Hierarchy For All Packages

Package Hierarchies:

com.sun.media.jai.codec, javax.media.jai, javax.media.jai.iterator, javax.media.jai.operator, javax.media.jai.widget

Class Hierarchy

- class java.lang.Object
  - class javax.media.jai.operator.**AffinePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class com.sun.media.jai.codec.**BMPEncodeParam** (implements com.sun.media.jai.codec.ImageEncodeParam)
  - class javax.media.jai.**BorderExtender**
    - class javax.media.jai.**BorderExtenderConstant**
    - class javax.media.jai.**BorderExtenderCopy**
    - class javax.media.jai.**BorderExtenderReflect**
    - class javax.media.jai.**BorderExtenderWrap**
    - class javax.media.jai.**BorderExtenderZero**
  - class javax.media.jai.**CollectionImage** (implements java.util.Collection, javax.media.jai.ImageJAI)
    - class javax.media.jai.**CollectionOp**
    - class javax.media.jai.**ImageSequence**
    - class javax.media.jai.**ImageStack**
  - class java.awt.image.ColorModel (implements java.awt.Transparency)
    - class java.awt.image.ComponentColorModel
      - class javax.media.jai.**FloatDoubleColorModel**
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas
      - class javax.media.jai.**CanvasJAI**
      - class javax.media.jai.widget.**ImageCanvas**
    - class java.awt.Container
      - class java.awt.ScrollPane
        - class javax.media.jai.widget.**ScrollingImagePanel** (implements java.awt.event.AdjustmentListener, java.awt.event.ComponentListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener)
  - class javax.media.jai.operator.**ConjugatePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class javax.media.jai.**CoordinateImage**
  - class javax.media.jai.**CopyPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class java.awt.image.DataBuffer
    - class javax.media.jai.**DataBufferDouble**
    - class javax.media.jai.**DataBufferFloat**
  - class javax.media.jai.operator.**DFTPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class javax.media.jai.operator.**DivideComplexPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class com.sun.media.jai.codec.**FPXDecodeParam** (implements com.sun.media.jai.codec.ImageDecodeParam)
  - class java.awt.Graphics
    - class java.awt.Graphics2D
      - class javax.media.jai.**GraphicsJAI**
      - class javax.media.jai.**RenderableGraphics** (implements java.awt.image.renderable.RenderableImage)
      - class javax.media.jai.**TiledImageGraphics**
  - class javax.media.jai.**Histogram** (implements java.io.Serializable)
  - class javax.media.jai.operator.**IDFTPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class com.sun.media.jai.codec.**ImageCodec**
  - class com.sun.media.jai.codec.**ImageDecoderImpl** (implements com.sun.media.jai.codec.ImageDecoder)
  - class com.sun.media.jai.codec.**ImageEncoderImpl** (implements com.sun.media.jai.codec.ImageEncoder)
  - class javax.media.jai.operator.**ImageFunctionPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
  - class javax.media.jai.**ImageLayout** (implements java.lang.Cloneable, java.io.Serializable)
  - class javax.media.jai.**ImageMIPMap** (implements javax.media.jai.ImageJAI)
    - class javax.media.jai.**ImagePyramid**
  - class java.io.InputStream
    - class com.sun.media.jai.codec.**SeekableStream** (implements java.io.DataInput)
      - class com.sun.media.jai.codec.**ByteArraySeekableStream**
      - class com.sun.media.jai.codec.**FileCacheSeekableStream**
      - class com.sun.media.jai.codec.**FileSeekableStream**
      - class com.sun.media.jai.codec.**ForwardSeekableStream**

