# package java.awt.image

Interface Index

- <u>ImageConsumer</u>
- <u>ImageObserver</u>
- ImageProducer

Class Index

- <u>ColorModel</u>
- <u>CropImageFilter</u>
- <u>DirectColorModel</u>
- <u>FilteredImageSource</u>
- <u>ImageFilter</u>
- IndexColorModel
- <u>MemoryImageSource</u>
- <u>PixelGrabber</u>
- <u>RGBImageFilter</u>

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.ColorModel

java.lang.Object

+----java.awt.image.ColorModel

public class ColorModel extends Object

A class that encapsulates the methods for translating from pixel values to alpha, red, green, and blue color components for an image. This class is abstract.

See Also: IndexColorModel, DirectColorModel

Variable Index

pixel\_bits

Constructor Index

## ColorModel(int)

Constructs a ColorModel which describes a pixel of the specified number of bits.

Method Index

• getAlpha(int)

The subclass must provide a function which provides the alpha color compoment for the specified pixel.

• getBlue(int)

The subclass must provide a function which provides the blue color component for the specified pixel.

getGreen(int)

The subclass must provide a function which provides the green color compoment for the specified pixel.

#### getPixelSize()

Returns the number of bits per pixel described by this ColorModel.

#### getRGB(int)

Returns the color of the pixel in the default RGB color model.

#### getRGBdefault()

Return a ColorModel which describes the default format for integer RGB values used throughout the AWT image interfaces.

## getRed(int)

The subclass must provide a function which provides the red color compoment for the specified pixel.

Variables

# 🤍 pixel\_bits

protected int pixel\_bits

Constructors

# 🥪 ColorModel

public ColorModel(int bits)

Constructs a ColorModel which describes a pixel of the specified number of bits.

Methods

# 오 getRGBdefault

public static <u>ColorModel</u> getRGBdefault()

Return a ColorModel which describes the default format for integer RGB values used throughout the AWT image interfaces. The format for the RGB values is an integer with 8 bits each of alpha, red, green, and blue color components ordered correspondingly from the most significant byte to the least significant byte, as in: 0xAARRGGBB

## getPixelSize

public int getPixelSize()

Returns the number of bits per pixel described by this ColorModel.

## 🤍 getRed

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

The subclass must provide a function which provides the red color component for the specified pixel.

#### **Returns:**

The red color component ranging from 0 to 255

## 🤍 getGreen

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

The subclass must provide a function which provides the green color compoment for the specified pixel.

**Returns:** 

The green color component ranging from 0 to 255

## 🧶 getBlue

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

The subclass must provide a function which provides the blue color component for the specified pixel.

**Returns:** 

The blue color component ranging from 0 to 255

## 🤍 getAlpha

public abstract int getAlpha(int pixel)

The subclass must provide a function which provides the alpha color component for the specified pixel.

#### **Returns:**

The alpha transparency value ranging from 0 to 255

#### 🧶 getRGB

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

Returns the color of the pixel in the default RGB color model. See Also: getRGBdefault

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.CropImageFilter

java.lang.Object

---java.awt.image.ImageFilter +----java.awt.image.CropImageFilter

public class CropImageFilter extends **ImageFilter** 

An ImageFilter class for cropping images. This class extends the basic ImageFilter Class to extract a given rectangular region of an existing Image and provide a source for a new image containing just the extracted region. It is meant to be used in conjunction with a FilteredImageSource object to produce cropped versions of existing images.

#### See Also:

FilteredImageSource, ImageFilter

Constructor Index

## <u>CropImageFilter</u>(int, int, int)

Constructs a CropImageFilter that extracts the absolute rectangular region of pixels from its source Image as specified by the x, y, w, and h parameters.

Method Index

## setDimensions(int, int)

Override the source image's dimensions and pass the dimensions of the rectangular cropped region to the ImageConsumer.

• <u>setPixels</u>(int, int, int, int, ColorModel, byte[], int, int)

Determine whether the delivered byte pixels intersect the region to be extracted and passes through only that subset of pixels that appear in the output region.

• setPixels(int, int, int, int, ColorModel, int[], int, int)

Determine if the delivered int pixels intersect the region to be extracted and pass through only that subset of pixels that appear in the output region.

setProperties(Hashtable)

Passes along the properties from the source object after adding a property indicating the cropped region.



## 🥪 CropImageFilter

Constructs a CropImageFilter that extracts the absolute rectangular region of pixels from its source Image as specified by the x, y, w, and h parameters. **Parameters:** 

- x the x location of the top of the rectangle to be extracted
- y the y location of the top of the rectangle to be extracted
- w the width of the rectangle to be extracted
- h the height of the rectangle to be extracted

Methods

## setProperties

```
public void setProperties(<u>Hashtable</u> props)
```

Passes along the properties from the source object after adding a property indicating the cropped region.

#### **Overrides:**

setProperties in class ImageFilter

#### 🛢 setDimensions

Override the source image's dimensions and pass the dimensions of the rectangular cropped region to the ImageConsumer.

## **Overrides:**

setDimensions in class ImageFilter

```
See Also:
```

<u>ImageConsumer</u>

#### 🤍 setPixels

```
public void setPixels(int x,
```

```
int y,
int w,
int h,
<u>ColorModel</u> model,
byte pixels[],
int off,
int scansize)
```

Determine whether the delivered byte pixels intersect the region to be extracted and passes through only that subset of pixels that appear in the output region. **Overrides:** 

setPixels in class ImageFilter

## 🧶 setPixels

Determine if the delivered int pixels intersect the region to be extracted and pass through only that subset of pixels that appear in the output region. **Overrides:** 

setPixels in class ImageFilter

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.DirectColorModel

java.lang.Object

---java.awt.image.ColorModel +----java.awt.image.DirectColorModel

public class DirectColorModel extends ColorModel

A ColorModel class that specifies a translation from pixel values to alpha, red, green, and blue color components for pixels which have the color components embedded directly in the bits of the pixel itself. This color model is similar to an X11 TrueColor visual.

Many of the methods in this class are final. This is because the underlying native graphics code makes assumptions about the layout and operation of this class and those assumptions are reflected in the implementations of the methods here that are marked final. You can subclass this class for other reaons, but you cannot override or modify the behaviour of those methods.

See Also:

ColorModel

Constructor Index

## DirectColorModel(int, int, int, int)

Constructs a DirectColorModel from the given masks specifying which bits in the pixel contain the red, green and blue color components.

## DirectColorModel(int, int, int, int, int)

Constructs a DirectColorModel from the given masks specifying which bits in the pixel contain the alhpa, red, green and blue color components.

Method Index

## getAlpha(int)

Return the alpha transparency value for the specified pixel in the range 0-255.

