

java.awt.image Reference

22.1 *AreaAveragingScaleFilter* ★

Description

The `AreaAveragingScaleFilter` class scales an image using a simple smoothing algorithm.

Class Definition

```
public class java.awt.image.AreaAveragingScaleFilter
    extends java.awt.image.ReplicateScaleFilter {

    // Constructor
    public AreaAveragingScaleFilter (int width, int height);

    // Instance Methods
    public void setHints (int hints);
    public void setPixels (int x, int y, int w, int h, ColorModel model,
        byte[] pixels, int off, int scansize);

    public void setPixels (int x, int y, int w, int h, ColorModel model,
        int[] pixels, int off, int scansize);
}
```

Constructor

AreaAveragingScaleFilter

```
public AreaAveragingScaleFilter (int width, int height)
```

Parameters *width* Width of scaled image.
 height Height of scaled image.

Description Constructs an `AverageScaleFilter` that scales the original image to the specified size.

Instance Methods

setHints

```
public void setHints (int hints)
```

Parameters *hints* Flags indicating how data will be delivered.

Overrides `ImageFilter.setHints(int)`

Description Gives this filter hints about how data will be delivered.

setPixels

```
public void setPixels (int x, int y, int w, int h,  
ColorModel model, byte[] pixels, int off, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.

y y-coordinate of top-left corner of pixel data delivered with this method call.

w Width of the rectangle of pixel data delivered with this method call.

h Height of the rectangle of pixel data delivered with this method call.

model Color model of image data.

pixels Image data.

off Offset from beginning of the pixels array.

scansize Size of each line of data in pixels array.

Overrides `ReplicateScaleFilter.setPixels(int, int, int, int, ColorModel, byte[], int, int)`

Description Receives a rectangle of image data from the `ImageProducer`; scales these pixels and delivers them to any `ImageConsumers`.

```
public void setPixels (int x, int y, int w, int h,  
ColorModel model, int[] pixels, int off, int scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>w</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>h</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>off</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Overrides	ReplicateScaleFilter.setPixels(int, int, int, int, ColorModel, int[], int, int)	
Description	Receives a rectangle of image data from the ImageProducer; scales these pixels and delivers them to any ImageConsumers.	

See Also

ColorModel, ReplicateScaleFilter

22.2 ColorModel**Description**

The abstract ColorModel class defines the way a Java program represents colors. It provides methods for extracting different color components from a pixel.

Class Definition

```
public class java.awt.image.ColorModel
    extends java.lang.Object {

    // Variables
    protected int pixel_bits;

    // Constructors
    public ColorModel (int bits);

    // Class Methods
    public static ColorModel getRGBdefault();

    // Instance Methods
    public void finalize(); ★
    public abstract int getAlpha (int pixel);
    public abstract int getBlue (int pixel);
```

```
public abstract int getGreen (int pixel);
public int getPixelSize();
public abstract int getRed (int pixel);
public int getRGB (int pixel);
}
```

ProtectedVariables

pixel_bits

```
protected int pixel_bits
```

The `pixel_bits` variable saves the `ColorModel`'s bits setting (the total number of bits per pixel).

Constructors

ColorModel

```
public ColorModel (int bits)
```

Parameters *bits* The number of bits required per pixel using this model.

Description Constructs a `ColorModel` object.

Class Methods

getRGBdefault

```
public static ColorModel getRGBdefault()
```

Returns The default `ColorModel` format, which uses 8 bits for each of a pixel's color components: alpha (transparency), red, green, and blue.

Instance Methods

finalize

```
public void finalize() ★
```

Overrides `Object.finalize()`

Description Cleans up when this object is garbage collected.

getAlpha

```
public abstract int getAlpha (int pixel)
```

Parameters *pixel* A pixel encoded with this `ColorModel`.

Returns The current alpha setting of the pixel.

getBlue

```
public abstract int getBlue (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current blue setting of the pixel.

getGreen

```
public abstract int getGreen (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current green setting of the pixel.

getPixelSize

```
public int getPixelSize()
```

Returns The current pixel size for the color model.

getRed

```
public abstract int getRed (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current red setting of the pixel.

getRGB

```
public int getRGB (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current combined red, green, and blue settings of the pixel.

Description Gets the color of *pixel* in the default RGB color model.

See Also

DirectColorModel, IndexColorModel, Object

22.3 CropImageFilter

Description

The CropImageFilter class creates a smaller image by cropping (i.e., extracting a rectangular region from) a larger image.

Class Definition

```
public class java.awt.image.CropImageFilter
    extends java.awt.image.ImageFilter {

    // Constructors
    public CropImageFilter (int x, int y, int width, int height);

    // Instance Methods
    public void setDimensions (int width, int height);
    public void setPixels (int x, int y, int width, int height, ColorModel model,
        byte[] pixels, int offset, int scansize);
    public void setPixels (int x, int y, int width, int height, ColorModel model,
        int[] pixels, int offset, int scansize);
    public void setProperties (Hashtable properties);
}
```

Constructors

CropImageFilter

```
public CropImageFilter (int x, int y, int width, int
height)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of piece to crop.
	<i>y</i>	y-coordinate of top-left corner of piece to crop.
	<i>width</i>	Width of image to crop.
	<i>height</i>	Height of image to crop.

Description	Constructs a <code>CropImageFilter</code> that crops the specified region from the original image.
-------------	--

Instance Methods

setDimensions

```
public void setDimensions (int width, int height)
```

Parameters	<i>width</i>	Ignored parameter.
	<i>height</i>	Ignored parameter.

Overrides	<code>ImageFilter.setDimensions(int, int)</code>
-----------	--

Description	Called with the original image's dimensions; these dimensions are ignored. The method in turn calls the <code>ImageConsumer</code> with the dimensions of the cropped image.
-------------	--

setPixels

```
public void setPixels (int x, int y, int width, int
height, ColorModel model, byte[] pixels, int offset, int
scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.

Overrides ImageFilter.setPixels(int, int, int, int, ColorModel, byte[], int, int)

Description Receives a rectangle of image data from the ImageProducer; crops these pixels and delivers them to any ImageConsumers.

```
public void setPixels (int x, int y, int width, int
height, ColorModel model, int[] pixels, int offset, int
scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.

Overrides ImageFilter.setPixels(int, int, int, int, ColorModel, int[], int, int)

Description Receives a rectangle of image data from the ImageProducer; crops these pixels and delivers them to any ImageConsumers.

setProperties

public void setProperties (Hashtable properties)

Parameters *properties* The properties for the image.

Overrides ImageFilter.setProperties(Hashtable)

Description Adds the “cropect” image property to the properties list.

See Also

ColorModel, Hashtable, ImageFilter

22.4 DirectColorModel

Description

The DirectColorModel class provides a ColorModel that specifies a translation between pixels and alpha, red, green, and blue component values, where the color values are embedded directly within the pixel.

Class Definition

```
public class java.awt.image.DirectColorModel
    extends java.awt.image.ColorModel {

    // Constructors
    public DirectColorModel (int bits, int redMask, int greenMask,
        int blueMask);
    public DirectColorModel (int bits, int redMask, int greenMask,
        int blueMask,
        int alphaMask);

    // Instance Methods
    public final int getAlpha (int pixel);
    public final int getAlphaMask();
    public final int getBlue (int pixel);
    public final int getBlueMask();
    public final int getGreen (int pixel);
    public final int getGreenMask();
    public final int getRed (int pixel);
    public final int getRedMask();
    public final int getRGB (int pixel);
}
```


Constructors

DirectColorModel

```
public DirectColorModel (int bits, int redMask, int
greenMask, int blueMask)
```

Parameters	<i>bits</i>	The number of bits required per pixel of an image using this model.
	<i>redMask</i>	The location of the red component of a pixel.
	<i>greenMask</i>	The location of the green component of a pixel.
	<i>blueMask</i>	The location of the blue component of a pixel.
Throws	IllegalArgumentException	If the mask bits are not contiguous or overlap.
Description	Constructs a DirectColorModel object with the given size and color masks; the alpha (transparency) component is not used.	

