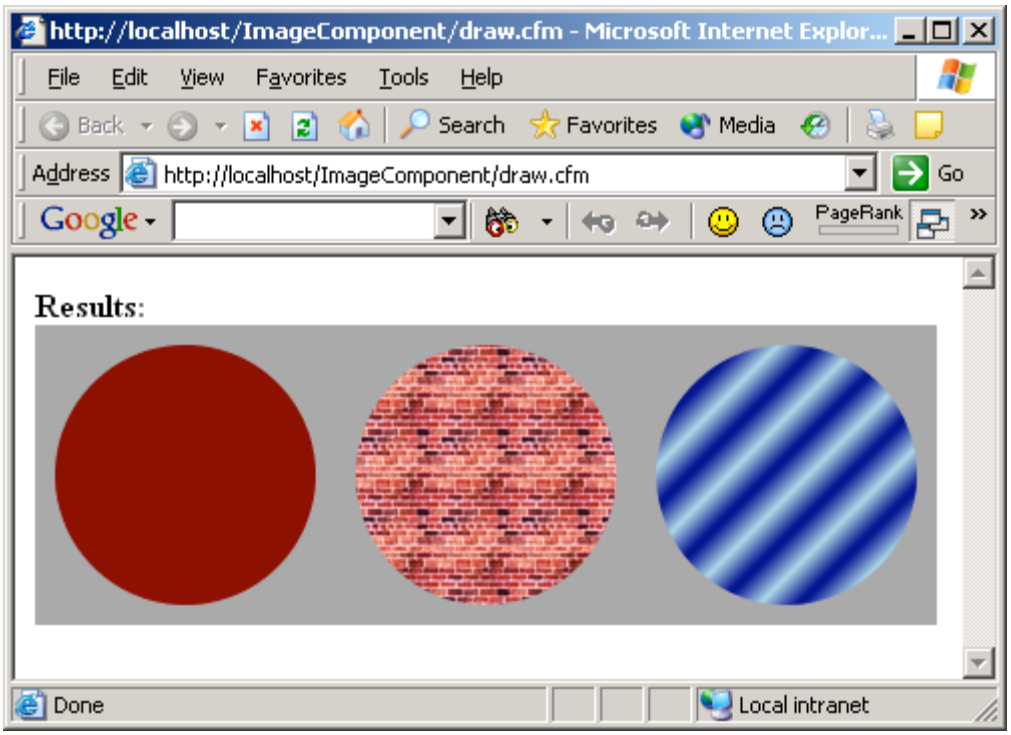
		or Texture Object	

Example

The following example draws three shapes and demonstrates the three types of fills.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset darkgray = myImage.getColorByName("darkGray") />
<cfset darkRed = myImage.getColorByName("darkRed") />

<cfset lightblue = myImage.getColorByName("lightblue") />
<cfset darkBlue = myImage.getColorByName("darkBlue") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(darkgray) />

<!--- create a new new image --->
<cfset myImage.createImage(450, 150) />

<!--- draw a circle with a solid fill --->
<cfset myImage.setFill(darkRed) />
<cfset myImage.drawOval(10, 10, 130, 130) />

<!--- draw a circle with a texture fill --->
<cfset texture = myImage.createTexture("d:\examples\bricks.jpg", 0, 0,¬
40, 40) />
<cfset myImage.setFill(texture) />
<cfset myImage.drawOval(160, 10, 130, 130) />