        - class com.sun.media.jai.codec.**MemoryCacheSeekableStream**
        - class com.sun.media.jai.codec.**SegmentedSeekableStream**
- class javax.media.jai.**IntegerSequence**
- class javax.media.jai.**Interpolation** (implements java.io.Serializable)
    - class javax.media.jai.**InterpolationBilinear**
    - class javax.media.jai.**InterpolationNearest**
    - class javax.media.jai.**InterpolationTable**
        - class javax.media.jai.**InterpolationBicubic**
        - class javax.media.jai.**InterpolationBicubic2**
- class javax.media.jai.**JAI**
- class javax.media.jai.**JaiI18N**
- class javax.media.jai.iterator.**JaiI18N**
- class javax.media.jai.operator.**JaiI18N**
- class javax.media.jai.widget.**JaiI18N**
- class com.sun.media.jai.codec.**JaiI18N**
- class com.sun.media.jai.codec.**JPEGEncodeParam** (implements com.sun.media.jai.codec.ImageEncodeParam)
- class javax.media.jai.**KernelJAI** (implements java.io.Serializable)
- class javax.media.jai.**LookupTableJAI** (implements java.io.Serializable)
    - class javax.media.jai.**ColorCube**
- class javax.media.jai.operator.**MagnitudePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.operator.**MagnitudeSquaredPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.operator.**MultiplyComplexPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.**MultiResolutionRenderableImage** (implements java.awt.image.renderable.RenderableImage, java.io.Serializable)
- class javax.media.jai.**NoParameterDefault**
- class javax.media.jai.**OperationDescriptorImpl** (implements javax.media.jai.OperationDescriptor)
    - class javax.media.jai.operator.**AbsoluteDescriptor**
    - class javax.media.jai.operator.**AddCollectionDescriptor**
    - class javax.media.jai.operator.**AddConstDescriptor**
    - class javax.media.jai.operator.**AddConstToCollectionDescriptor**
    - class javax.media.jai.operator.**AddDescriptor**
    - class javax.media.jai.operator.**AffineDescriptor**
    - class javax.media.jai.operator.**AndConstDescriptor**
    - class javax.media.jai.operator.**AndDescriptor**
    - class javax.media.jai.operator.**AWTImageDescriptor**
    - class javax.media.jai.operator.**BandCombineDescriptor**
    - class javax.media.jai.operator.**BandSelectDescriptor**
    - class javax.media.jai.operator.**BMPDescriptor**
    - class javax.media.jai.operator.**BorderDescriptor**
    - class javax.media.jai.operator.**BoxFilterDescriptor**
    - class javax.media.jai.operator.**ClampDescriptor**
    - class javax.media.jai.operator.**ColorConvertDescriptor**
    - class javax.media.jai.operator.**CompositeDescriptor**
    - class javax.media.jai.operator.**ConjugateDescriptor**
    - class javax.media.jai.operator.**ConstantDescriptor**
    - class javax.media.jai.operator.**ConvolveDescriptor**
    - class javax.media.jai.operator.**CropDescriptor**
    - class javax.media.jai.operator.**DCTDescriptor**
    - class javax.media.jai.operator.**DFTDescriptor**
    - class javax.media.jai.operator.**DivideByConstDescriptor**
    - class javax.media.jai.operator.**DivideComplexDescriptor**
    - class javax.media.jai.operator.**DivideDescriptor**
    - class javax.media.jai.operator.**DivideIntoConstDescriptor**
    - class javax.media.jai.operator.**EncodeDescriptor**
    - class javax.media.jai.operator.**ErrorDiffusionDescriptor**
    - class javax.media.jai.operator.**ExpDescriptor**
    - class javax.media.jai.operator.**ExtremaDescriptor**
    - class javax.media.jai.operator.**FileLoadDescriptor**
    - class javax.media.jai.operator.**FileStoreDescriptor**
    - class javax.media.jai.operator.**FormatDescriptor**
    - class javax.media.jai.operator.**FPXDescriptor**