```
public DirectColorModel (int bits, int redMask, int
greenMask, int blueMask, int alphaMask)
```

Parameters	<i>bits</i>	The number of bits required per pixel of an image using this model.
	<i>redMask</i>	The location of the red component of a pixel.
	<i>greenMask</i>	The location of the green component of a pixel.
	<i>blueMask</i>	The location of the blue component of a pixel.
	<i>alphaMask</i>	The location of the alpha component of a pixel.
Throws	IllegalArgumentException	If the mask bits are not contiguous or overlap.
Description	Constructs a DirectColorModel object with the given size and color masks.	

Instance Methods

getAlpha

```
public final int getAlpha (int pixel)
```

Parameters	<i>pixel</i>	A pixel encoded with this ColorModel.
Returns	The current alpha setting of the pixel.	
Overrides	ColorModel.getAlpha(int)	

getAlphaMask

```
public final int getAlphaMask()
```

Returns The current alpha mask setting of the color model.

getBlue

```
public final int getBlue (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current blue setting of the pixel.

Overrides ColorModel.getBlue(int)

getBlueMask

```
public final int getBlueMask()
```

Returns The current blue mask setting of the color model.

getGreen

```
public final int getGreen (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current green setting of the pixel.

Overrides ColorModel.getGreen(int)

getGreenMask

```
public final int getGreenMask()
```

Returns The current green mask setting of the color model.

getRed

```
public final int getRed (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current red setting of the pixel.

Overrides ColorModel.getRed(int)

getRedMask

```
public final int getRedMask()
```

Returns The current red mask setting of the color model.

getRGB

```
public final int getRGB (int pixel)
```

Parameters *pixel* A pixel encoded with this `ColorModel`.

Returns The current combined red, green, and blue settings of the pixel.

Overrides `ColorModel.getRGB(int)`

Description Gets the color of `pixel` in the default RGB color model.

See Also

`ColorModel`

22.5 FilteredImageSource

Description

The `FilteredImageSource` class acts as glue to put an original `ImageProducer` and `ImageFilter` together to create a new image. As the `ImageProducer` for the new image, `FilteredImageSource` is responsible for registering image consumers for the new image.

Class Definition

```
public class java.awt.image.FilteredImageSource
    extends java.lang.Object
    implements java.awt.image.ImageProducer {

    // Constructors
    public FilteredImageSource (ImageProducer original,
        ImageFilter filter);

    // Instance Methods
    public synchronized void addConsumer (ImageConsumer ic);
    public synchronized boolean isConsumer (ImageConsumer ic);
    public synchronized void removeConsumer (ImageConsumer ic);
    public void requestTopDownLeftRightResend (ImageConsumer ic);
    public void startProduction (ImageConsumer ic);
}
```

Constructors

FilteredImageSource

```
public FilteredImageSource (ImageProducer original,
    ImageFilter filter)
```

Parameters	<i>original</i>	An ImageProducer that generates the image to be filtered.
	<i>filter</i>	The ImageFilter to use to process image data delivered by original.
Description	Constructs a FilteredImageSource object to filter an image generated by an ImageProducer.	

Class Methods

addConsumer

```
public synchronized void addConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer interested in receiving the new image.

Implements ImageProducer.addConsumer (ImageConsumer)

Description Registers an ImageConsumer as interested in Image information.

isConsumer

```
public synchronized boolean isConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to check.

Returns true if ImageConsumer is registered with this ImageProducer, false otherwise.

Implements ImageProducer.isConsumer (ImageConsumer)

removeConsumer

```
public synchronized void removeConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to remove.

Implements ImageProducer.removeConsumer (ImageConsumer)

Description Removes an ImageConsumer from the registered consumers for this ImageProducer.

requestTopDownLeftRightResend

```
public void requestTopDownLeftRightResend (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to communicate with.

Implements ImageProducer.requestTopDownLeftRightResend()

Description Requests the retransmission of the Image data in top-down, left-to-right order.

startProduction

```
public void startProduction (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to communicate with.

Implements ImageProducer.startProduction(ImageConsumer)

Description Registers ImageConsumer as interested in Image information and tells ImageProducer to start creating the filtered Image data immediately.

See Also

ImageFilter, ImageConsumer, ImageProducer, Object

22.6 ImageConsumer

Description

ImageConsumer is an interface that provides the means to consume pixel data and render it for display.

Interface Definition

```
public abstract interface java.awt.image.ImageConsumer {

    // Constants
    public final static int COMPLETESCANLINES;
    public final static int IMAGEABORTED;
    public final static int IMAGEERROR;
    public final static int RANDOMPIXELORDER;
    public final static int SINGLEFRAME;
    public final static int SINGLEFRAMEDONE;
    public final static int SINGLEPASS;
    public final static int STATICIMAGEDONE;
    public final static int TOPDOWNLEFTRIGHT;

    // Interface Methods
    public abstract void imageComplete (int status);
    public abstract void setColorModel (ColorModel model);
    public abstract void setDimensions (int width, int height);
    public abstract void setHints (int hints);
    public abstract void setPixels (int x, int y, int width, int height,
        ColorModel model, byte[] pixels, int offset, int scansize);
    public abstract void setPixels (int x, int y, int width, int height,
        ColorModel model, int[] pixels, int offset, int scansize);
    public abstract void setProperties (Hashtable properties);
}
```

Constants

COMPLETESCANLINES

```
public final static int COMPLETESCANLINES
```

Hint flag for the `setHints(int)` method; indicates that the image will be delivered one or more scanlines at a time.

IMAGEABORTED

```
public final static int IMAGEABORTED
```

Status flag for the `imageComplete(int)` method indicating that the loading process for the image aborted.

IMAGEERROR

```
public final static int IMAGEERROR
```

Status flag for the `imageComplete(int)` method indicating that an error happened during image loading.

RANDOMPIXELORDER

```
public final static int RANDOMPIXELORDER
```

Hint flag for the `setHints(int)` method; indicates that the pixels will be delivered in no particular order.

SINGLEFRAME

```
public final static int SINGLEFRAME
```

Hint flag for the `setHints(int)` method; indicates that the image consists of a single frame.

SINGLEFRAMEDONE