<!--- draw a circle with a graident fill --->
<cfset myGradient = myImage.createGradient(0, 0, 10, 10, darkBlue, ¬
lightblue, true) />
<cfset myImage.setFill(myGradient) />
<cfset myImage.drawOval(310, 10, 130, 130) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\fills.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/fills.png">
</p>
```

Results



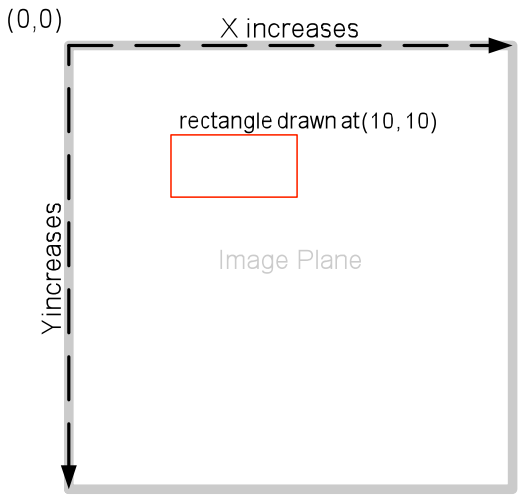
See Also

createColor() getColorByName() createGradient() createTexture()

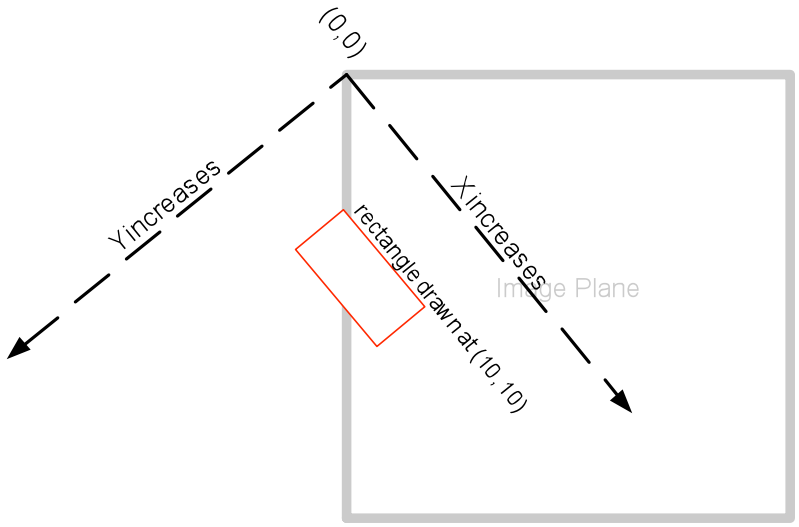
setRotation()

Description

The setRotation() method rotates the plane on which shapes and images are drawn into your image. Typically the drawing plane could be represented as follows:



If you were to use the setRotation() method to set the drawing plane, the results could be represented as follows:



Syntax

setRotation(angle, x, y)

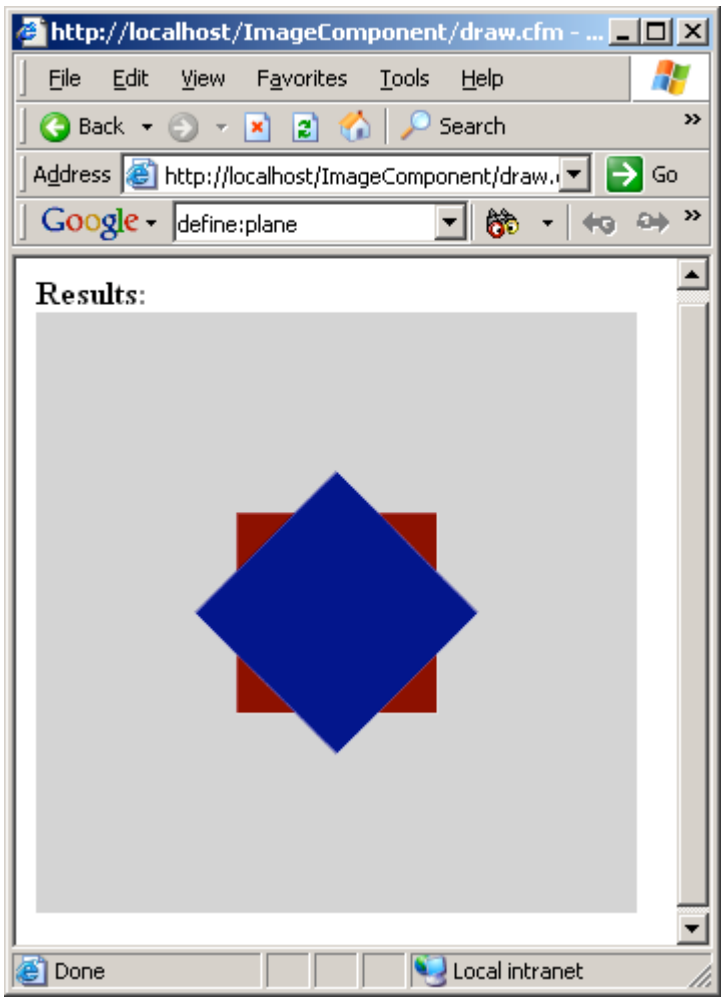
Parameter	Required	Type	Description
Angle	Yes	Numeric	The angle of rotation
X	No	Numeric	The X location where the rotation takes place. Defaults to 0.
Y	No	Numeric	The Y location where the rotation takes place. Defaults to 0.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset lightGray = myImage.getColorByName("lightGray") />
<cfset darkRed = myImage.getColorByName("darkRed") />
<cfset darkBlue = myImage.getColorByName("darkBlue") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(lightGray) />

<!--- create a new new image --->
<cfset myImage.createImage(300, 300) />

<!--- draw a square --->
<cfset myImage.setFill(darkRed) />
<cfset myImage.setStroke(0) />
<cfset myImage.drawRectangle(100, 100, 100, 100) />

<!--- set the rotation --->
<cfset myImage.setRotation(45, 150, 150) />

<!--- draw another square --->
<cfset myImage.setFill(darkBlue) />
<cfset myImage.setStroke(0) />
<cfset myImage.drawRectangle(100, 100, 100, 100) />
```

```
<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\rotation.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/rotation.png">
</p>
```

Results



setShear()

Description

The setShear() method distorts an image using a shear transform.

Syntax

setShear(x, y)

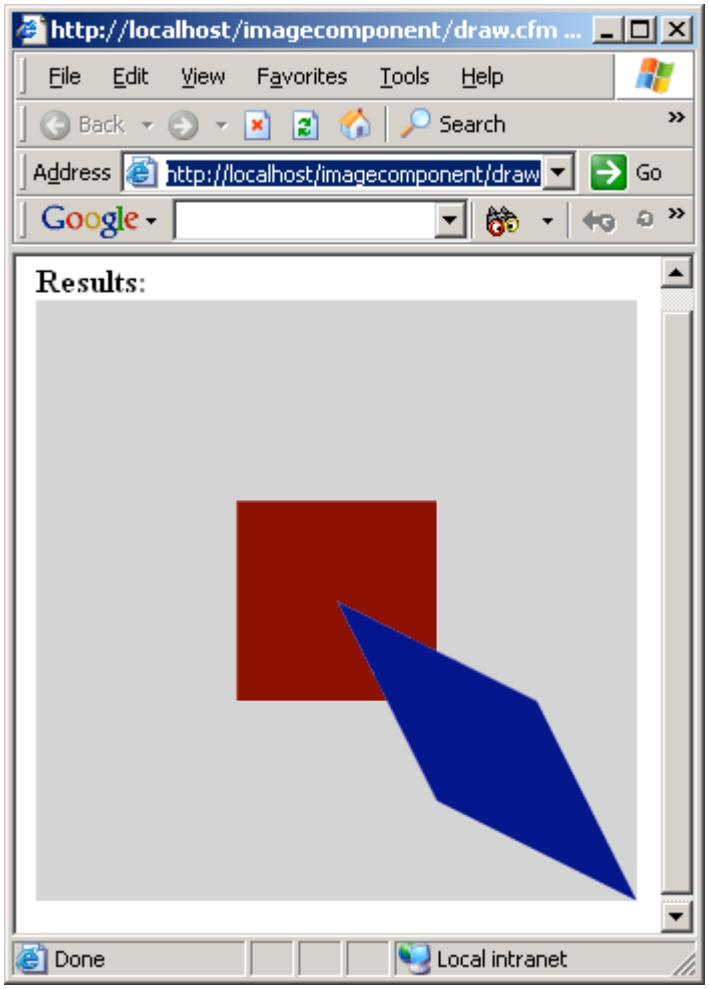
Parameter	Required	Type	Description
X	Yes	Numeric	The multiplier by which the transformation occurs in the X axis.