    - class javax.media.jai.operator.**GIFDescriptor**
    - class javax.media.jai.operator.**GradientMagnitudeDescriptor**
    - class javax.media.jai.operator.**HistogramDescriptor**
    - class javax.media.jai.operator.**IDCTDescriptor**
    - class javax.media.jai.operator.**IDFTDescriptor**
    - class javax.media.jai.operator.**IIPDescriptor**
    - class javax.media.jai.operator.**IIPResolutionDescriptor**
    - class javax.media.jai.operator.**ImageFunctionDescriptor**
    - class javax.media.jai.operator.**InvertDescriptor**
    - class javax.media.jai.operator.**JPEGDescriptor**
    - class javax.media.jai.operator.**LogDescriptor**
    - class javax.media.jai.operator.**LookupDescriptor**
    - class javax.media.jai.operator.**MagnitudeDescriptor**
    - class javax.media.jai.operator.**MagnitudeSquaredDescriptor**
    - class javax.media.jai.operator.**MatchCDFDescriptor**
    - class javax.media.jai.operator.**MaxDescriptor**
    - class javax.media.jai.operator.**MeanDescriptor**
    - class javax.media.jai.operator.**MedianFilterDescriptor**
    - class javax.media.jai.operator.**MinDescriptor**
    - class javax.media.jai.operator.**MultiplyComplexDescriptor**
    - class javax.media.jai.operator.**MultiplyConstDescriptor**
    - class javax.media.jai.operator.**MultiplyDescriptor**
    - class javax.media.jai.operator.**NotDescriptor**
    - class javax.media.jai.operator.**OrConstDescriptor**
    - class javax.media.jai.operator.**OrderedDitherDescriptor**
    - class javax.media.jai.operator.**OrDescriptor**
    - class javax.media.jai.operator.**OverlayDescriptor**
    - class javax.media.jai.operator.**PatternDescriptor**
    - class javax.media.jai.operator.**PeriodicShiftDescriptor**
    - class javax.media.jai.operator.**PhaseDescriptor**
    - class javax.media.jai.operator.**PiecewiseDescriptor**
    - class javax.media.jai.operator.**PNGDescriptor**
    - class javax.media.jai.operator.**PNMDescriptor**
    - class javax.media.jai.operator.**PolarToComplexDescriptor**
    - class javax.media.jai.operator.**RenderableDescriptor**
    - class javax.media.jai.operator.**RescaleDescriptor**
    - class javax.media.jai.operator.**RotateDescriptor**
    - class javax.media.jai.operator.**ScaleDescriptor**
    - class javax.media.jai.operator.**ShearDescriptor**
    - class javax.media.jai.operator.**StreamDescriptor**
    - class javax.media.jai.operator.**SubtractConstDescriptor**
    - class javax.media.jai.operator.**SubtractDescriptor**
    - class javax.media.jai.operator.**SubtractFromConstDescriptor**
    - class javax.media.jai.operator.**ThresholdDescriptor**
    - class javax.media.jai.operator.**TIFFDescriptor**
    - class javax.media.jai.operator.**TranslateDescriptor**
    - class javax.media.jai.operator.**TransposeDescriptor**
    - class javax.media.jai.operator.**URLDescriptor**
    - class javax.media.jai.operator.**WarpDescriptor**
    - class javax.media.jai.operator.**XorConstDescriptor**
    - class javax.media.jai.operator.**XorDescriptor**
- class javax.media.jai.**OperationGraph**
- class javax.media.jai.**OperationRegistry** (implements java.io.Externalizable)
- class java.awt.image.renderable.ParameterBlock (implements java.lang.Cloneable, java.io.Serializable)
    - class javax.media.jai.**ParameterBlockJAI**
- class javax.media.jai.**PartialOrderNode** (implements java.lang.Cloneable, java.io.Serializable)
- class javax.media.jai.**PerspectiveTransform** (implements java.lang.Cloneable, java.io.Serializable)
- class javax.media.jai.operator.**PhasePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.**PlanarImage** (implements javax.media.jai.ImageJAI, java.awt.image.RenderedImage)
    - class javax.media.jai.**OpImage**
        - class javax.media.jai.**AreaOpImage**
        - class javax.media.jai.**PointOpImage**