#### getAlphaMask()

Returns the mask indicating which bits in a pixel contain the alpha transparency component.

getBlue(int)

Returns the blue color component for the specified pixel in the range 0-255.

## getBlueMask()

Returns the mask indicating which bits in a pixel contain the blue color component.

getGreen(int)

Returns the green color component for the specified pixel in the range 0-255.

## getGreenMask()

Returns the mask indicating which bits in a pixel contain the green color component.

getRGB(int)

Returns the color of the pixel in the default RGB color model.

getRed(int)

Returns the red color component for the specified pixel in the range 0-255.

## getRedMask()

Returns the mask indicating which bits in a pixel contain the red color component.

Constructors

## DirectColorModel

Constructs a DirectColorModel from the given masks specifying which bits in the pixel contain the red, green and blue color components. Pixels described by this color model will all have alpha components of 255 (fully opaque). All of the bits in each mask must be contiguous and fit in the specified number of least significant bits of the integer.

#### DirectColorModel

Constructs a DirectColorModel from the given masks specifying which bits in the pixel contain the alhpa, red, green and blue color components. All of the bits in each mask must be contiguous and fit in the specified number of least significant bits of the integer.

Methods

## < getRedMask

public final int getRedMask()

Returns the mask indicating which bits in a pixel contain the red color component.

#### 🧕 getGreenMask

public final int getGreenMask()

Returns the mask indicating which bits in a pixel contain the green color component.

#### 🧶 getBlueMask

public final int getBlueMask()

Returns the mask indicating which bits in a pixel contain the blue color component.

## 🧶 getAlphaMask

public final int getAlphaMask()

Returns the mask indicating which bits in a pixel contain the alpha transparency component.

#### 🧶 getRed

public final int getRed(int pixel)

Returns the red color component for the specified pixel in the range 0-255. **Overrides:** <u>getRed</u> in class <u>ColorModel</u>

#### 🧶 getGreen

public final int getGreen(int pixel)

Returns the green color component for the specified pixel in the range 0–255. **Overrides:** 

<u>getGreen</u> in class <u>ColorModel</u>

## 🧶 getBlue

public final int getBlue(int pixel)

Returns the blue color component for the specified pixel in the range 0-255. **Overrides:** 

<u>getBlue</u> in class <u>ColorModel</u>

## 🔍 getAlpha

public final int getAlpha(int pixel)

Return the alpha transparency value for the specified pixel in the range 0–255. **Overrides:** 

getAlpha in class ColorModel

# 🧶 getRGB

public final int getRGB(int pixel)

Returns the color of the pixel in the default RGB color model. **Overrides:** <u>getRGB</u> in class <u>ColorModel</u> **See Also:** <u>getRGBdefault</u>

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

<u>All Packages Class Hierarchy This Package Previous Next Index</u>

# Class java.awt.image.FilteredImageSource

java.lang.Object

+----java.awt.image.FilteredImageSource

public class FilteredImageSource extends Object implements ImageProducer

This class is an implementation of the ImageProducer interface which takes an existing image and a filter object and uses them to produce image data for a new filtered version of the original image. Here is an example which filters an image by swapping the red and blue compents:

```
Image src = getImage("doc:///demo/images/duke/T1.gif");
ImageFilter colorfilter = new RedBlueSwapFilter();
Image img = createImage(new FilteredImageSource(src.getSource(),
                                                colorfilter));
```

See Also: ImageProducer



• FilteredImageSource(ImageProducer, ImageFilter)

Constructs an ImageProducer object from an existing ImageProducer and a filter object.

Method Index

addConsumer(ImageConsumer)

Adds an ImageConsumer to the list of consumers interested in data for this image. • **isConsumer**(ImageConsumer)

Determines whether an ImageConsumer is on the list of consumers currently interested in data for this image.

removeConsumer(ImageConsumer)

Removes an ImageConsumer from the list of consumers interested in data for this

image.

#### <u>requestTopDownLeftRightResend</u>(ImageConsumer)

Requests that a given ImageConsumer have the image data delivered one more time in top-down, left-right order.

#### startProduction(ImageConsumer)

Adds an ImageConsumer to the list of consumers interested in data for this image, and immediately starts delivery of the image data through the ImageConsumer interface.



#### FilteredImageSource

Constructs an ImageProducer object from an existing ImageProducer and a filter object.

See Also:

ImageFilter, createImage



#### 🧶 addConsumer

public synchronized void addConsumer(<u>ImageConsumer</u> ic)

Adds an ImageConsumer to the list of consumers interested in data for this image. **See Also:** 

<u>ImageConsumer</u>

## 🤍 isConsumer

public synchronized boolean isConsumer(<u>ImageConsumer</u> ic)

Determines whether an ImageConsumer is on the list of consumers currently interested in data for this image.

#### **Returns:**

true if the ImageConsumer is on the list; false otherwise See Also: ImageConsumer

<u>ImageConsumer</u>

#### 🛢 removeConsumer

public synchronized void removeConsumer(<u>ImageConsumer</u> ic)

Removes an ImageConsumer from the list of consumers interested in data for this image.

See Also:

**ImageConsumer** 

## startProduction

```
public void startProduction(<u>ImageConsumer</u> ic)
```

Adds an ImageConsumer to the list of consumers interested in data for this image, and immediately starts delivery of the image data through the ImageConsumer interface.

See Also:

**ImageConsumer** 

## requestTopDownLeftRightResend

```
public void requestTopDownLeftRightResend(<u>ImageConsumer</u> ic)
```

Requests that a given ImageConsumer have the image data delivered one more time in top-down, left-right order. The request is handed to the ImageFilter for further processing, since the ability to preserve the pixel ordering depends on the filter.

See Also: ImageConsumer

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

# Interface java.awt.image.ImageConsumer

public interface **ImageConsumer** extends <u>Object</u>

The interface for objects expressing interest in image data through the ImageProducer interfaces. When a consumer is added to an image producer, the producer delivers all of the data about the image using the method calls defined in this interface.

See Also:

**ImageProducer** 

Variable Index

#### <u>COMPLETESCANLINES</u>

The pixels will be delivered in (multiples of) complete scanlines at a time.

#### IMAGEABORTED

The image creation process was deliberately aborted.

#### IMAGEERROR

An error was encountered while producing the image.

## RANDOMPIXELORDER

The pixels will be delivered in a random order.

## SINGLEFRAME

The image contain a single static image.

#### SINGLEFRAMEDONE

One frame of the image is complete but there are more frames to be delivered.

## • SINGLEPASS

The pixels will be delivered in a single pass.

## STATICIMAGEDONE

The image is complete and there are no more pixels or frames to be delivered.

## TOPDOWNLEFTRIGHT

The pixels will be delivered in top-down, left-to-right order.