Y	Yes	Numeric	The multiplier by which the transformation occurs in the Y axis.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset lightGray = myImage.getColorByName("lightGray") />
<cfset darkRed = myImage.getColorByName("darkRed") />
<cfset darkBlue = myImage.getColorByName("darkBlue") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(lightGray) />

<!--- create a new new image --->
<cfset myImage.createImage(300, 300) />

<!--- draw a square --->
<cfset myImage.setFill(darkRed) />
<cfset myImage.setStroke(0) />
<cfset myImage.drawRectangle(100, 100, 100, 100) />

<!--- set the rotation --->
<cfset myImage.setShear(50, 50) />

<!--- draw another square --->
<cfset myImage.setFill(darkBlue) />
<cfset myImage.setStroke(0) />
<cfset myImage.drawRectangle(100, 100, 100, 100) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\rotation.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/rotation.png">
</p>
```

Results



setStroke()

Description

The setStroke() method sets the stroke settings used when drawing shapes into the image. Using set stroke you can control the stroke width, color, miter limit, dashing settings, end cap style and line join style.

Syntax

setStroke(weight, color, cap, join, miterLimit, dashLengthList, dashOffset)

Parameter	Required	Type	Description
Weight	Yes	Numeric	The width of the stroke.
Color	No	Color Object	The Color Object to use when drawing the stroke.
Cap	No	String	The end cap style for lines drawn in this stroke. Options are: • Butt • Round • Square *(default)*

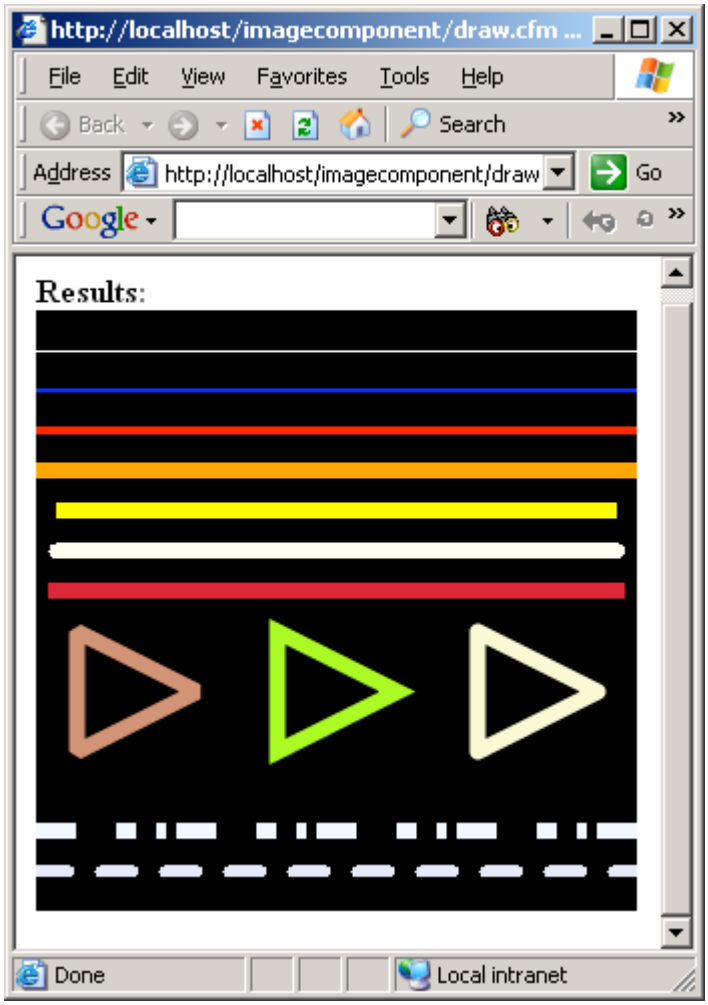
Join	No	String	The styles with witch two lines are joined. Options are: • Bevel • Miter *(default)* • Round
MiterLimit	No	Numeric	The angle at which miter joins are trimmed. Defaults to 10.
DashLengthList	No	Comma Delimited List of Numeric Values	A comma delimited list of links of dash segments. Each item alternates between a length of line and a length of empty space.
DashOffset	No	Numeric	The offset from the beginning of the line where the dash pattern begins.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset black = myImage.getColorByName("black") />
<cfset blue = myImage.getColorByName("blue") />
<cfset red = myImage.getColorByName("red") />
<cfset orange = myImage.getColorByName("orange") />
<cfset yellow = myImage.getColorByName("yellow") />
<cfset ivory = myImage.getColorByName("ivory") />
<cfset crimson = myImage.getColorByName("crimson") />
<cfset feldspar = myImage.getColorByName("feldspar") />
<cfset greenYellow = myImage.getColorByName("greenYellow") />
<cfset lightGoldenRodYellow = ¬
myImage.getColorByName("lightGoldenRodYellow") />
<cfset aliceBlue = myImage.getColorByName("aliceBlue") />
<cfset lavender = myImage.getColorByName("lavender") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(black) />

<!--- create a new new image --->
<cfset myImage.createImage(300, 300) />

<!--- draw the default line into the image --->
<cfset myImage.drawLine(0,20, 300,20) />

<!--- draw some examples of stroke weights and colors --->
<cfset myImage.setStroke(2, blue) />
<cfset myImage.drawLine(0,40, 300,40) />

<cfset myImage.setStroke(4, red) />
<cfset myImage.drawLine(0,60, 300,60) />

<cfset myImage.setStroke(8, orange) />
<cfset myImage.drawLine(0,80, 300,80) />

<!--- draw three lines demonstrating line caps --->
```

```
<cfset myImage.setStroke(8, yellow, "Butt") />
<cfset myImage.drawLine(10,100, 290,100) />
<cfset myImage.setStroke(8, ivory, "Round") />
<cfset myImage.drawLine(10,120, 290,120) />
<cfset myImage.setStroke(8, crimson, "Square") />
<cfset myImage.drawLine(10,140, 290,140) />

<!--- draw three polygon demponstrating joins --->
<cfset myImage.setStroke(8, feldspar, "Butt", "Bevel") />
<cfset myPolygon = myImage.createPolygon() />
<cfset myImage.addPolygonPoint(myPolygon, 20, 160) />
<cfset myImage.addPolygonPoint(myPolygon, 20, 220) />
<cfset myImage.addPolygonPoint(myPolygon, 80, 190) />
<cfset myImage.drawPolygon(myPolygon) />

<cfset myImage.setStroke(8, greenYellow, "Butt", "Miter") />
<cfset myPolygon = myImage.createPolygon() />
<cfset myImage.addPolygonPoint(myPolygon, 120, 160) />
<cfset myImage.addPolygonPoint(myPolygon, 120, 220) />
<cfset myImage.addPolygonPoint(myPolygon, 180, 190) />
<cfset myImage.drawPolygon(myPolygon) />

<cfset myImage.setStroke(8, lightGoldenRodYellow, "Butt", "Round") />
<cfset myPolygon = myImage.createPolygon() />
<cfset myImage.addPolygonPoint(myPolygon, 220, 160) />
<cfset myImage.addPolygonPoint(myPolygon, 220, 220) />
<cfset myImage.addPolygonPoint(myPolygon, 280, 190) />
<cfset myImage.drawPolygon(myPolygon) />

<!--- draw two examples of dashed lines --->
<cfset myImage.setStroke(8, aliceBlue, "butt", "miter", 10, ¬
"20,20,10,10,5,5", 0) />
<cfset myImage.drawLine(0,260, 300,260) />
<cfset myImage.setStroke(6, lavender, "round", "miter", 10, ¬
"16,16", 0) />
<cfset myImage.drawLine(0,280, 300,280) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\lines.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/lines.png">
</p>
```