```
public final static int SINGLEFRAMEDONE
```

Status flag for the `imageComplete(int)` method indicating a single frame of the image has loaded.

SINGLEPASS

```
public final static int SINGLEPASS
```

Hint flag for the `setHints(int)` method; indicates that each pixel will be delivered once (i.e., the producer will not make multiple passes over the image).

STATICIMAGEDONE

```
public final static int STATICIMAGEDONE
```

Status flag for the `imageComplete(int)` method indicating that the image has fully and successfully loaded, and that there are no additional frames.

TOPDOWNLEFTRIGHT

```
public final static int TOPDOWNLEFTRIGHT
```

Hint flag for the `setHints(int)` method; indicates that pixels will be delivered in a top to bottom, left to right order.

Interface Methods**imageComplete**

```
public abstract void imageComplete (int status)
```

Parameters *status* Image loading status flags.

Description Called when the image, or a frame of an image sequence, is complete to report the completion status.

setColorModel

```
public abstract void setColorModel (ColorModel model)
```

Parameters *model* The color model for the image.

Description Tells the ImageConsumer the color model used for most of the pixels in the image.

setDimensions

```
public abstract void setDimensions (int width, int height)
```

Parameters *width* Width for image.

height Height for image.

Description Tells the consumer the image's dimensions.

setHints

```
public abstract void setHints (int hints)
```

Parameters *hints* Image consumption hints.

Description Gives the consumer information about how pixels will be delivered.

setPixels

```
public abstract void setPixels (int x, int y, int width,  
int height, ColorModel model, byte[] pixels, int offset,  
int scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.

Description Delivers a rectangular block of pixels to the image consumer.

```
public abstract void setPixels (int x, int y, int width,  
int height, ColorModel model, int[] pixels, int offset,  
int scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.

Description Delivers a rectangular block of pixels to the image consumer.

setPropertyies

```
public abstract void setProperties (Hashtable properties)
```

Parameters	<i>properties</i>	The properties for the image.
------------	-------------------	-------------------------------

Description Delivers a Hashtable that contains the image's properties.

See Also

ColorModel, Hashtable, ImageFilter, PixelGrabber, Object

22.7 ImageFilter

Description

The ImageFilter class sits between the ImageProducer and ImageConsumer as an image is being created to provide a filtered version of that image. Image filters are always used in conjunction with a FilteredImageSource. As an implementer of the ImageConsumer interface, an image filter receives pixel data from the original image's source and delivers it to another image consumer. The ImageFilter class implements a null filter (i.e., the new image is the same as the original); to produce a filter that modifies an image, create a subclass of ImageFilter.

Class Definition

```
public class java.awt.image.ImageFilter
    extends java.lang.Object
    implements java.awt.image.ImageConsumer, java.lang.Cloneable {

    // Variables
    protected ImageConsumer consumer;

    // Constructors
    public ImageFilter();

    // Instance Methods
    public Object clone();
    public ImageFilter getFilterInstance (ImageConsumer ic);
    public void imageComplete (int status);
    public void resendTopDownLeftRight (ImageProducer ip);
    public void setColorModel (ColorModel model);
    public void setDimensions (int width, int height);
    public void setHints (int hints);
    public void setPixels (int x, int y, int width, int height,
        ColorModel model, byte[] pixels, int offset, int scansize);
    public void setPixels (int x, int y, int width, int height,
        ColorModel model, int[] pixels, int offset, int scansize);
    public void setProperties (Hashtable properties);
}
```

Protected Variables

consumer

protected ImageConsumer consumer

The consumer variable is a reference to the actual ImageConsumer for the Image.

Constructors

ImageFilter

```
public ImageFilter()
```

Description Constructs an empty ImageFilter instance.

Instance Methods

clone

```
public Object clone()
```

Overrides Object.clone()

Returns A copy of the ImageFilter instance.

getFilterInstance

```
public ImageFilter getFilterInstance (ImageConsumer ic)
```

Parameters *ic* The consumer in question.

Returns A copy of the ImageFilter instance.

Description Returns the filter that will do the filtering for ic.

imageComplete

```
void imageComplete (int status)
```

Parameters *status* Image loading completion status flags.

Implements ImageConsumer.imageComplete(int)

Description Called by the ImageProducer to indicate an image's completion status. ImageFilter passes these flags to the consumer unchanged.

resendTopDownLeftRight

```
public void resendTopDownLeftRight (ImageProducer ip)
```

Parameters *ip* The ImageProducer generating the original image.

Description Called by the ImageConsumer to ask the filter to resend the image data in the top-down, left-to-right order. In ImageFilter, this method calls the same method in the ImageProducer, thus relaying the request.

setColorModel

```
void setColorModel (ColorModel model)
```

Parameters *model* The color model for the image.
 Implements `ImageConsumer.setColorModel(ColorModel)`
 Description Sets the image's color model.

setDimensions

```
void setDimensions (int width, int height)
```

Parameters *width* Width for image.
 height Height for image.
 Implements `ImageConsumer.setDimensions(int, int)`
 Description Sets the image's dimensions.

setHints

```
void setHints (int hints)
```

Parameters *hints* Image consumption hints.
 Implements `ImageConsumer.setHints(int)`
 Description Called by the `ImageProducer` to deliver hints about how the image data will be delivered. `ImageFilter` passes these hints on to the `ImageConsumer`.

setPixels

```
void setPixels (int x, int y, int width, int height,
ColorModel model, byte[] pixels, int offset, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.
 y y-coordinate of top-left corner of pixel data delivered with this method call.
 width Width of the rectangle of pixel data delivered with this method call.
 height Height of the rectangle of pixel data delivered with this method call.
 model Color model of image data.
 pixels Image data.
 offset Offset from beginning of the pixels array.
 scansize Size of each line of data in pixels array.
 Implements `ImageConsumer.setPixels(int, int, int, int, ColorModel, byte[], int, int)`

Description Delivers a rectangular block of pixels to the `ImageFilter`. `ImageFilter` passes these pixels on to the consumer unchanged.

```
void setPixels (int x, int y, int width, int height,  
ColorModel model, int[] pixels, int offset, int scansize)
```

Parameters

<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
<i>model</i>	Color model of image data.
<i>pixels</i>	Image data.
<i>offset</i>	Offset from beginning of the pixels array.
<i>scansize</i>	Size of each line of data in pixels array.

Implements `ImageConsumer.setPixels(int, int, int, int, ColorModel, int[], int, int)`

Description Delivers a rectangular block of pixels to the `ImageFilter`. `ImageFilter` passes these pixels on to the consumer unchanged.

setPropertyies

```
void setProperties (Hashtable properties)
```

Parameters *properties* The properties for the image.

Implements `ImageConsumer.setProperties(Hashtable)`

Description Initializes the image's properties. `ImageFilter` adds the property "filter" to the `Hashtable`, and passes the result on to the image consumer; the value of the property is the string returned by the filter's `toString()` method. If the property "filter" is already in the `Hashtable`, `ImageFilter` adds the string returned by its `toString()` method to the value already associated with that property.