Method Index

## • imageComplete(int)

The imageComplete method is called when the ImageProducer is finished delivering all of the pixels that the source image contains, or when a single frame of a multi-frame animation has been completed, or when an error in loading or producing the image has occured.

#### setColorModel(ColorModel)

The ColorModel object used for the majority of the pixels reported using the setPixels method calls.

• **setDimensions**(int, int)

The dimensions of the source image are reported using the setDimensions method call.

setHints(int)

The ImageProducer can deliver the pixels in any order, but the ImageConsumer may be able to scale or convert the pixels to the destination ColorModel more efficiently or with higher quality if it knows some information about how the pixels will be delivered up front.

#### • setPixels(int, int, int, int, ColorModel, byte[], int, int)

The pixels of the image are delivered using one or more calls to the setPixels method.

• <u>setPixels</u>(int, int, int, int, ColorModel, int[], int, int)

The pixels of the image are delivered using one or more calls to the setPixels method.

• setProperties(Hashtable)

Sets the extensible list of properties associated with this image.

Variables

## RANDOMPIXELORDER

public final static int RANDOMPIXELORDER

The pixels will be delivered in a random order. This tells the ImageConsumer not to use any optimizations that depend on the order of pixel delivery, which should be the default assumption in the absence of any call to the setHints method. **See Also:** 

<u>setHints</u>

#### **DOPDOWNLEFTRIGHT**

public final static int TOPDOWNLEFTRIGHT

The pixels will be delivered in top-down, left-to-right order. See Also: <u>setHints</u>

#### COMPLETESCANLINES

public final static int COMPLETESCANLINES

The pixels will be delivered in (multiples of) complete scanlines at a time. See Also: setHints

## SINGLEPASS

public final static int SINGLEPASS

The pixels will be delivered in a single pass. Each pixel will appear in only one call to any of the setPixels methods. An example of an image format which does not meet this criterion is a progressive JPEG image which defines pixels in multiple passes, each more refined than the previous. **See Also:** 

<u>setHints</u>

#### SINGLEFRAME

public final static int SINGLEFRAME

The image contain a single static image. The pixels will be defined in calls to the setPixels methods and then the imageComplete method will be called with the STATICIMAGEDONE flag after which no more image data will be delivered. An example of an image type which would not meet these criteria would be the output of a video feed, or the representation of a 3D rendering being manipulated by the user. The end of each frame in those types of images will be indicated by calling imageComplete with the SINGLEFRAMEDONE flag.

#### See Also:

setHints, imageComplete

#### **IMAGEERROR**

public final static int IMAGEERROR

An error was encountered while producing the image. **See Also:** 

<u>imageComplete</u>

#### SINGLEFRAMEDONE

public final static int SINGLEFRAMEDONE

One frame of the image is complete but there are more frames to be delivered. See Also:

<u>imageComplete</u>

## STATICIMAGEDONE

public final static int STATICIMAGEDONE

The image is complete and there are no more pixels or frames to be delivered. See Also:

<u>imageComplete</u>

#### IMAGEABORTED

public final static int IMAGEABORTED

The image creation process was deliberately aborted. See Also: <u>imageComplete</u>

Methods

## 🧶 setDimensions

The dimensions of the source image are reported using the setDimensions method call.

## 🤍 setProperties

public abstract void setProperties(<u>Hashtable</u> props)

Sets the extensible list of properties associated with this image.

#### 🧶 setColorModel

public abstract void setColorModel(<u>ColorModel</u> model)

The ColorModel object used for the majority of the pixels reported using the setPixels method calls. Note that each set of pixels delivered using setPixels contains its own ColorModel object, so no assumption should be made that this model will be the only one used in delivering pixel values. A notable case where multiple ColorModel objects may be seen is a filtered image when for each set of pixels that it filters, the filter determines whether the pixels can be sent on untouched, using the original ColorModel, or whether the pixels should be modified (filtered) and passed on using a ColorModel more convenient for the filtering process.

See Also:

 $\underline{ColorModel}$ 

## 🤍 setHints

public abstract void setHints(int hintflags)

The ImageProducer can deliver the pixels in any order, but the ImageConsumer may be able to scale or convert the pixels to the destination ColorModel more efficiently or with higher quality if it knows some information about how the pixels will be delivered up front. The setHints method should be called before any calls to any of the setPixels methods with a bit mask of hints about the manner in which the pixels will be delivered. If the ImageProducer does not follow the guidelines for the indicated hint, the results are undefined.

## setPixels

The pixels of the image are delivered using one or more calls to the setPixels method. Each call specifies the location and size of the rectangle of source pixels that are contained in the array of pixels. The specified ColorModel object should be used to convert the pixels into their corresponding color and alpha components. Pixel (m,n) is stored in the pixels array at index (n \* scansize + m + off). The pixels delivered using this method are all stored as bytes.

#### See Also:

**ColorModel** 

#### 🤍 setPixels

The pixels of the image are delivered using one or more calls to the setPixels method. Each call specifies the location and size of the rectangle of source pixels that are contained in the array of pixels. The specified ColorModel object should be used to convert the pixels into their corresponding color and alpha components. Pixel (m,n) is stored in the pixels array at index (n \* scansize + m + off). The pixels delivered using this method are all stored as ints. **See Also:** 

ColorModel

#### 🌻 imageComplete

public abstract void imageComplete(int status)

The imageComplete method is called when the ImageProducer is finished delivering all of the pixels that the source image contains, or when a single frame of a multi-frame animation has been completed, or when an error in loading or producing the image has occured. The ImageConsumer should remove itself from the list of consumers registered with the ImageProducer at this time, unless it is interested in successive frames.

# See Also:

 $\underline{removeConsumer}$ 

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.ImageFilter

java.lang.Object

+----java.awt.image.ImageFilter

public class ImageFilter extends Object implements ImageConsumer, Cloneable

This class implements a filter for the set of interface methods that are used to deliver data from an ImageProducer to an ImageConsumer. It is meant to be used in conjunction with a FilteredImageSource object to produce filtered versions of existing images. It is a base class that provides the calls needed to implement a "Null filter" which has no effect on the data being passed through. Filters should subclass this class and override the methods which deal with the data that needs to be filtered and modify it as necessary.

#### See Also:

FilteredImageSource, ImageConsumer

Variable Index

consumer

The consumer of the particular image data stream for which this instance of the ImageFilter is filtering data.

Constructor Index

ImageFilter()

Method Index

oclone() Clones this object.

#### • **getFilterInstance**(ImageConsumer)

Returns a unique instance of an ImageFilter object which will actually perform the filtering for the specified ImageConsumer.

• **imageComplete**(int)

Filters the information provided in the imageComplete method of the ImageConsumer interface.