Results



See Also

createColor() getColorByName()

setTransparency()

Description

The setTransparency() method sets the transparency of items drawn into an image.

Syntax

setTransparency(transparency)

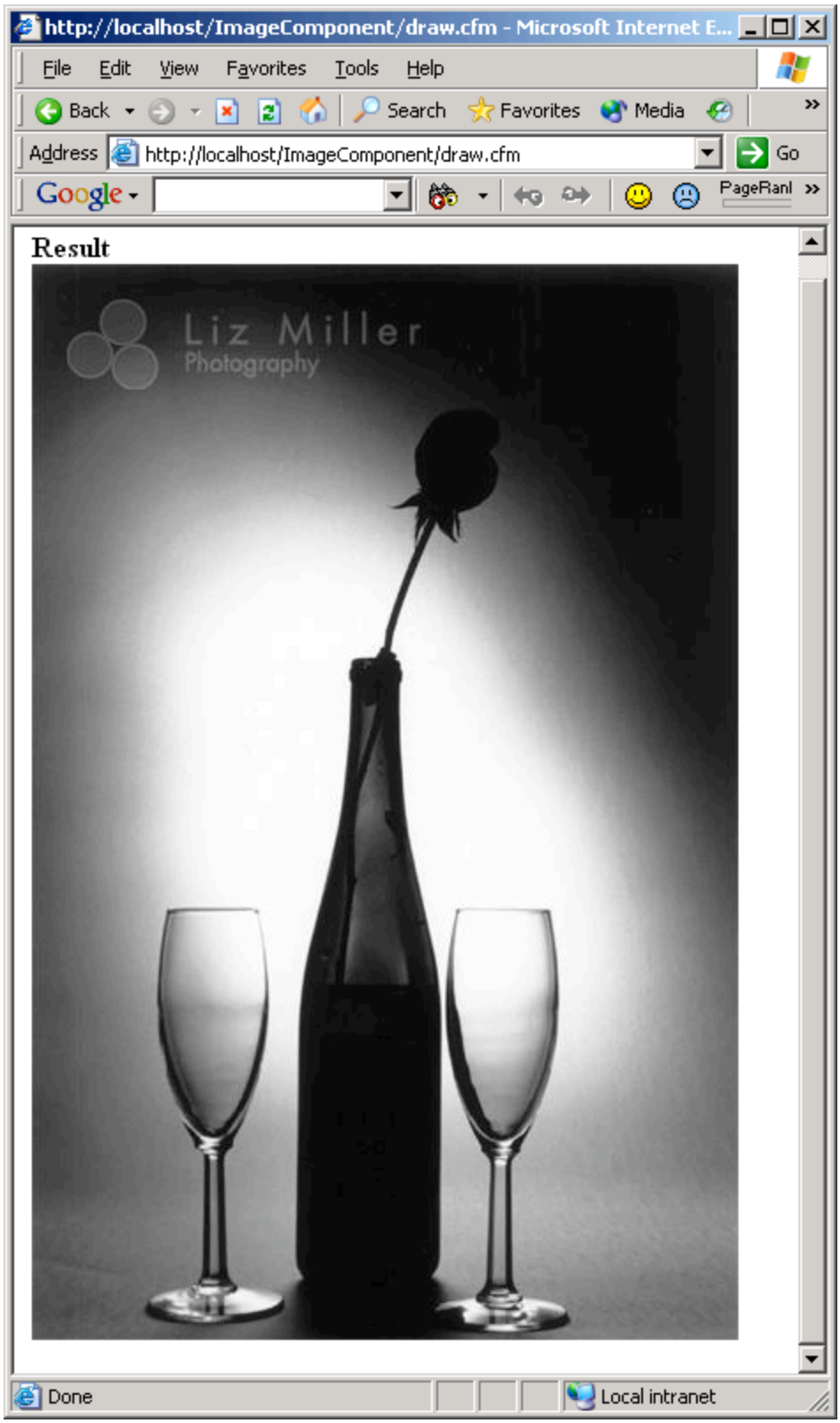
Parameter	Required	Type	Description
Transparency	Yes	Numeric	The percent transparent elements drawn into the image are.

Example

The following example draws a company logo into the upper left corner of an image. The logo has an alpha channel which is used to composite the two images so that the image shows through the transparent portions of the logo. Additionally, the logo is made to be slightly transparent using setTransparency().