See Also

`Cloneable`, `ColorModel`, `CropImageFilter`, `Hashtable`, `ImageConsumer`, `ImageProducer`, `Object`, `ReplicateImageFilter`, `RGBImageFilter`

22.8 *ImageObserver*

Description

`ImageObserver` is an interface that provides constants and the callback mechanism to receive asynchronous information about the status of an image as it loads.

Interface Definition

```
public abstract interface java.awt.image.ImageObserver {

    // Constants
    public static final int ABORT;
    public static final int ALLBITS;
    public static final int ERROR;
    public static final int FRAMEBITS;
    public static final int HEIGHT;
    public static final int PROPERTIES;
    public static final int SOMEBITS;
    public static final int WIDTH;

    // Interface Methods
    public abstract boolean imageUpdate (Image image, int infoflags,
        int x, int y, int width, int height);
}
```

Constants

ABORT

```
public static final int ABORT
```

The `ABORT` flag indicates that the image aborted during loading. An attempt to reload the image may succeed, unless `ERROR` is also set.

ALLBITS

```
public static final int ALLBITS
```

The `ALLBITS` flag indicates that the image has completely loaded successfully. The `x`, `y`, `width`, and `height` arguments to `imageUpdate()` should be ignored.

ERROR

```
public static final int ERROR
```

The `ERROR` flag indicates that an error happened during the image loading process. An attempt to reload the image will fail.

FRAMEBITS

```
public static final int FRAMEBITS
```

The FRAMEBITS flag indicates that a complete frame of a multi-frame image has loaded. The *x*, *y*, *width*, and *height* arguments to `imageUpdate()` should be ignored.

HEIGHT

```
public static final int HEIGHT
```

The HEIGHT flag indicates that the height information is available for an image; the image's height is in the *height* argument to `imageUpdate()`.

PROPERTIES

```
public static final int PROPERTIES
```

The PROPERTIES flag indicates that the properties information is available for an image.

SOMEBITS

```
public static final int SOMEBITS
```

The SOMEBITS flag indicates that the image has started loading and some pixels are available. The bounding rectangle for the pixels that have been delivered so far is indicated by the *x*, *y*, *width*, and *height* arguments to `imageUpdate()`.

WIDTH

```
public static final int WIDTH
```

The WIDTH flag indicates that the width information is available for an image; the image's width is in the *width* argument to `imageUpdate()`.

Interface Methods**imageUpdate**

```
public abstract boolean imageUpdate (Image image, int  
infoflags, int x, int y, int width, int height)
```

Parameters	<i>image</i>	Image that is being loaded.
	<i>infoflags</i>	The ImageObserver flags for the information that is currently available.
	<i>x</i>	Meaning depends on infoflags that are set.
	<i>y</i>	Meaning depends on infoflags that are set.

	<i>width</i>	Meaning depends on infoflags that are set.
	<i>height</i>	Meaning depends on infoflags that are set.
Returns		true if image has completed loading (successfully or unsuccessfully), false if additional information needs to be loaded.
Description		Provides the callback mechanism for the asynchronous loading of images.

See Also

Component, Image, Object

22.9 ImageProducer

Description

ImageProducer is an interface that provides the methods necessary for the production of images and the communication with classes that implement the ImageConsumer interface.

Interface Definition

```
public abstract interface java.awt.image.ImageProducer {

    // Interface Methods
    public abstract void addConsumer (ImageConsumer ic);
    public abstract boolean isConsumer (ImageConsumer ic);
    public abstract void removeConsumer (ImageConsumer ic);
    public abstract void requestTopDownLeftRightResend (ImageConsumer ic);
    public abstract void startProduction (ImageConsumer ic);
}
```

Interface Methods**addConsumer**

```
public abstract void addConsumer (ImageConsumer ic)
```

Parameters *ic* An ImageConsumer that wants to receive image data.

Description Registers an ImageConsumer as interested in image information.

isConsumer

```
public abstract boolean isConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to check.

Returns true if ImageConsumer has registered with the ImageProducer, false otherwise.

removeConsumer

```
public abstract void removeConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to remove.

Description Removes an ImageConsumer from registered consumers for this ImageProducer.

requestTopDownLeftRightResend

```
public abstract void requestTopDownLeftRightResend  
(ImageConsumer ic)
```

Parameters *ic* ImageConsumer to communicate with.

Description Requests the retransmission of the image data in top-down, left-to-right order.

startProduction

```
public abstract void startProduction (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to communicate with.

Description Registers ImageConsumer as interested in image information and tells ImageProducer to start sending the image data immediately.

See Also

FilteredImageSource, Image, ImageConsumer, ImageFilter, MemoryImageSource, Object

22.10 IndexColorModel

Description

The IndexColorModel class is a ColorModel that uses a color map lookup table (with a maximum size of 256) to convert pixel values into their alpha, red, green, and blue component parts.

Class Definition

```

public class java.awt.image.IndexColorModel
    extends java.awt.image.ColorModel {

    // Constructors
    public IndexColorModel (int bits, int size,
        byte[] colorMap, int start, boolean hasalpha);
    public IndexColorModel (int bits, int size,
        byte[] colorMap, int start, boolean hasalpha, int transparent);
    public IndexColorModel (int bits, int size,
        byte[] red, byte[] green, byte[] blue);
    public IndexColorModel (int bits, int size,
        byte[] red, byte[] green, byte[] blue, byte[] alpha);
    public IndexColorModel (int bits, int size,
        byte[] red, byte[] green, byte[] blue, int transparent);

    // Instance Methods
    public final int getAlpha (int pixel);
    public final void getAlphas (byte[] alphas);
    public final int getBlue (int pixel);
    public final void getBlues (byte[] blues);
    public final int getGreen (int pixel);
    public final void getGreens (byte[] greens);
    public final int getMapSize();
    public final int getRed (int pixel);
    public final void getReds (byte[] reds);
    public final int getRGB (int pixel);
    public final int getTransparentPixel();
}

```

Constructors

IndexColorModel

```

public IndexColorModel (int bits, int size, byte[]
colorMap, int start, boolean hasalpha)

```

Parameters	<i>bits</i>	The number of bits in a pixel.
	<i>size</i>	The number of entries in the color map. Note: this is not the size of the colorMap parameter.
	<i>colorMap</i>	Color component values in red, green, blue, alpha order; the alpha component is optional, and may not be present.
	<i>start</i>	The starting position in colorMap array.
	<i>hasalpha</i>	If hasalpha is true, alpha components are present in colorMap array.

Throws `ArrayIndexOutOfBoundsException`
 If `size` is invalid.

Description Constructs an `IndexColorModel` object with the given component settings. The size of `colorMap` must be at least $3 * \text{size} + \text{start}$, if `hasalpha` is false; if `hasalpha` is true, `colorMap.length` must be at least $4 * \text{size} + \text{start}$.