## <u>resendTopDownLeftRight</u>(ImageProducer)

Responds to a request for a TopDownLeftRight (TDLR) ordered resend of the pixel data from an ImageConsumer.

#### setColorModel(ColorModel)

Filter the information provided in the setColorModel method of the ImageConsumer interface.

#### setDimensions(int, int)

Filters the information provided in the setDimensions method of the ImageConsumer interface.

## setHints(int)

Filters the information provided in the setHints method of the ImageConsumer interface.

<u>setPixels</u>(int, int, int, int, ColorModel, byte[], int, int)

Filters the information provided in the setPixels method of the ImageConsumer interface which takes an array of bytes.

#### • <u>setPixels</u>(int, int, int, int, ColorModel, int[], int, int)

Filters the information provided in the setPixels method of the ImageConsumer interface which takes an array of integers.

#### setProperties(Hashtable)

Passes the properties from the source object along after adding a property indicating the stream of filters it has been run through.

Variables

#### 🥯 consumer

protected <u>ImageConsumer</u> consumer

The consumer of the particular image data stream for which this instance of the ImageFilter is filtering data. It is not initialized during the constructor, but rather during the getFilterInstance() method call when the FilteredImageSource is creating a unique instance of this object for a particular image data stream. **See Also:** 

getFilterInstance, ImageConsumer

Constructors

ImageFilter

public ImageFilter()

Methods

## 🤍 getFilterInstance

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

Returns a unique instance of an ImageFilter object which will actually perform the filtering for the specified ImageConsumer. The default implementation just clones this object.

#### 🧶 setDimensions

Filters the information provided in the setDimensions method of the ImageConsumer interface. See Also: <u>setDimensions</u>

#### setProperties

```
public void setProperties(<u>Hashtable</u> props)
```

Passes the properties from the source object along after adding a property indicating the stream of filters it has been run through.

## setColorModel

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

Filter the information provided in the setColorModel method of the ImageConsumer interface. See Also: <u>setColorModel</u>

## 🜻 setHints

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

Filters the information provided in the setHints method of the ImageConsumer interface.
See Also:
setHints

## 🧶 setPixels

Filters the information provided in the setPixels method of the ImageConsumer interface which takes an array of bytes. **See Also:** 

setPixels

## 🛢 setPixels

Filters the information provided in the setPixels method of the ImageConsumer interface which takes an array of integers.

See Also:

<u>setPixels</u>

## 🔍 imageComplete

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

Filters the information provided in the imageComplete method of the ImageConsumer interface.
See Also:
<u>imageComplete</u>

## resendTopDownLeftRight

public void resendTopDownLeftRight(<u>ImageProducer</u> ip)

Responds to a request for a TopDownLeftRight (TDLR) ordered resend of the pixel data from an ImageConsumer. The ImageFilter can respond to this request in one of three ways.

1. If the filter can determine that it will forward the pixels in TDLR order if its upstream producer object sends them in TDLR order, then the request is automatically forwarded by default to the indicated ImageProducer using this filter as the requesting ImageConsumer, so no override is necessary.

- 2. If the filter can resend the pixels in the right order on its own (presumably because the generated pixels have been saved in some sort of buffer), then it can override this method and simply resend the pixels in TDLR order as specified in the ImageProducer API.
- 3. If the filter simply returns from this method then the request will be ignored and no resend will occur.

 $@see \ Image Producer {\it \# request Top Down Left Right Resend} \\$ 

#### **Parameters:**

ip – The ImageProducer that is feeding this instance of the filter – also the ImageProducer that the request should be forwarded to if necessary.

#### 🤍 clone

public Object clone()

Clones this object. **Overrides:** <u>clone</u> in class <u>Object</u>

All Packages Class Hierarchy This Package Previous Next Index

# Interface java.awt.image.ImageObserver

public interface **ImageObserver** extends <u>Object</u>

An asynchronous update interface for receiving notifications about Image information as the Image is constructed.

Variable Index

## • ABORT

An image which was being tracked asynchronously was aborted before production was complete.

## ALLBITS

A static image which was previously drawn is now complete and can be drawn again in its final form.

#### • ERROR

An image which was being tracked asynchronously has encountered an error.

## FRAMEBITS

Another complete frame of a multi-frame image which was previously drawn is now available to be drawn again.

## • HEIGHT

The height of the base image is now available and can be taken from the height argument to the imageUpdate callback method.

## PROPERTIES

The properties of the image are now available.

## • SOMEBITS

More pixels needed for drawing a scaled variation of the image are available.

# • WIDTH

The width of the base image is now available and can be taken from the width argument to the imageUpdate callback method.

Method Index

• **imageUpdate**(Image, int, int, int, int)

This method is called when information about an image which was previously requested using an asynchronous interface becomes available.

Variables

## 🔍 WIDTH

public final static int WIDTH

The width of the base image is now available and can be taken from the width argument to the imageUpdate callback method.
See Also:
getWidth, imageUpdate

#### 🔍 HEIGHT

public final static int HEIGHT

The height of the base image is now available and can be taken from the height argument to the imageUpdate callback method. **See Also:** 

getHeight, imageUpdate

#### PROPERTIES

public final static int PROPERTIES

The properties of the image are now available. See Also: <u>getProperty, imageUpdate</u>

#### SOMEBITS

public final static int SOMEBITS

More pixels needed for drawing a scaled variation of the image are available. The bounding box of the new pixels can be taken from the x, y, width, and height arguments to the imageUpdate callback method. See Also:

drawImage, imageUpdate

#### **9** FRAMEBITS

public final static int FRAMEBITS

Another complete frame of a multi-frame image which was previously drawn is now available to be drawn again. The x, y, width, and height arguments to the imageUpdate callback method should be ignored. See Also:

drawImage, imageUpdate

## 🭳 ALLBITS

public final static int ALLBITS

A static image which was previously drawn is now complete and can be drawn again in its final form. The x, y, width, and height arguments to the imageUpdate callback method should be ignored.

See Also:

drawImage, imageUpdate

## 오 ERROR

public final static int ERROR

An image which was being tracked asynchronously has encountered an error. No further information will become available and drawing the image will fail. As a convenience, the ABORT flag will be indicated at the same time to indicate that the image production was aborted.

See Also:

<u>imageUpdate</u>

## 🤍 ABORT

public final static int ABORT

An image which was being tracked asynchronously was aborted before production was complete. No more information will become available without further action to trigger another image production sequence. If the ERROR flag was not also set in this image update, then accessing any of the data in the image will restart the production again, probably from the beginning.

See Also:

<u>imageUpdate</u>

Methods

🤍 imageUpdate

This method is called when information about an image which was previously requested using an asynchronous interface becomes available. Asynchronous interfaces are method calls such as getWidth(ImageObserver) and drawImage(img, x, y, ImageObserver) which take an ImageObserver object as an argument. Those methods register the caller as interested either in information about the overall image itself (in the case of getWidth(ImageObserver)) or about an output version of an image (in the case of the drawImage(img, x, y, [w, h,] ImageObserver) call).