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- open a new image --->
<cfset myImage.readImage("d:\examples\wineRose.jpg") />

<!--- set the transparency used when drawing into the image --->
<cfset myImage.setTransparency(25) />

<!--- draw an image into the image --->
<cfset myImage.drawImage("d:\examples\logo.png", 20, 20) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\wineRoseLogo.jpg", "jpg") />

<!--- output the images --->
<b>Result</b><br>
<img src="/examples/wineRoseLogo.jpg">
```

Results


http://localhost/ImageComponent/draw.cfm - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ImageComponent/draw.cfm
Go
Google
PageRanl
Result
Liz Miller
Photography
Done
Local intranet

Colors and Fills

createColor()

Description

The createColor() method is used to create a new Color Object. The color is an ARGB color, or a color made up Red, Green, Blue and Alpha colors. Color Objects are used throughout the Image Component to set stroke and fill colors and much more.

Syntax

Color = createColor(red, green, blue, alpha)

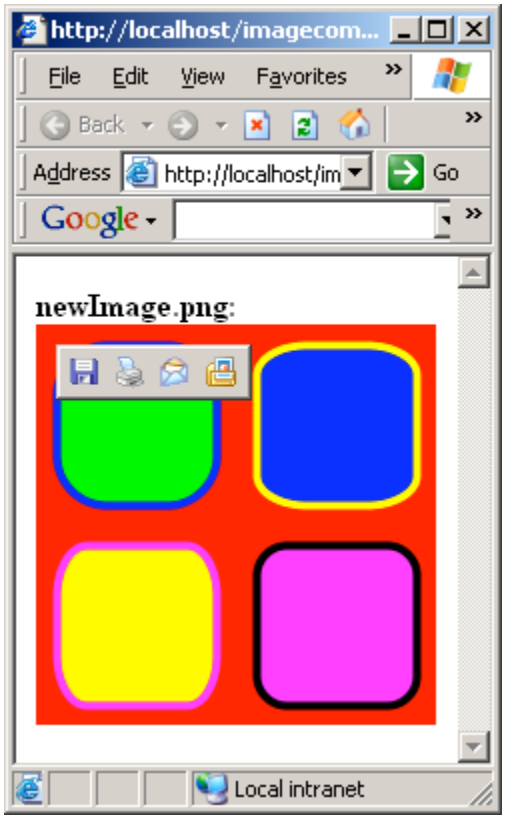
Parameter	Required	Type	Description
Red	Yes	Numeric	The value for the color's red channel. Must be between 0 and 255.
Green	Yes	Numeric	The value for the color's green channel. Must be between 0 and 255.
Blue	Yes	Numeric	The value for the color's blue channel. Must be between 0 and 255.
Alpha	No	Numeric	The value for the color's aplha channel. Must be between 0 and 255. Defaults to 255, completely opaque.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset red = myImage.createColor(255,0,0) />
<cfset green = myImage.createColor(0,255,0) />
<cfset blue = myImage.createColor(0,0,255) />
<cfset yellow = myImage.createColor(255,255,0) />
<cfset purple = myImage.createColor(255,0,255) />
<cfset black = myImage.createColor(0,0,0) />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(red) />

<!--- create a new image --->
<cfset myImage.createImage(200, 200) />

<!--- draw four rounded rectanges into the new image --->
<!--- round rectangle 1 --->
<cfset myImage.setFill(green) />
<cfset myImage.setStroke(4, blue) />
<cfset myImage.drawRoundRectangle(10, 10, 80, 80, 50, 50) />
```

```
<!--- round rectangle 2 --->
<cfset myImage.setFill(blue) />
<cfset myImage.setStroke(4, yellow) />
<cfset myImage.drawRoundRectangle(110, 10, 80, 80, 50, 30) />

<!--- round rectangle 3 --->
<cfset myImage.setFill(yellow) />
<cfset myImage.setStroke(4, purple) />
<cfset myImage.drawRoundRectangle(10, 110, 80, 80, 30, 50) />

<!--- round rectangle 4 --->
<cfset myImage.setFill(purple) />
<cfset myImage.setStroke(4, black) />
<cfset myImage.drawRoundRectangle(110, 110, 80, 80, 30, 30) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>newImage.png:</b><br>
<img src="/examples/newImage.png">
</p>
```

Results



See Also

getColorByName() setStroke() setFill()

createGradient()

Description

The createGradient() method creates and returns a Gradient Object which can be used for filling shapes. Two coordinates are used to set the angle of the gradient. Both of these coordinates are in the image plane, not within the plane of the shape drawn.

Syntax

Gradient = createGradient(x1,y1,x2,y2,Color1,Color2,cycle)

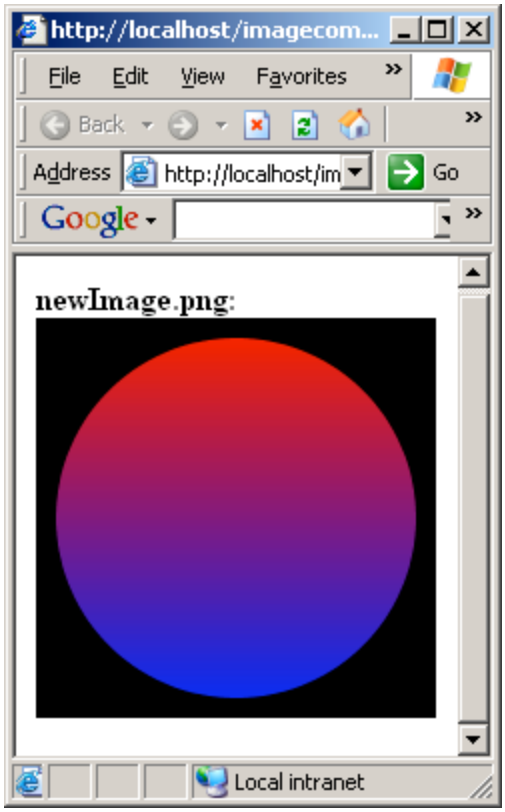
Parameter	Required	Type	Description
x1	Yes	Numeric	Sets first X coordinate used to set the angle of the gradient.
y1	Yes	Numeric	Sets first Y coordinate used to set the angle of the gradient.
x2	Yes	Numeric	Sets second X coordinate used to set the angle of the gradient.
y2	Yes	Numeric	Sets second Y coordinate used to set the angle of the gradient
Color1	Yes	Color Object	Sets the first color of the gradient.
Color2	Yes	Color Object	Sets the second color of the gradient.
Cycle	No	Boolean	Indicates if the gradient should cycle (true) or to extend each color indefinitely in either direction after the fill (false). *Default's to False*