```
public IndexColorModel (int bits, int size, byte[]  
colorMap, int start, boolean hasalpha, int transparent)
```

Parameters *bits* The number of bits in a pixel.

size The number of entries in the color map. Note: this is not the size of the `colorMap` parameter.

colorMap Color component values in red, green, blue, alpha order; the alpha component is optional, and may not be present.

start The starting position in `colorMap` array.

hasalpha If `hasalpha` is true, alpha components are present in `colorMap` array.

transparent Position of `colorMap` entry for transparent pixel entry.

Throws `ArrayIndexOutOfBoundsException`
 If `size` invalid.

Description Constructs an `IndexColorModel` object with the given component settings. The size of `colorMap` must be at least $3 * \text{size} + \text{start}$, if `hasalpha` is false; if `hasalpha` is true, `colorMap.length` must be at least $4 * \text{size} + \text{start}$. The color map has a transparent pixel; its location is given by `transparent`.

```
public IndexColorModel (int bits, int size, byte[] red,  
byte[] green, byte[] blue)
```

Parameters *bits* The number of bits in a pixel.

size The number of entries in the color map.

red Red color component values.

green Green color component values.

blue Blue color component values.

Throws `ArrayIndexOutOfBoundsException`
 If `size` invalid.

Description Constructs an `IndexColorModel` object with the given component settings. There is no alpha component. The length of the `red`, `green`, and `blue` arrays must be greater than `size`.

```
public IndexColorModel (int bits, int size, byte[] red,
byte[] green, byte[] blue, byte[] alpha)
```

Parameters	<i>bits</i>	The number of bits in a pixel.
	<i>size</i>	The number of entries in the color map.
	<i>red</i>	Red color component values.
	<i>green</i>	Green color component values.
	<i>blue</i>	Blue color component values.
	<i>alpha</i>	Alpha component values.
Throws	ArrayIndexOutOfBoundsException	If <i>size</i> is invalid.
	NullPointerException	If <i>size</i> is positive and <i>alpha</i> array is null.
Description	Constructs an <code>IndexColorModel</code> object with the given component settings. The length of the <i>red</i> , <i>green</i> , <i>blue</i> , and <i>alpha</i> arrays must be greater than <i>size</i> .	

```
public IndexColorModel (int bits, int size, byte[] red,
byte[] green, byte[] blue, int transparent)
```

Parameters	<i>bits</i>	The number of bits in a pixel.
	<i>size</i>	The number of entries in the color map.
	<i>red</i>	Red color component values.
	<i>green</i>	Green color component values.
	<i>blue</i>	Blue color component values.
	<i>transparent</i>	Position of transparent pixel entry.
Throws	ArrayIndexOutOfBoundsException	If <i>size</i> is invalid.
Description	Constructs an <code>IndexColorModel</code> object with the given component settings. The length of the <i>red</i> , <i>green</i> , <i>blue</i> , and <i>alpha</i> arrays must be greater than <i>size</i> . The color map has a transparent pixel; its location is given by <i>transparent</i> .	

Instance Methods

getAlpha

```
public final int getAlpha (int pixel)
```

Parameters	<i>pixel</i>	A pixel encoded with this <code>ColorModel</code> .
Returns	The current alpha setting of the pixel.	
Overrides	<code>ColorModel.getAlpha(int)</code>	

getAlphas

```
public final void getAlphas (byte[] alphas)
```

Parameters *alphas* The alpha values of the pixels in the color model.

Description Copies the alpha values from the color map into the array `alphas[]`.

getBlue

```
public final int getBlue (int pixel)
```

Parameters *pixel* A pixel encoded with this `ColorModel`.

Returns The current blue setting of the pixel.

Overrides `ColorModel.getBlue(int)`

getBlues

```
public final void getBlues (byte[] blues)
```

Parameters *blues* The blue values of the pixels in the color model.

Description Copies the blue values from the color map into the array `blues[]`.

getGreen

```
public final int getGreen (int pixel)
```

Parameters *pixel* A pixel encoded with this `ColorModel`.

Returns The current green setting of the pixel.

Overrides `ColorModel.getGreen(int)`

getGreens

```
public final void getGreens (byte[] greens)
```

Parameters *greens* The green values of the pixels in the color model.

Description Copies the green values from the color map into the array `greens[]`.

getMapSize

```
public final int getMapSize()
```

Returns The current size of the color map table.

getRed

```
public final int getRed (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current red setting of the pixel.

Overrides ColorModel.getRed(int)

getReds

```
public final void getReds (byte[] reds)
```

Parameters *reds* The red values of the pixels in the color model.

Description Copies the red values from the color map into the array reds[].

getRGB

```
public final int getRGB (int pixel)
```

Parameters *pixel* A pixel encoded with this ColorModel.

Returns The current combined red, green, and blue settings of the pixel.

Overrides ColorModel.getRGB(int)

Description Gets the color of pixel in the default RGB color model.

getTransparentPixel

```
public final int getTransparentPixel()
```

Returns The array index for the transparent pixel in the color model.

See Also

ColorModel

22.11 *MemoryImageSource*

Description

The `MemoryImageSource` class allows you to create images completely in memory. You provide an array of data; it serves as an image producer for that data. In the 1.1 release, new methods support using this class for animation (notably `setAnimated()` and the various overrides of `newPixels()`).

Class Definition

```
public class java.awt.image.MemoryImageSource
    extends java.lang.Object
    implements java.awt.image.ImageProducer {

    // Constructors
    public MemoryImageSource (int w, int h, ColorModel cm,
        byte[] pix, int off, int scan);
    public MemoryImageSource (int w, int h, ColorModel cm,
        byte[] pix, int off, int scan, Hashtable props);
    public MemoryImageSource (int w, int h, ColorModel cm,
        int[] pix, int off, int scan);
    public MemoryImageSource (int w, int h, ColorModel cm,
        int[] pix, int off, int scan, Hashtable props);
    public MemoryImageSource (int w, int h, int[] pix,
        int off, int scan);
    public MemoryImageSource (int w, int h, int[] pix,
        int off, int scan, Hashtable props);

    // Instance Methods
    public synchronized void addConsumer (ImageConsumer ic);
    public synchronized boolean isConsumer (ImageConsumer ic);
    public void newPixels(); ★
    public synchronized void newPixels (int x, int y,
        int w, int h); ★
    public synchronized void newPixels (int x, int y,
        int w, int h, boolean framenotify); ★
    public synchronized void newPixels (byte[] newpix,
        ColorModel newmodel, int offset, int scansize); ★
    public synchronized void newPixels (int[] newpix,
        ColorModel newmodel, int offset, int scansize); ★
    public synchronized void removeConsumer (ImageConsumer ic);
    public void requestTopDownLeftRightResend (ImageConsumer ic);
    public synchronized void setAnimated (boolean animated); ★
    public synchronized void setFullBufferUpdates
        (boolean fullbuffers); ★
    public void startProduction (ImageConsumer ic);
}
```

Constructors

MemoryImageSource

```
public MemoryImageSource (int w, int h, ColorModel cm,
    byte[] pix, int off, int scan)
```

Parameters *w* Width of the image being created.

h Height of the image being created.
cm ColorModel of the image being created.
pix Array of pixel information.
off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.
Description Constructs a MemoryImageSource object with the given parameters to serve as an ImageProducer for a new image.

```
public MemoryImageSource (int w, int h, ColorModel cm,
byte[] pix, int off, int scan, Hashtable props)
```

Parameters *w* Width of the image being created.
h Height of the image being created.
cm ColorModel of the image being created.
pix Array of pixel information.
off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.
props Hashtable of properties associated with image.