This method should return true if further updates are needed or false if the required information has been acquired. The image which was being tracked is passed in using the img argument. Various constants are combined to form the infoflags argument which indicates what information about the image is now available. The interpretation of the x, y, width, and height arguments depends on the contents of the infoflags argument.

#### See Also:

getWidth, getHeight, drawImage

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

# Interface java.awt.image.ImageProducer

public interface **ImageProducer** extends <u>Object</u>

The interface for objects which can produce the image data for Images. Each image contains an ImageProducer which is used to reconstruct the image whenever it is needed, for example, when a new size of the Image is scaled, or when the width or height of the Image is being requested.

See Also:

<u>ImageConsumer</u>

Method Index

#### addConsumer(ImageConsumer)

This method is used to register an ImageConsumer with the ImageProducer for access to the image data during a later reconstruction of the Image.

## isConsumer(ImageConsumer)

This method determines if a given ImageConsumer object is currently registered with this ImageProducer as one of its consumers.

#### <u>removeConsumer</u>(ImageConsumer)

This method removes the given ImageConsumer object from the list of consumers currently registered to receive image data.

#### <u>requestTopDownLeftRightResend</u>(ImageConsumer)

This method is used by an ImageConsumer to request that the ImageProducer attempt to resend the image data one more time in TOPDOWNLEFTRIGHT order so that higher quality conversion algorithms which depend on receiving pixels in order can be used to produce a better output version of the image.

## startProduction(ImageConsumer)

This method both registers the given ImageConsumer object as a consumer and starts an immediate reconstruction of the image data which will then be delivered to this consumer and any other consumer which may have already been registered with the producer.

Methods

#### 🎈 addConsumer

public abstract void addConsumer(<u>ImageConsumer</u> ic)

This method is used to register an ImageConsumer with the ImageProducer for access to the image data during a later reconstruction of the Image. The ImageProducer may, at its discretion, start delivering the image data to the consumer using the ImageConsumer interface immediately, or when the next available image reconstruction is triggered by a call to the startProduction method. See Also:

startProduction

🤍 isConsumer

public abstract boolean isConsumer(<u>ImageConsumer</u> ic)

This method determines if a given ImageConsumer object is currently registered with this ImageProducer as one of its consumers.

removeConsumer

public abstract void removeConsumer(<u>ImageConsumer</u> ic)

This method removes the given ImageConsumer object from the list of consumers currently registered to receive image data. It is not considered an error to remove a consumer that is not currently registered. The ImageProducer should stop sending data to this consumer as soon as is feasible.

## startProduction

public abstract void startProduction(ImageConsumer ic)

This method both registers the given ImageConsumer object as a consumer and starts an immediate reconstruction of the image data which will then be delivered to this consumer and any other consumer which may have already been registered with the producer. This method differs from the addConsumer method in that a reproduction of the image data should be triggered as soon as possible. **See Also:** 

addConsumer

#### requestTopDownLeftRightResend

public abstract void requestTopDownLeftRightResend(<u>ImageConsumer</u> ic)

This method is used by an ImageConsumer to request that the ImageProducer attempt to resend the image data one more time in TOPDOWNLEFTRIGHT order so that higher quality conversion algorithms which depend on receiving pixels in order can be used to produce a better output version of the image. The ImageProducer is free to ignore this call if it cannot resend the data in that order. If the data can be resent, then the ImageProducer should respond by executing the following minimum set of ImageConsumer method calls:

```
ic.setHints(TOPDOWNLEFTRIGHT | < otherhints >);
ic.setPixels(...); // As many times as needed
ic.imageComplete();
```

## See Also: setHints

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

<u>All Packages Class Hierarchy This Package Previous Next Index</u>

# Class java.awt.image.IndexColorModel

java.lang.Object

---java.awt.image.ColorModel +----java.awt.image.IndexColorModel

public class IndexColorModel extends ColorModel

A ColorModel class that specifies a translation from pixel values to alpha, red, green, and blue color components for pixels which represent indices into a fixed colormap. An optional transparent pixel value can be supplied which indicates a completely transparent pixel, regardless of any alpha value recorded for that pixel value. This color model is similar to an X11 PseudoColor visual.

Many of the methods in this class are final. The reason for this is that the underlying native graphics code makes assumptions about the layout and operation of this class and those assumptions are reflected in the implementations of the methods here that are marked final. You can subclass this class for other reaons, but you cannot override or modify the behaviour of those methods.

See Also: ColorModel



IndexColorModel(int, int, byte[], byte[], byte[])

Constructs an IndexColorModel from the given arrays of red, green, and blue components.

IndexColorModel(int, int, byte[], byte[], byte[], int)

Constructs an IndexColorModel from the given arrays of red, green, and blue components.

## IndexColorModel(int, int, byte[], byte[], byte[], byte[])

Constructs an IndexColorModel from the given arrays of red, green, blue and alpha components.

IndexColorModel(int, int, byte[], int, boolean)

Constructs an IndexColorModel from a single arrays of packed red, green, blue and

optional alpha components.

IndexColorModel(int, int, byte[], int, boolean, int)

Constructs an IndexColorModel from a single arrays of packed red, green, blue and optional alpha components.

Method Index

• getAlpha(int)

Returns the alpha transparency value for the specified pixel in the range 0-255.

getAlphas(byte[])

Copies the array of alpha transparency values into the given array.

getBlue(int)

Returns the blue color component for the specified pixel in the range 0–255.

getBlues(byte[])

Copies the array of blue color components into the given array.

getGreen(int)

Returns the green color component for the specified pixel in the range 0-255.

getGreens(byte[])

Copies the array of green color components into the given array.

getMapSize()

Returns the size of the color component arrays in this IndexColorModel.

getRGB(int)

Returns the color of the pixel in the default RGB color model.

• getRed(int)

Returns the red color component for the specified pixel in the range 0-255.

getReds(byte[])

Copies the array of red color components into the given array.

## • getTransparentPixel()

Returns the index of the transparent pixel in this IndexColorModel or -1 if there is no transparent pixel.

Constructors

# IndexColorModel

Constructs an IndexColorModel from the given arrays of red, green, and blue components. Pixels described by this color model will all have alpha components of 255 (fully opaque). All of the arrays specifying the color components must have at least the specified number of entries.

**Parameters:** 

bits – The number of bits each pixel occupies.

size - The size of the color component arrays.

r – The array of red color components.

g – The array of green color components.

b - The array of blue color components.