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset red = myImage.createColor(255,0,0) />
<cfset blue = myImage.createColor(0,0,255) />

<!--- create a new image --->
<cfset myImage.createImage(200, 200) />

<!--- create a new gradient --->
<cfset myGraident = myImage.createGradient(0,0, 0,200, red, blue) />

<!--- draw a grediated circle into the image --->
<cfset myImage.setFill(myGraident) />
<cfset myImage.drawOval(10, 10, 180, 180) />

<!--- output the image in PNG format --->
<cfset myImage.writeImage("d:\examples\newImage.png", "png") />

<!--- the new image --->
<p>
<b>newImage.png:</b><br>
```

```
<img src="/examples/newImage.png">
</p>
```

Results



See Also

createColor() getColorByName()

createTexture()

Description

The createTexture() method creates a new repeating Texture Object which can be used to fill shapes drawn into the image.

Syntax

Texture = createTexture(textureFilePath, x, y, width, height)

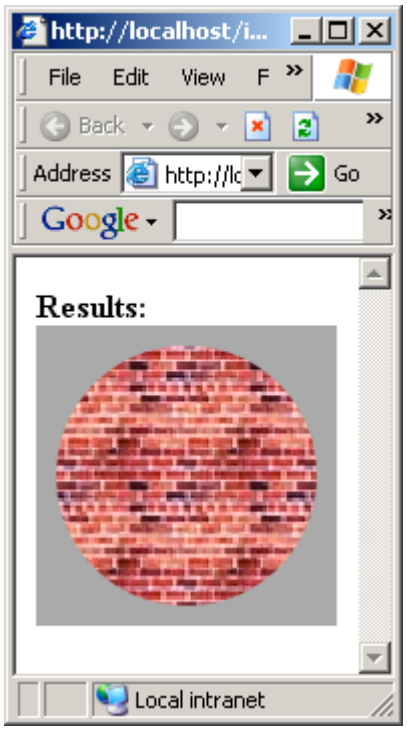
Parameter	Required	Type	Description
TextureFilePath	Yes	String	The path to the image file to use as a texture.
X	No	Numeric	The X coordinate upper left corner to use as the origin of the texture.
Y	No	Numeric	The Y coordinate upper left corner to use as the origin of the texture.
Width	No	Numeric	The width to set the texture image to.
Height	No	Numeric	The height to set the texture image

			to.

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component","Image") />

<!--- create some colors --->
<cfset darkgray = myImage.getColorByName("darkGray") />

<!--- set the background color --->
<cfset myImage.setBackgroundColor(darkgray) />

<!--- create a new new image --->
<cfset myImage.createImage(150, 150) />

<!--- draw a circle with a texture fill --->
<cfset texture = myImage.createTexture("d:\examples\bricks.jpg", 0, 0, ¬
60, 60) />
<cfset myImage.setFill(texture) />
<cfset myImage.drawOval(10, 10, 130, 130) />

<!--- output the new image --->
<cfset myImage.writeImage("d:\examples\texture.png", "png") />

<!--- the new images --->
<p>
<b>Results:</b><br>
<img src="/examples/texture.png">
</p>
```

Results



See Also

setFill() createGradient()

getColorByName()

Description

The getColorByName() method returns a color based on a name passed into the method. Call the getColorList() method to get a list of all available named colors. Color Objects are used throughout the Image Component to set stroke and fill colors and much more.

Syntax

Color = getColorByName(name)

Parameter	Required	Type	Description
Name	Yes	String	Any supported color name.

Example

```
<cfset darkSeaGreen = myImage.getColorByName("DarkSeaGreen") />
```

See Also

createColor() setStroke() setFill() getColorList()

getColorFromPixel()

Description

The getColorFromPixel() method is used to return the color of the pixel at the specified coordinates. This color can be used with the setStroke() and setFill() methods.

Syntax

ColorList = getColorFromPixel(x, y)

Parameter	Required	Type	Description
X	Yes	Numeric	The X coordinate for to sample the color from.
Y	Yes	Numeric	The Y coordinate for to sample the color from.

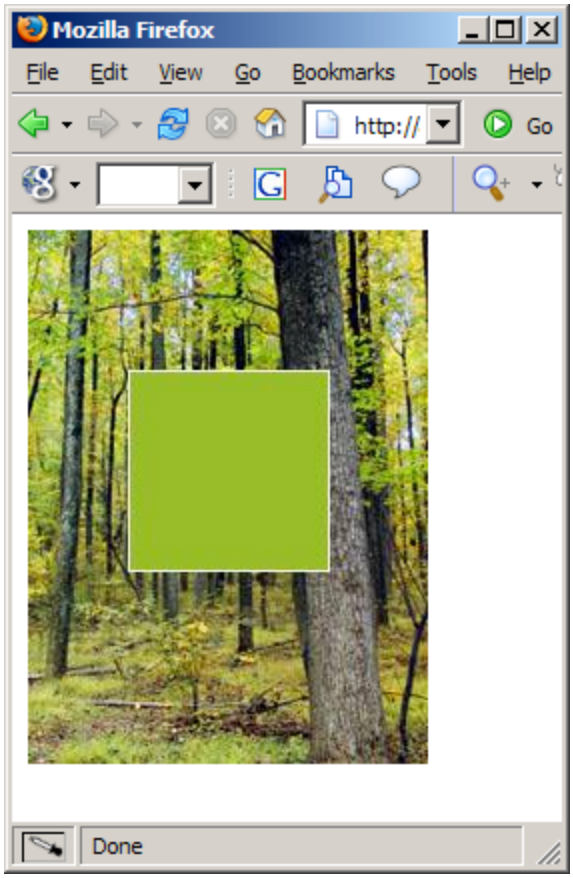
Example

This example demonstrates getting a color from a random pixel in the image and using it to fill a square drawn in to the image.