Description Constructs a MemoryImageSource object with the given parameters to serve as an ImageProducer for a new image.

```
public MemoryImageSource (int w, int h, ColorModel cm,
int[] pix, int off, int scan)
```

Parameters *w* Width of the image being created.
h Height of the image being created.
cm ColorModel of the image being created.
pix Array of pixel information.
off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.

Description Constructs a MemoryImageSource object with the given parameters to serve as an ImageProducer for a new image.

```
public MemoryImageSource (int w, int h, ColorModel cm,
int[] pix, int off, int scan, Hashtable props)
```

Parameters *w* Width of the image being created.
h Height of the image being created.
cm ColorModel of the image being created.
pix Array of pixel information.

off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.
props Hashtable of properties associated with image.
Description Constructs a `MemoryImageSource` object with the given parameters to serve as an `ImageProducer` for a new image.

```
public MemoryImageSource (int w, int h, int[] pix, int  
off, int scan)
```

Parameters *w* Width of the image being created.
h Height of the image being created.
pix Array of pixel information.
off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.

Description Constructs a `MemoryImageSource` object with the given parameters to serve as an `ImageProducer` for a new image.

```
public MemoryImageSource (int w, int h, int[] pix, int  
off, int scan, Hashtable props)
```

Parameters *w* Width of the image being created.
h Height of the image being created.
pix Array of pixel information.
off The offset of the first pixel in the array; elements prior to this pixel are ignored.
scan The number of pixels per scan line in the array.

props Hashtable of properties associated with image.
Description Constructs a `MemoryImageSource` object with the given parameters to serve as an `ImageProducer` for a new image.

Class Methods

addConsumer

```
public synchronized void addConsumer (ImageConsumer ic)
```

Parameters *ic* `ImageConsumer` requesting image data.

Implements `ImageProducer.addConsumer (ImageConsumer)`

Description Registers an `ImageConsumer` as interested in `Image` information.

isConsumer

```
public synchronized boolean isConsumer (ImageConsumer ic)
```

- Parameters *ic* ImageConsumer to check.
- Returns true if ImageConsumer is registered with this ImageProducer, false otherwise.
- Implements ImageProducer.isConsumer (ImageConsumer)

newPixels

```
public synchronized void newPixels() ★
```

- Description Notifies the MemoryImageSource that there is new data available. The MemoryImageSource notifies all ImageConsumers that there is new data, sending the full rectangle and notifying the consumers that the frame is complete.

```
public synchronized void newPixels (int x, int y, int w,
int h, boolean framenotify) ★
```

- Parameters *x* x coordinate of the top left corner of the new image data.
- y* y coordinate of the top left corner of the new image data.
- w* Width of the new image data.
- h* Height of the new image data.

- Description Notifies the MemoryImageSource that there is new data available. The MemoryImageSource notifies all ImageConsumers that there is new data in the rectangle described by *x*, *y*, *w*, and *h*. The consumers are notified that the frame is complete.

```
public synchronized void newPixels (int x, int y, int w,
int h, boolean framenotify) ★
```

- Parameters *x* x coordinate of the top left corner of the new image data.
- y* y coordinate of the top left corner of the new image data.
- w* Width of the new image data.
- h* Height of the new image data.
- framenotify* Determines whether this is a complete frame or not.

- Description Notifies the MemoryImageSource that there is new data available. The MemoryImageSource notifies all ImageConsumers that there is new data in the rectangle described by *x*, *y*, *w*, and *h*. If *framenotify* is true, the consumers will also be notified that a frame is complete.

```
public synchronized void newPixels (byte[] newpix,  
ColorModel newmodel, int offset, int scansize) ★
```

Parameters *newpix* New array of image data.
 newmodel The color model to use for the new data.
 offset Offset into the data array
 scansize Size of each line.

Description Changes the image data for this MemoryImageSource and notifies its ImageConsumers that new data is available.

```
public synchronized void newPixels (int[] newpix,  
ColorModel newmodel, int offset, int scansize) ★
```

Parameters *newpix* New array of image data.
 newmodel The color model to use for the new data.
 offset Offset into the data array
 scansize Size of each line.

Description Changes the image data for this MemoryImageSource and notifies its ImageConsumers that new data is available.

removeConsumer

```
public void removeConsumer (ImageConsumer ic)
```

Parameters *ic* ImageConsumer to remove.

Implements ImageProducer.removeConsumer(ImageConsumer)

Description Removes an ImageConsumer from registered consumers for this ImageProducer.

requestTopDownLeftRightResend

```
public void requestTopDownLeftRightResend (ImageConsumer  
ic)
```

Parameters *ic* ImageConsumer requesting image data.

Implements ImageProducer.requestTopDownLeftRightResend(ImageConsumer)

Description Requests the retransmission of the Image data in top-down, left-to-right order.

setAnimated

```
public void setAnimated (boolean animated) ★
```

Parameters *animated* Flag indicating whether this image is animated.

Description To use this MemoryImageSource for animation, call setAnimated(true). The newPixels() methods will not work otherwise.

setFullBufferUpdates

```
public void setFullBufferUpdates (boolean fullbuffers) ★
```

Parameters *fullbuffers* true to send full buffers; false otherwise.

Description This method is only important for animations; i.e., you should call `setAnimated(true)` before using this function. If you do request to send full buffers, then any rectangle parameters passed to `newPixels()` will be ignored and the entire image will be sent to the consumers.

startProduction

```
public void startProduction (ImageConsumer ic)
```

Parameters *ic* ImageConsumer requesting image data.

Implements ImageProducer.startProduction(ImageConsumer)

Description Registers ImageConsumer as interested in Image information and tells ImageProducer to start sending the image data immediately.

See Also

ColorModel, Hashtable, ImageConsumer, ImageProducer, Object

22.12 PixelGrabber

Description

The PixelGrabber class is an ImageConsumer that captures the pixels from an image and saves them in an array.

Class Definition

```
public class java.awt.image.PixelGrabber
    extends java.lang.Object
    implements java.awt.image.ImageConsumer {

    // Constructors
    public PixelGrabber (Image img, int x, int y, int w, int h,
        boolean forceRGB); ★
    public PixelGrabber (Image image, int x, int y, int width,
        int height, int[] pixels, int offset, int scansize);
    public PixelGrabber (ImageProducer ip, int x, int y, int width,
        int height, int[] pixels, int offset, int scansize);

    // Instance Methods
    public synchronized void abortGrabbing(); ★
    public synchronized ColorModel getColorModel(); ★
```

```
public synchronized int getHeight(); ★
public synchronized Object getPixels(); ★
public synchronized int getStatus(); ★
public synchronized int getWidth(); ★
public boolean grabPixels() throws InterruptedException;
public synchronized boolean grabPixels (long ms)
    throws InterruptedException;
public synchronized void imageComplete (int status);
public void setColorModel (ColorModel model);
public void setDimensions (int width, int height);
public void setHints (int hints);
public void setPixels (int x, int y, int width, int height,
    ColorModel model, byte[] pixels, int offset, int scansize);
public void setPixels (int x, int y, int width, int height,
    ColorModel model, int[] pixels, int offset, int scansize);
public void setProperties (Hashtable properties);
public synchronized void startGrabbing(); ★
public synchronized int status(); ☆
}
```

Constructors

PixelGrabber

```
public PixelGrabber (Image img, int x, int y, int w, int
h, boolean forceRGB) ★
```

Parameters	<i>img</i>	Image to use as source of pixel data.
	<i>x</i>	x-coordinate of top-left corner of pixel data.
	<i>y</i>	y-coordinate of top-left corner of pixel data.
	<i>w</i>	Width of pixel data.
	<i>h</i>	Height of pixel data.
	<i>forceRGB</i>	true to force the use of the RGB color model; false otherwise.