#### IndexColorModel

Constructs an IndexColorModel from the given arrays of red, green, and blue components. Pixels described by this color model will all have alpha components of 255 (fully opaque), except for the indicated transparent pixel. All of the arrays specifying the color components must have at least the specified number of entries. **Parameters:** 

#### bits - The number of bits each pixel occupies.

size – The size of the color component arrays.

r – The array of red color components.

g – The array of green color components.

b – The array of blue color components.

trans - The index of the transparent pixel.

#### IndexColorModel

Constructs an IndexColorModel from the given arrays of red, green, blue and alpha components. All of the arrays specifying the color components must have at least the specified number of entries.

#### **Parameters:**

bits – The number of bits each pixel occupies.

size - The size of the color component arrays.

r – The array of red color components.

g – The array of green color components.

b – The array of blue color components.

a - The array of alpha value components.

## IndexColorModel

byte cmap[], int start, boolean hasalpha)

Constructs an IndexColorModel from a single arrays of packed red, green, blue and optional alpha components. The array must have enough values in it to fill all of the needed component arrays of the specified size.

## **Parameters:**

bits – The number of bits each pixel occupies. size – The size of the color component arrays. cmap – The array of color components. start – The starting offset of the first color component. hasalpha – Indicates whether alpha values are contained in the cmap array.

## IndexColorModel

Constructs an IndexColorModel from a single arrays of packed red, green, blue and optional alpha components. The specified transparent index represents a pixel which will be considered entirely transparent regardless of any alpha value specified for it. The array must have enough values in it to fill all of the needed component arrays of the specified size.

#### **Parameters:**

bits – The number of bits each pixel occupies.

size - The size of the color component arrays.

cmap - The array of color components.

start – The starting offset of the first color component.

hasalpha – Indicates whether alpha values are contained in the cmap array. trans – The index of the fully transparent pixel.

Methods

## 🌻 getMapSize

public final int getMapSize()

Returns the size of the color component arrays in this IndexColorModel.

## 🔍 getTransparentPixel

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

Returns the index of the transparent pixel in this IndexColorModel or -1 if there is no transparent pixel.

#### 🔍 getReds

public final void getReds(byte r[])

Copies the array of red color components into the given array. Only the initial entries of the array as specified by getMapSize() are written.

## 🧶 getGreens

public final void getGreens(byte g[])

Copies the array of green color components into the given array. Only the initial entries of the array as specified by getMapSize() are written.

## 🧶 getBlues

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

Copies the array of blue color components into the given array. Only the initial entries of the array as specified by getMapSize() will be written.

#### 🧶 getAlphas

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

Copies the array of alpha transparency values into the given array. Only the initial entries of the array as specified by getMapSize() will be written.

## 🜻 getRed

public final int getRed(int pixel)

Returns the red color component for the specified pixel in the range 0–255. **Overrides:** 

getRed in class ColorModel

### 🧶 getGreen

public final int getGreen(int pixel)

Returns the green color component for the specified pixel in the range 0–255. **Overrides:** 

getGreen in class ColorModel

# 🧶 getBlue

public final int getBlue(int pixel)

Returns the blue color component for the specified pixel in the range 0-255. **Overrides:** 

<u>getBlue</u> in class <u>ColorModel</u>

## 🧶 getAlpha

public final int getAlpha(int pixel)

Returns the alpha transparency value for the specified pixel in the range 0–255. **Overrides:** 

getAlpha in class ColorModel

# 🧶 getRGB

public final int getRGB(int pixel)

Returns the color of the pixel in the default RGB color model. **Overrides:** <u>getRGB</u> in class <u>ColorModel</u> **See Also:** <u>getRGBdefault</u>

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

# Class java.awt.image.MemoryImageSource

java.lang.Object

+----java.awt.image.MemoryImageSource

public class MemoryImageSource extends Object implements ImageProducer

This class is an implementation of the ImageProducer interface which uses an array to produce pixel values for an Image. Here is an example which calculates a 100x100 image representing a fade from black to blue along the X axis and a fade from black to red along the Y axis:

```
int w = 100;
int h = 100;
int pix[] = new int[w * h];
int index = 0;
for (int y = 0; y < h; y++) {
    int red = (y * 255) / (h - 1);
    for (int x = 0; x < w; x++) {
        int blue = (x * 255) / (w - 1);
        pix[index++] = (255)
```



MemoryImageSource(int, int, ColorModel, byte[], int, int) Constructs an ImageProducer object which uses an array of bytes to produce data for an Image object.

MemoryImageSource(int, int, ColorModel, byte[], int, int, Hashtable) Constructs an ImageProducer object which uses an array of bytes to produce data for an Image object.

MemoryImageSource(int, int, ColorModel, int[], int, int) Constructs an ImageProducer object which uses an array of integers to produce data for an Image object.

MemoryImageSource(int, int, ColorModel, int[], int, int, Hashtable)

Constructs an ImageProducer object which uses an array of integers to produce data for an Image object.

-

MemoryImageSource(int, int, int[], int, int)

Constructs an ImageProducer object which uses an array of integers in the default RGB ColorModel to produce data for an Image object.

•

MemoryImageSource(int, int, int[], int, int, Hashtable)
Constructs an ImageProducer object which uses an array of integers
in the default RGB ColorModel to produce data for an Image object.



```
addConsumer(ImageConsumer)
Adds an ImageConsumer to the list of consumers interested in
data for this image.

isConsumer(ImageConsumer)
Determine if an ImageConsumer is on the list of consumers currently
interested in data for this image.

removeConsumer(ImageConsumer)
Remove an ImageConsumer from the list of consumers interested in
data for this image.

requestTopDownLeftRightResend(ImageConsumer)
Requests that a given ImageConsumer have the image data delivered
one more time in top-down, left-right order.
```

startProduction(ImageConsumer)

```
Adds an ImageConsumer to the list of consumers interested in data for this image, and immediately start delivery of the image data through the ImageConsumer interface.
```



MemoryImageSource

Constructs an ImageProducer object which uses an array of bytes to produce data for an Image object. **See Also:** 

<u>createImage</u>

## 🔎 MemoryImageSource

Constructs an ImageProducer object which uses an array of bytes to produce data for an Image object. See Also: <u>createImage</u>

## 🔎 MemoryImageSource

Constructs an ImageProducer object which uses an array of integers to produce data for an Image object. **See Also:** 

<u>createImage</u>

## MemoryImageSource

Constructs an ImageProducer object which uses an array of integers to produce data for an Image object.