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image") />

<!--- read an image --->
<cfset myImage.readImage(expandPath("forest1.jpg")) />

<!--- get a color from the image --->
<cfset myColor = myImage.getColorFromPixel(randRange(1, ¬
myImage.getWidth()), randRange(1, myImage.getHeight())) />
```

```
<!--- set the current color to myColor --->
<cfset myImage.setFill(myColor) />

<!--- draw a shape on the image --->
<cfset myImage.drawRectangle(50, 70, 100, 100) />

<!--- write the image --->
<cfset myImage.writeImage(expandPath("forest2.jpg"), "jpg") />

<!--- display the new image --->
<img src="forest2.jpg">
```

Results



See Also

createColor() setStroke() setFill()

getColorList()

Description

The getColorList() method returns a comma delimited list of named colors which can be used in the getColorByName() method.

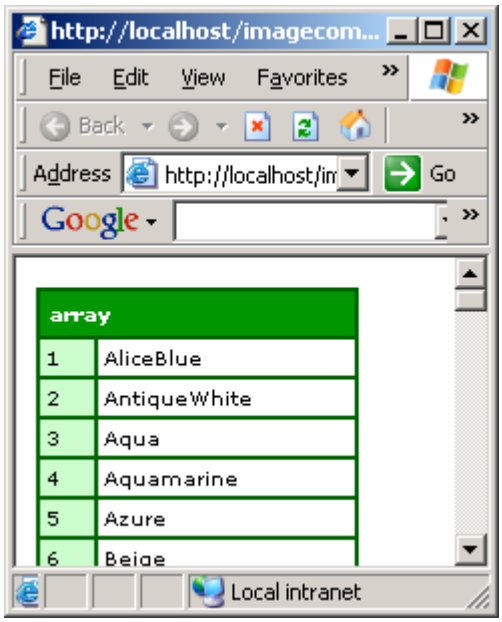
Syntax

ColorList = getColorList()

Example

```
<!--- create the object --->
<cfset myImage = CreateObject("Component", "Image") />

<!--- get the color list --->
<cfset colorList = myImage.getColorList() />

<!--- convert the list of colors to an array and show them --->
<cfdump var="#ListToArray(colorList)#" />
```

Results



Extensibility

Overview

The Alagad Image Component uses a Java BufferedImage object internally to store image data. You have access to this data via the getBufferedImage() and setBufferedImage() methods. This means that you can grab the raw data the Image Component is working with, manipulate it directly and, optionally, you can put it back into the Image Component. This also means that anything that can be done to a BufferedImage can be done with help from the Image Component.

This capability is particularly useful when used in conjunction with another CFC which extends the Alagad Image Component. For more information on extending the Alagad Image Component see the section Image Component Extensibility.

getBufferedImage()

Description

The getBufferedImage() method is used to get the Java BufferedImage object which the Image Component uses internally to store image data.

Syntax
BufferedImage = getBufferedImage()

Example
```
<cfset myBufferedImage = myImage.getBufferedImage() />
```

See Also
Image Component Extensibility.

setBufferedImage()
The setBufferedImage() method is used to set the Java BufferedImage object which the Image Component uses internally to store image data.

Syntax
getBufferedImage(BufferedImage)

Example
```
<cfset myImage.getBufferedImage(myBufferedImage) />
```

See Also
Image Component Extensibility.

Image Component Extensibility
Although the Image Component provides all of the most requested functionality, from time to time users have needs for functionality which is not avaliable. However, the Alagad Image Component provides mechanisms for extending and enhancing the capabilities of the Alagad Image Component.

The following section is a reprint of an article initially published on DougHughes.net:

Adding Your Own Functionality to the Alagad Image Component
Although the Image Component has all of the most requested functionality, from time to time I get requests for features I can't or won't implement. Lucky for these users, they can easy to add features to the Image Component themselves!

I tend to get two types of feature questions or requests. These can be categorized as follows:

- Does the Image Component write to GIF Files?
- Does the Image Component support a particular type of functionality in a particular way?

Let me address these below.

Does the Image Component write to GIF Files

The short answer to this question is no. (But keep reading!) The Image Component leverages classes provided by the Java platform underlying ColdFusion. The classes which come with Java do not support writing to GIF files so the Image Component does not natively support writing to GIF files.

But wait! You can still write to GIF files, it's just a little harder.

The Image Component uses a Java BufferedImage internally to store image data. However, you as a user of the Image Component, have access to this data via the getBufferedImage() and setBufferedImage() methods. This means that you can grab the raw data the Image Component is working with, manipulate it and, optionally, put it back into the Image Component. This also means that anything you can do to a BufferedImage you can do with help from the Image Component.

How does this relate to writing GIF images? Well, today I stumbled across this page: http://www.acme.com/java/software/Acme.JPM.Encoders.ImageEncoder.html. This page shows the Java Docs for a freeware Java GifEncoder class. This particular class is part of a larger package provided by ACME Laboratories (no association with Wile E. Coyote). Read the website for more information about ACME.

You may note that the GifEncoder method accepts an Image object and an OutputStream object. Coincidently, Image is the superclass for BufferedImage! An OutputStream is also quite easy to create from ColdFusion. I'll show you how in a second.

If you download the ACME package and extract it into a Java class path you will have the beginnings of the ability to write GIF images from the Alagad Image Component. I am running ColdFusion on top of JRun, so I extracted the ACME package into my c:\JRun4\lib directory. You will need to figure out where to extract this package yourself. Don't forget to restart ColdFusion before continuing.

Once ColdFusion was done restarting, I created a new CFM page and started coding. I started by reading an image:

```
<!--- create the Image.cfc and read an image --->
<cfset myImage = CreateObject("Component", "Image") />
<cfset myImage.readImage(expandPath("transparentImage.png")) />
```

The GifEncoder class's constructor requires an Image object and a Outputstream. So, I added the next few lines of code.