Description Constructs a `PixelGrabber` instance to grab the specified area of the image.

```
public PixelGrabber (Image image, int x, int y, int width,
int height, int[] pixels, int offset, int scansize)
```

Parameters	<i>image</i>	Image to use as source of pixel data.
	<i>x</i>	x-coordinate of top-left corner of pixel data.
	<i>y</i>	y-coordinate of top-left corner of pixel data.
	<i>width</i>	Width of pixel data.
	<i>height</i>	Height of pixel data.
	<i>pixels</i>	Where to store pixel data when <code>grabPixels()</code> called.

offset Offset from beginning of each line in pixels array.

scansize Size of each line of data in pixels array.

Description Constructs a `PixelGrabber` instance to grab the specified area of the image and store the pixel data from this area in the array `pixels[]`.

```
public PixelGrabber (ImageProducer ip, int x, int y, int
width, int height, int[] pixels, int offset, int scansize)
```

Parameters *ip* ImageProducer to use as source of pixel data.

x x-coordinate of top-left corner of pixel data.

y y-coordinate of top-left corner of pixel data.

width Width of pixel data.

height Height of pixel data.

pixels Where to store pixel data when `grabPixels()` called.

offset Offset from beginning of each line in pixels array.

scansize Size of each line of data in pixels array.

Description Constructs a `PixelGrabber` instance to grab data from the specified area of the image generated by an `ImageProducer` and store the pixel data from this area in the array `pixels[]`.

Instance Methods

abortGrabbing

```
public synchronized void abortGrabbing() ★
```

Description Stops the `PixelGrabber`'s image-grabbing process.

getColorModel

```
public synchronized ColorModel getColorModel() ★
```

Returns The color model the `PixelGrabber` is using for its array.

getHeight

```
public synchronized int getHeight() ★
```

Returns The height of the grabbed image, or -1 if the height is not known.

getPixels

```
public synchronized Object getPixels() ★
```

Returns The array of pixels.

Description Either a byte array or an integer array is returned, or null if the size and format of the image are not yet known. Because the `PixelGrabber` may change its mind about what `ColorModel` it's using, different calls to this method may return different arrays until the image acquisition is complete.

getStatus

```
public synchronized int getStatus() ★
```

Returns A combination of `ImageObserver` flags indicating what data is available.

getWidth

```
public synchronized int getWidth() ★
```

Returns The width of the grabbed image, or -1 if the width is not known.

grabPixels

```
public boolean grabPixels() throws InterruptedException
```

Throws `InterruptedException`
 If image grabbing is interrupted before completion.

Returns `true` if the image has completed loading, `false` if the loading process aborted or an error occurred.

Description Starts the process of grabbing the pixel data from the source and storing it in the array `pixels[]` from constructor. Returns when the image is complete, loading aborts, or an error occurs.

```
public synchronized boolean grabPixels (long ms) throws  
InterruptedException
```

Parameters *ms* Milliseconds to wait for completion.

Returns `true` if image has completed loading, `false` if the loading process aborted, or an error or a timeout occurred.

Throws `InterruptedException`
 If image grabbing is interrupted before completion.

Description Starts the process of grabbing the pixel data from the source and storing it in the array `pixels[]` from constructor. Returns when the image is complete, loading aborts, an error occurs, or a timeout occurs.

imageComplete

```
public synchronized void imageComplete (int status)
```

Parameters *status* Image loading completion status flags.

Implements `ImageConsumer.imageComplete(int)`

Description Called by the `ImageProducer` to indicate that the image has been delivered.

setColorModel

```
void setColorModel (ColorModel model)
```

Parameters *model* The color model for the image.

Implements `ImageConsumer.setColorModel(ColorModel)`

Description Does nothing.

setDimensions

```
void setDimensions (int width, int height)
```

Parameters *width* Width for image.

height Height for image.

Implements `ImageConsumer.setDimensions(int, int)`

Description Does nothing.

setHints

```
void setHints (int hints)
```

Parameters *hints* Image consumption hints.

Implements `ImageConsumer.setHints(int)`

Description Does nothing.

setPixels

```
void setPixels (int x, int y, int width, int height,
ColorModel model, byte[] pixels, int offset, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.

y y-coordinate of top-left corner of pixel data delivered with this method call.

	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Implements	ImageConsumer.setPixels(int, int, int, int, ColorModel, byte[], int, int)	
Description	Called by the ImageProducer to deliver pixel data from the image.	

```
void setPixels (int x, int y, int width, int height,  
ColorModel model, int[] pixels, int offset, int scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Implements	ImageConsumer.setPixels(int, int, int, int, ColorModel, int[], int, int)	
Description	Called by the ImageProducer to deliver pixel data from the image.	

setPropertyies

```
void setPropertyies (Hashtable properties)
```

Parameters	<i>properties</i>	The properties for the image.
Implements	ImageConsumer.setPropertyies(Hashtable)	
Description	Does nothing.	

startGrabbing

```
public synchronized void startGrabbing() ★
```

Description Starts the PixelGrabber's image-grabbing process.

status

```
public synchronized int status () ☆
```

Returns The ImageObserver flags OR'ed together representing the available information about the image. Replaced by getStatus().

See Also

ColorModel, Hashtable, Image, ImageConsumer, ImageProducer, InterruptedException, MemoryImageSource, Object

22.13 ReplicateScaleFilter ★**Description**

The ReplicateScaleFilter class uses a simple-minded algorithm to scale an image. If the image is to be reduced, rows and columns of pixels are removed. If the image is to be expanded, rows and columns are duplicated (replicated).

Class Definition

```
public class ReplicateScaleFilter
    extends java.awt.image.ImageFilter {

    // Variables
    protected int destHeight;
    protected int destWidth;
    protected Object outpixbuf;
    protected int srcHeight;
    protected int srcWidth;
    protected int[] srccols;
    protected int[] srcrows;

    // Constructor
    public ReplicateScaleFilter(int width, int height);

    // Instance Methods
    public void setDimensions (int w, int h);
    public void setPixels(int x, int y, int w, int h, ColorModel model,
        byte[] pixels, int off, int scansize);
    public void setPixels(int x, int y, int w, int h, ColorModel model,
        int[] pixels, int off, int scansize);
```

```
    public void setProperties(Hashtable props);  
}
```

Variables

destHeight

protected int destHeight
Height of the scaled image.

destWidth

protected int destWidth
Width of the scaled image.

outpixmap

protected Object outpixmap
An internal buffer.

srcHeight

protected int srcHeight
Height of the original image.

srcWidth

protected int srcWidth
Width of the original image.

srccols

protected int[] srccols
Internal array used to map incoming columns to outgoing columns.

srcrows

protected int[] srcrows
Internal array used to map incoming rows to outgoing rows.