      - class javax.media.jai.**NullOpImage**
    - class javax.media.jai.**SourcelessOpImage**
    - class javax.media.jai.**StatisticsOpImage**
    - class javax.media.jai.**UntiledOpImage**
    - class javax.media.jai.**WarpOpImage**
      - class javax.media.jai.**ScaleOpImage**
  - class javax.media.jai.**RemoteImage**
  - class javax.media.jai.**RenderedImageAdapter**
    - class javax.media.jai.**WritableRenderedImageAdapter** (implements java.awt.image.WritableRenderedImage)
  - class javax.media.jai.**RenderedOp** (implements java.io.Serializable)
  - class javax.media.jai.**Snapshot**
  - class javax.media.jai.**SnapshotImage** (implements java.awt.image.TileObserver)
  - class javax.media.jai.**SnapshotProxy**
  - class javax.media.jai.**TiledImage** (implements java.awt.image.WritableRenderedImage)
- class com.sun.media.jai.codec.**PNGDecodeParam** (implements com.sun.media.jai.codec.ImageDecodeParam)
- class com.sun.media.jai.codec.**PNGEncodeParam** (implements com.sun.media.jai.codec.ImageEncodeParam)
  - class com.sun.media.jai.codec.**PNGEncodeParam.Gray**
  - class com.sun.media.jai.codec.**PNGEncodeParam.Palette**
  - class com.sun.media.jai.codec.**PNGEncodeParam.RGB**
- class com.sun.media.jai.codec.**PNGSuggestedPaletteEntry** (implements java.io.Serializable)
- class com.sun.media.jai.codec.**PNMEncodeParam** (implements com.sun.media.jai.codec.ImageEncodeParam)
- class javax.media.jai.operator.**PolarToComplexPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.**ProductOperationGraph** (implements java.io.Serializable)
- class javax.media.jai.**PropertyGeneratorFromSource** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.**PropertySourceImpl** (implements javax.media.jai.PropertySource)
- class javax.media.jai.iterator.**RandomIterFactory**
- class java.awt.image.Raster
  - class java.awt.image.WritableRaster
    - class javax.media.jai.**WritableRasterJAI**
- class javax.media.jai.**RasterAccessor**
- class javax.media.jai.**RasterFactory**
- class javax.media.jai.**RasterFormatTag**
- class javax.media.jai.iterator.**RectIterFactory**
- class javax.media.jai.**RegistryInitData**
- class javax.media.jai.**RenderableImageAdapter** (implements javax.media.jai.PropertySource, java.awt.image.renderable.RenderableImage)
- class javax.media.jai.**RenderableOp** (implements javax.media.jai.PropertySource, java.awt.image.renderable.RenderableImage, java.io.Serializable)
- class java.awt.RenderingHints.Key
  - class javax.media.jai.**JAI.RenderingKey**
- class javax.media.jai.**ROI** (implements java.io.Serializable)
  - class javax.media.jai.**ROIShape**
- class javax.media.jai.**ROIShape.PolyShape**
- class javax.media.jai.**ROIShape.PolyShape.PolyEdge** (implements java.util.Comparator)
- class javax.media.jai.iterator.**RookIterFactory**
- class javax.media.jai.operator.**RotatePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class java.awt.image.SampleModel
  - class java.awt.image.ComponentSampleModel
    - class javax.media.jai.**ComponentSampleModelJAI**
- class javax.media.jai.operator.**ScalePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class com.sun.media.jai.codec.**SectorStreamSegmentMapper** (implements com.sun.media.jai.codec.StreamSegmentMapper)
- class javax.media.jai.**SequentialImage**
- class javax.media.jai.operator.**ShearPropertyGenerator** (implements javax.media.jai.PropertyGenerator)
- class javax.media.jai.**Storage**
- class javax.media.jai.**Store**
- class com.sun.media.jai.codec.**StreamSegment**
- class com.sun.media.jai.codec.**StreamSegmentMapperImpl** (implements com.sun.media.jai.codec.StreamSegmentMapper)
- class com.sun.media.jai.codec.**TIFFDecodeParam** (implements com.sun.media.jai.codec.ImageDecodeParam)
- class com.sun.media.jai.codec.**TIFFDirectory**