See Also:

<u>createImage</u>

#### MemoryImageSource

Constructs an ImageProducer object which uses an array of integers in the default RGB ColorModel to produce data for an Image object. See Also: <u>createImage, getRGBdefault</u>

## MemoryImageSource

Constructs an ImageProducer object which uses an array of integers in the default RGB ColorModel to produce data for an Image object. **See Also:** 

createImage, getRGBdefault



## 🤍 addConsumer

public synchronized void addConsumer(<u>ImageConsumer</u> ic)

Adds an ImageConsumer to the list of consumers interested in data for this image. See Also:

<u>ImageConsumer</u>

## 🤍 isConsumer

public synchronized boolean isConsumer(ImageConsumer ic)

Determine if an ImageConsumer is on the list of consumers currently interested in

data for this image. **Returns:** true if the ImageConsumer is on the list; false otherwise See Also: ImageConsumer

removeConsumer

public synchronized void removeConsumer(<u>ImageConsumer</u> ic)

Remove an ImageConsumer from the list of consumers interested in data for this image.

See Also:

ImageConsumer

## startProduction

public void startProduction(<u>ImageConsumer</u> ic)

Adds an ImageConsumer to the list of consumers interested in data for this image, and immediately start delivery of the image data through the ImageConsumer interface.

## See Also:

ImageConsumer

#### requestTopDownLeftRightResend

public void requestTopDownLeftRightResend(ImageConsumer ic)

Requests that a given ImageConsumer have the image data delivered one more time in top-down, left-right order. See Also: ImageConsumer

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.PixelGrabber

java.lang.Object

+----java.awt.image.PixelGrabber

public class **PixelGrabber** extends Object implements ImageConsumer

The PixelGrabber class implements an ImageConsumer which can be attached to an Image or ImageProducer object to retrieve a subset of the pixels in that image. Here is an example:

```
public abstract void handlesinglepixel(int x, int y, int pixel);
public void handlepixels(Image img, int x, int y, int w, int h) {
        int[] pixels = new int[w * h];
        PixelGrabber pg = new PixelGrabber(img, x, y, w, h, pixels, 0, w);
        try {
            pg.grabPixels();
        } catch (InterruptedException e) {
            System.err.println("interrupted waiting for pixels!");
            return:
        }
        if ((pg.status() & ImageObserver.ABORT) != 0) {
            System.err.println("image fetch aborted or errored");
            return;
        }
        for (int j = 0; j < h; j++) {
            for (int i = 0; i < w; i++) {
                handlesinglepixel(x+i, y+j, pixels[j * w + i]);
            }
        }
}
```

Constructor Index

## <u>PixelGrabber</u>(Image, int, int, int, int[], int, int)

Create a PixelGrabber object to grab the (x, y, w, h) rectangular section of pixels from the specified image into the given array.

#### • **<u>PixelGrabber</u>**(ImageProducer, int, int, int, int[], int, int)

Create a PixelGrabber object to grab the (x, y, w, h) rectangular section of pixels from the image produced by the specified ImageProducer into the given array.

Method Index

## grabPixels()

Request the Image or ImageProducer to start delivering pixels and wait for all of the pixels in the rectangle of interest to be delivered.

grabPixels(long)

Request the Image or ImageProducer to start delivering pixels and wait for all of the pixels in the rectangle of interest to be delivered or until the specified timeout has elapsed.

#### imageComplete(int)

The imageComplete method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### setColorModel(ColorModel)

The setColorModel method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## setDimensions(int, int)

The setDimensions method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### • setHints(int)

The setHints method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### • setPixels(int, int, int, int, ColorModel, byte[], int, int)

The setPixels method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## <u>setPixels</u>(int, int, int, int, ColorModel, int[], int, int)

The setPixels method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## • **setProperties**(Hashtable)

The setProperties method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### status()

Return the status of the pixels.

Constructors

## 🥥 PixelGrabber

```
public PixelGrabber(<u>Image</u> img,
```

```
int x,
int y,
int w,
int h,
int pix[],
int off,
int scansize)
```

Create a PixelGrabber object to grab the (x, y, w, h) rectangular section of pixels

from the specified image into the given array. The pixels are stored into the array in the default RGB ColorModel. The RGB data for pixel (i, j) where (i, j) is inside the rectangle (x, y, w, h) is stored in the array at pix[(j - y) \* scansize + (i - x) + off].

## **Parameters:**

img – the image to retrieve pixels from

x - the x coordinate of the upper left corner of the rectangle of pixels to retrieve from the image, relative to the default (unscaled) size of the image y - the y coordinate of the upper left corner of the rectangle of pixels to retrieve from the image

w - the width of the rectangle of pixels to retrieve

h – the height of the rectangle of pixels to retrieve

pix – the array of integers which are to be used to hold the RGB pixels retrieved from the image

off - the offset into the array of where to store the first pixel

scansize – the distance from one row of pixels to the next in the array **See Also:** 

getRGBdefault

## 🥥 PixelGrabber

```
public PixelGrabber(<u>ImageProducer</u> ip,
```

```
int x,
int y,
int w,
int h,
int pix[],
int off,
int scansize)
```

Create a PixelGrabber object to grab the (x, y, w, h) rectangular section of pixels from the image produced by the specified ImageProducer into the given array. The pixels are stored into the array in the default RGB ColorModel. The RGB data for pixel (i, j) where (i, j) is inside the rectangle (x, y, w, h) is stored in the array at pix[(j - y) \* scansize + (i - x) + off].

## **Parameters:**

img – the image to retrieve pixels from

x - the x coordinate of the upper left corner of the rectangle of pixels to retrieve from the image, relative to the default (unscaled) size of the image y - the y coordinate of the upper left corner of the rectangle of pixels to retrieve from the image

w - the width of the rectangle of pixels to retrieve

h – the height of the rectangle of pixels to retrieve

pix – the array of integers which are to be used to hold the RGB pixels retrieved from the image

off – the offset into the array of where to store the first pixel

scansize – the distance from one row of pixels to the next in the array **See Also:** 

#### getRGBdefault

Methods

## grabPixels

public boolean grabPixels() throws <u>InterruptedException</u>

Request the Image or ImageProducer to start delivering pixels and wait for all of the pixels in the rectangle of interest to be delivered. **Returns:** 

true if the pixels were successfully grabbed, false on abort, error or timeout Throws:InterruptedException

Another thread has interrupted this thread.

## grabPixels

public synchronized boolean grabPixels(long ms) throws InterruptedException

Request the Image or ImageProducer to start delivering pixels and wait for all of the pixels in the rectangle of interest to be delivered or until the specified timeout has elapsed.

#### **Parameters:**

ms – the number of milliseconds to wait for the image pixels to arrive before timing out

#### **Returns:**

true if the pixels were successfully grabbed, false on abort, error or timeout **Throws:**InterruptedException

Another thread has interrupted this thread.