Constructor

ReplicateScaleFilter

```
public ReplicateScaleFilter (int width, int height)  
Parameters    width        Width of scaled image.
```

height Height of scaled image.

Description Constructs a `ReplicateScaleFilter` that scales the original image to the specified size. If both width and height are -1, the destination image size will be set to the source image size. If either one of the parameters is -1, it will be set to preserve the aspect ratio of the original image.

Instance Methods

setDimensions

```
public void setDimensions (int w, int h)
```

Parameters *w* Width of the source image.
h Height of the source image.

Overrides `ImageFilter.setDimensions(int, int)`

Description Sets the size of the source image.

setPixels

```
void setPixels (int x, int y, int w, int h, ColorModel  
model, byte[] pixels, int off, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.
y y-coordinate of top-left corner of pixel data delivered with this method call.
w Width of the rectangle of pixel data delivered with this method call.
h Height of the rectangle of pixel data delivered with this method call.
model Color model of image data.
pixels Image data.
off Offset from beginning of the pixels array.
scansize Size of each line of data in pixels array.

Overrides `ImageFilter.setPixels(int, int, int, int, ColorModel, byte[], int, int)`

Description Receives a rectangle of image data from the `ImageProducer`; scales these pixels and delivers them to any `ImageConsumers`.

```
void setPixels (int x, int y, int w, int h, ColorModel  
model, int[] pixels, int off, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.

	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>w</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>h</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>off</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Overrides	<code>ImageFilter.setPixels(int, int, int, int, ColorModel, int[], int, int)</code>	
Description		Receives a rectangle of image data from the <code>ImageProducer</code> ; scales these pixels and delivers them to any <code>ImageConsumers</code> .

setProperty

```
public void setProperties (Hashtable props)
```

Parameters	<i>props</i>	The properties for the image.
Overrides	<code>ImageFilter.setProperties(Hashtable)</code>	
Description		Adds the “rescale” image property to the properties list.

See Also

`ColorModel`, `Hashtable`, `ImageConsumer`, `ImageFilter`, `ImageProducer`

22.14 *RGBImageFilter*

Description

`RGBImageFilter` is an abstract class that helps you filter images based on each pixel’s color and position. In most cases, the only method you need to implement in subclasses is `filterRGB()`, which returns a new pixel value based on the old pixel’s color and position. `RGBImageFilter` cannot be used to implement filters that depend on the value of neighboring pixels, or other factors aside from color and position.

Class Definition

```
public abstract class java.awt.image.RGBImageFilter
    extends java.awt.image.ImageFilter {

    // Variables
    protected boolean canFilterIndexColorModel;
```

```

protected ColorModel newmodel;
protected ColorModel oldmodel;

// Instance Methods
public IndexColorModel filterIndexColorModel (IndexColorModel icm);
public abstract int filterRGB (int x, int y, int rgb);
public void filterRGBPixels (int x, int y, int width,
    int height, int[] pixels, int off, int scansize);
public void setColorModel (ColorModel model);
public void setPixels (int x, int y, int width, int height,
    ColorModel model, byte[] pixels, int offset, int scansize);
public void setPixels (int x, int y, int width, int height,
    ColorModel model, int[] pixels, int offset, int scansize);
public void substituteColorModel (ColorModel oldModel,
    ColorModel newModel);
}

```

Variables

canFilterIndexColorModel

```
protected boolean canFilterIndexColorModel
```

Setting the `canFilterIndexColorModel` variable to true indicates the filter can filter `IndexColorModel` images. To filter an `IndexColorModel`, the filter must depend only on color, not on position.

newmodel

```
protected ColorModel newmodel
```

A place to store a new `ColorModel`.

origmodel

```
protected ColorModel origmodel
```

A place to store an old `ColorModel`.

Instance Methods

filterIndexColorModel

```
public IndexColorModel filterIndexColorModel
(IndexColorModel icm)
```

Parameters *icm* Color model to filter.

Returns Filtered color model.

Description Helper method for `setColorModel()` that runs the entire color table of `icm` through the `filterRGB()` method of the subclass. Used only if `canFilterIndexColorModel` is true, and the image uses an `IndexColorModel`.

filterRGB

```
public abstract int filterRGB (int x, int y, int rgb)
```

Parameters *x* x-coordinate of pixel data.
 y y-coordinate of pixel data.
 rgb Color value of pixel to filter.

Returns New color value of pixel.

Description Subclasses implement this method to provide a filtering function that generates new pixels.

filterRGBPixels

```
public void filterRGBPixels (int x, int y, int width, int height, int[] pixels, int off, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data within entire image.
 y y-coordinate of top-left corner of pixel data within entire image.
 width Width of pixel data within entire image.
 height Height of pixel data within entire image.
 pixels Image data.
 off Offset from beginning of each line in pixels array.
 scansize Size of each line of data in pixels array.

Description Helper method for `setPixels()` that filters each element of the `pixels` buffer through the subclass's `filterRGB()` method.

setColorModel

```
public void setColorModel (ColorModel model)
```

Parameters *model* The color model for the image.

Overrides `ImageFilter.setColorModel(ColorModel)`

Description Sets the image's color model.

setPixels

```
public void setPixels (int x, int y, int width, int height, ColorModel model, byte[] pixels, int offset, int scansize)
```

Parameters *x* x-coordinate of top-left corner of pixel data delivered with this method call.

	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Overrides	<code>ImageFilter.setPixels(int, int, int, int, ColorModel, byte[], int, int)</code>	
Description	Called by the <code>ImageProducer</code> to deliver a rectangular block of pixels for filtering.	

```
public void setPixels (int x, int y, int width, int
height, ColorModel model, int[] pixels, int offset, int
scansize)
```

Parameters	<i>x</i>	x-coordinate of top-left corner of pixel data delivered with this method call.
	<i>y</i>	y-coordinate of top-left corner of pixel data delivered with this method call.
	<i>width</i>	Width of the rectangle of pixel data delivered with this method call.
	<i>height</i>	Height of the rectangle of pixel data delivered with this method call.
	<i>model</i>	Color model of image data.
	<i>pixels</i>	Image data.
	<i>offset</i>	Offset from beginning of the pixels array.
	<i>scansize</i>	Size of each line of data in pixels array.
Overrides	<code>ImageFilter.setPixels(int, int, int, int, ColorModel, int[], int, int)</code>	
Description	Called by the <code>ImageProducer</code> to deliver a rectangular block of pixels for filtering.	

substituteColorModel

```
public void substituteColorModel (ColorModel oldModel,
ColorModel newModel)
```

Parameters	<i>oldModel</i>	New value for <code>origmodel</code> variable.
------------	-----------------	--

newModel New value for `newmodel` variable.

Description Helper method for `setColorModel()` to initialize the protected variables `newmodel` and `origmodel`.

See Also

`ColorModel`, `ImageFilter`