```
<!--- get the bufferedImage from the Image Component --->
<cfset BufferedImage = myImage.getBufferedImage() />
```

```
<!--- create a FileOutputStream (a type of OutputStream) and init to
the image to write to --->
<cfset FileOutputStream = CreateObject("Java", ¬
"java.io.FileOutputStream").init(JavaCast("string", ¬
expandPath("myOtherImage.gif"))) />
```

Once this code was in place I had everything I needed to write a Gif except a GifEncoder! This next line helped out with that:

```
<cfset GifEncoder = CreateObject("Java",
"Acme.JPM.Encoders.GifEncoder").init(BufferedImage, FileOutputStream) ¬
/>
```

Now I had a GifEncoder which was configured to write my PNG image to a GIF file, "myOtherImage.gif". Just one more line of code to actually do the job:

```
<cfset GifEncoder.encode() />
```

The very first time I ran this code it worked like a charm! My transparent PNG was successfully converted to a transparent GIF. Woo Hoo!

So, that's a good example of how to use getBufferedImage() and a little creative Java-from-ColdFusion to do things which the Image Component can't itself do.

But, it gets even better! Check out the next example, and be sure to see the last section too!

Does the Image Component support a particular type of functionality in a particular way?

Once again, the short answer to this question is no. But, you guessed it; there is a way to add most any functionality to the Image Component. Here's a good question I received the other day:

> I'm watermarking images with a logo. However, I don't want to be required to calculate the location for the watermark image each time I do this. Is there any way to simply say "Draw an image in the lower right corner, or the center top of the Image?"

Well, the Image Component does provide a way to watermark images by drawing one image into another with the drawImage() method. However, you're responsible for the calculations involved in determining where the new Image should be placed.

If you wanted to place an image in the bottom center of another image you could perform the following steps:

- Read the watermark image and get its width and height
- Read the image being watermarked and get its width and height

- Subtract the height of the watermark from the height of the image being watermarked to get the Y coordinate for drawing.
- Subtract the width of the watermark from the width of the image being watermarked and divide that number by two to get the X coordinate.
- Draw the watermark into the image using the drawImage() method, passing it the coordinate information.

This could easily be modified to place the watermark in various locations, but I'll leave that to you. Here's the code implementing the steps outlined above:

```
<!--- create the two image objects --->
<cfset myImage = CreateObject("Component", "Image") />
<cfset myWatermark = CreateObject("Component", "Image") />

<!--- read the image and watermark --->
<cfset myImage.readImage(expandPath("forest1.jpg")) />
<cfset myWatermark.readImage(expandPath("watermark.png")) />

<!--- get the image's width and height --->
<cfset imgWidth = myImage.getWidth() />
<cfset imgHeight = myImage.getHeight() />

<!--- get the watermark's width and height --->
<cfset waterWidth = myWatermark.getWidth() />
<cfset waterHeight = myWatermark.getHeight() />

<!--- get the coordinates to draw into --->
<cfset x = Round((imgWidth - waterWidth)/2) />
<cfset y = imgHeight - waterHeight />

<!--- draw the watermark into the image --->
<cfset myImage.drawImage(expandPath("watermark.png"), x, y) />

<!--- write the new image to disk --->
<cfset myImage.writeImage(expandPath("watermarkedImage.jpg"), "jpg") />
```

And here's the resulting image when I run the code:

The Big Picture

Now wouldn't it be cool if you could add methods to the Image Component to do these two things? Once again, you guessed it; you can! This time it's simply a matter of extending the Image.cfc file. The following example creates a new CFC named "superImage.cfc" which extends Image.cfc and adds the ability to draw images into the bottom center of your image and allows you to save the images as a GIF file. This is all in addition to everything else the Image Component can already do!

The following code encapsulates all of the functionality discussed above into two methods in a component which extends the Image Component:

```
<cfcomponent displayname="SuperImage"
    extends="Image"
    hint="I add a few methods to the Alagad Image Component.">

    <cffunction name="writeGif" access="public" hint="I write a gif ¬
Image!" output="false" returntype="void">
        <cfargument name="path" hint="I am the path to write the GIF ¬
image to." required="yes" type="string" />
        <cfset var BufferedImage = getBufferedImage() />
        <cfset var FileOutputStream = CreateObject("Java", ¬
"java.io.FileOutputStream").init(JavaCast("string", arguments.path)) />
        <cfset var GifEncoder = CreateObject("Java", ¬
"Acme.JPM.Encoders.GifEncoder").init(BufferedImage, FileOutputStream) />

        <!--- write the gif image! --->
        <cfset GifEncoder.encode() />
    </cffunction>

    <cffunction name="drawImageInCenterBottom" access="public" ¬
hint="I draw an image into the center bottom of the current Image." ¬
output="false"  returntype="void">
        <cfargument name="path" hint="I am the path of the image to ¬
draw" required="yes" type="string" />
        <cfset var myWatermark = CreateObject("Component",  ¬
"SuperImage") />
        <!--- get the image's width and height --->
        <cfset var imgWidth = getWidth() />
        <cfset var imgHeight = getHeight() />
        <!--- declare the watermark's width and height --->
        <cfset var waterWidth = 0 />
        <cfset var waterHeight = 0 />
        <!--- declare x and y --->
        <cfset x = 0 />
        <cfset y = 0 />

        <!--- read the watermark --->
        <cfset myWatermark.readImage(arguments.path) />

        <!--- get the watermark's width and height --->
        <cfset waterWidth = myWatermark.getWidth() />
        <cfset waterHeight = myWatermark.getHeight() />
```

```
        <!--- get the coordinates to draw into --->
        <cfset x = Round((imgWidth - waterWidth)/2) />
        <cfset y = imgHeight - waterHeight />

        <!--- draw the watermark into the image --->
        <cfset drawImage(arguments.path, x, y) />
    </cffunction>

</cfcomponent>
```

The following is some code using our new SuperImage component:

```
<cfset mySuperImage = CreateObject("Component", "SuperImage") />
<cfset mySuperImage.readImage(expandPath("someCrazyImage.png")) />
<cfset
mySuperImage.drawImageInCenterBottom(expandPath("watermark.png")) />
<cfset mySuperImage.writeGif(expandPath("myGif.gif")) />
```

Now how cool is that?! Can you do that with any competing image products? I think not!