#### 🧶 status

public synchronized int status()

Return the status of the pixels. The ImageObserver flags representing the available pixel information are returned. **Returns:** 

the bitwise OR of all relevant ImageObserver flags

See Also:

ImageObserver

## setDimensions

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

The setDimensions method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## 🤍 setHints

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

The setHints method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### setProperties

```
public void setProperties(<u>Hashtable</u> props)
```

The setProperties method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## setColorModel

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

The setColorModel method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

#### 🤍 setPixels

The setPixels method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## 🤍 setPixels

The setPixels method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

## 🧶 imageComplete

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

The imageComplete method is part of the ImageConsumer API which this class must implement to retrieve the pixels.

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>

All Packages Class Hierarchy This Package Previous Next Index

# Class java.awt.image.RGBImageFilter

java.lang.Object

+

---java.awt.image.ImageFilter +----java.awt.image.RGBImageFilter

public class **RGBImageFilter** extends ImageFilter

This class provides an easy way to create an ImageFilter which modifies the pixels of an image in the default RGB ColorModel. It is meant to be used in conjunction with a FilteredImageSource object to produce filtered versions of existing images. It is an abstract class that provides the calls needed to channel all of the pixel data through a single method which converts pixels one at a time in the default RGB ColorModel regardless of the ColorModel being used by the ImageProducer. The only method which needs to be defined to create a useable image filter is the filterRGB method. Here is an example of a definition of a filter which swaps the red and blue components of an image:

```
class RedBlueSwapFilter extends RGBImageFilter {
    public RedBlueSwapFilter() {
        // The filter's operation does not depend on the
        // pixel's location, so IndexColorModels can be
        // filtered directly.
        canFilterIndexColorModel = true;
    }
    public int filterRGB(int x, int y, int rgb) {
        return ((rgb & 0xff00ff00)
                  ((rgb & 0xff0000) >> 16)
                  ((rgb & Oxff)
```

Variable Index

#### canFilterIndexColorModel

This boolean indicates whether or not it is acceptable to apply the color filtering of the filterRGB method to the color table entries of an IndexColorModel object in lieu of pixel by pixel filtering.

• <u>newmodel</u>

<u>origmodel</u>

Constructor Index

RGBImageFilter()

Method Index

filterIndexColorModel (IndexColorModel) Filters an IndexColorModel object by running each entry in its color tables through the filterRGB function that RGBImageFilter subclasses must provide. filterRGB(int, int, int) Subclasses must specify a method to convert a single input pixel in the default RGB ColorModel to a single output pixel. filterRGBPixels(int, int, int, int, int[], int, int) Filters a buffer of pixels in the default RGB ColorModel by passing them one by one through the filterRGB method. setColorModel (ColorModel) If the ColorModel is an IndexColorModel, and the subclass has set the canFilterIndexColorModel flag to true, we substitute a filtered version of the color model here and wherever that original ColorModel object appears in the setPixels methods. 

setPixels(int, int, int, int, ColorModel, byte[], int, int)
If the ColorModel object is the same one that has already

been converted, then simply passes the pixels through with the converted ColorModel.

setPixels(int, int, int, int, ColorModel, int[], int, int)
If the ColorModel object is the same one that has already
been converted, then simply passes the pixels through with the
converted ColorModel, otherwise converts the buffer of integer
pixels to the default RGB ColorModel and passes the converted
buffer to the filterRGBPixels method to be converted one by one.

substituteColorModel(ColorModel, ColorModel)
Registers two ColorModel objects for substitution.

Variables



protected <u>ColorModel</u> origmodel

## 🥯 newmodel

protected <u>ColorModel</u> newmodel

## canFilterIndexColorModel

protected boolean canFilterIndexColorModel

This boolean indicates whether or not it is acceptable to apply the color filtering of the filterRGB method to the color table entries of an IndexColorModel object in lieu of pixel by pixel filtering. Subclasses should set this variable to true in their constructor if their filterRGB method does not depend on the coordinate of the pixel being filtered.

## See Also:

substituteColorModel, filterRGB, IndexColorModel



## 🥥 RGBImageFilter

public RGBImageFilter()



## 🔍 setColorModel

public void setColorModel(ColorModel model)

If the ColorModel is an IndexColorModel, and the subclass has set the canFilterIndexColorModel flag to true, we substitute a filtered version of the color model here and wherever that original ColorModel object appears in the setPixels methods. Otherwise overrides the default ColorModel used by the ImageProducer and specifies the default RGB ColorModel instead.

#### **Overrides:**

<u>setColorModel</u> in class <u>ImageFilter</u> See Also: <u>ImageConsumer</u>, <u>getRGBdefault</u>

## substituteColorModel

Registers two ColorModel objects for substitution. If the oldcm is encountered during any of the setPixels methods, the newcm is substituted and the pixels passed through untouched (but with the new ColorModel object).

#### **Parameters:**

oldcm – the ColorModel object to be replaced on the fly newcm – the ColorModel object to replace oldcm on the fly

## filterIndexColorModel

public IndexColorModel filterIndexColorModel(IndexColorModel icm)

Filters an IndexColorModel object by running each entry in its color tables through the filterRGB function that RGBImageFilter subclasses must provide. Uses coordinates of -1 to indicate that a color table entry is being filtered rather than an actual pixel value.

#### **Parameters:**

icm - the IndexColorModel object to be filtered

#### **Returns:**

a new IndexColorModel representing the filtered colors

#### filterRGBPixels

```
int scansize)
```

Filters a buffer of pixels in the default RGB ColorModel by passing them one by one through the filterRGB method. **See Also:** 

getRGBdefault, filterRGB

## setPixels

If the ColorModel object is the same one that has already been converted, then simply passes the pixels through with the converted ColorModel. Otherwise converts the buffer of byte pixels to the default RGB ColorModel and passes the converted buffer to the filterRGBPixels method to be converted one by one. **Overrides:** 

<u>setPixels</u> in class <u>ImageFilter</u>

See Also:

getRGBdefault, filterRGBPixels

## 🛢 setPixels

If the ColorModel object is the same one that has already been converted, then simply passes the pixels through with the converted ColorModel, otherwise converts the buffer of integer pixels to the default RGB ColorModel and passes the converted buffer to the filterRGBPixels method to be converted one by one. Converts a buffer of integer pixels to the default RGB ColorModel and passes the converted buffer to the filterRGBPixels method.

## **Overrides:**

<u>setPixels</u> in class <u>ImageFilter</u>

```
See Also:
```

getRGBdefault, filterRGBPixels

## 🧶 filterRGB

```
public abstract int filterRGB(int x,
```

int y, int rgb)

Subclasses must specify a method to convert a single input pixel in the default RGB ColorModel to a single output pixel. See Also: <u>getRGBdefault, filterRGBPixels</u>

<u>All Packages</u> <u>Class Hierarchy</u> <u>This Package</u> <u>Previous</u> <u>Next</u> <u>Index</u>