    - class com.sun.media.jai.codec.**TIFFEncodeParam** (implements com.sun.media.jai.codec.ImageEncodeParam)
    - class com.sun.media.jai.codec.**TIFFField**
    - class javax.media.jai.**TileCopy**
    - class javax.media.jai.operator.**TranslatePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
    - class javax.media.jai.operator.**TransposePropertyGenerator** (implements javax.media.jai.PropertyGenerator)
    - class javax.media.jai.**Warp** (implements java.io.Serializable)
        - class javax.media.jai.**WarpGrid**
        - class javax.media.jai.**WarpPerspective**
        - class javax.media.jai.**WarpPolynomial**
            - class javax.media.jai.**WarpAffine**
            - class javax.media.jai.**WarpCubic**
            - class javax.media.jai.**WarpGeneralPolynomial**
            - class javax.media.jai.**WarpQuadratic**
    - class javax.media.jai.operator.**WarpPropertyGenerator** (implements javax.media.jai.PropertyGenerator)

Interface Hierarchy

- interface java.lang.Cloneable
    - interface com.sun.media.jai.codec.**ImageDecodeParam**(also extends java.io.Serializable)
        - interface com.sun.media.jai.codec.**ImageEncodeParam**(also extends java.lang.Cloneable, java.io.Serializable)
    - interface com.sun.media.jai.codec.**ImageEncodeParam**(also extends com.sun.media.jai.codec.ImageDecodeParam, java.io.Serializable)
- interface javax.media.jai.**CollectionImageFactory**
- interface com.sun.media.jai.codec.**ImageDecoder**
- interface com.sun.media.jai.codec.**ImageEncoder**
- interface javax.media.jai.**ImageFunction**
- interface javax.media.jai.**OperationDescriptor**
- interface javax.media.jai.**PropertySource**
    - interface javax.media.jai.**ImageJAI**
- interface javax.media.jai.iterator.**RandomIter**
    - interface javax.media.jai.iterator.**WritableRandomIter**
- interface javax.media.jai.iterator.**RectIter**
    - interface javax.media.jai.iterator.**RookIter**
        - interface javax.media.jai.iterator.**WritableRookIter**(also extends javax.media.jai.iterator.WritableRectIter)
    - interface javax.media.jai.iterator.**WritableRectIter**
        - interface javax.media.jai.iterator.**WritableRookIter**(also extends javax.media.jai.iterator.RookIter)
- interface java.io.Serializable
    - interface com.sun.media.jai.codec.**ImageDecodeParam**(also extends java.lang.Cloneable)
        - interface com.sun.media.jai.codec.**ImageEncodeParam**(also extends java.lang.Cloneable, java.io.Serializable)
    - interface com.sun.media.jai.codec.**ImageEncodeParam**(also extends java.lang.Cloneable, com.sun.media.jai.codec.ImageDecodeParam)
    - interface javax.media.jai.**PropertyGenerator**
- interface com.sun.media.jai.codec.**StreamSegmentMapper**
- interface javax.media.jai.**TileCache**
- interface javax.media.jai.**TileScheduler**
- interface javax.media.jai.widget.**ViewportListener**

Package javax.media.jai

Interface Summary	
CollectionImageFactory	The `CollectionImageFactory` interface (often abbreviated CIF) is intended to be implemented by classes that wish to act as factories to produce different collection image operators.
ImageFunction	ImageFunction is a common interface for vector-valued functions which are to be evaluated at positions in the X-Y coordinate system.
ImageJAI	An interface implemented by all JAI image classes.
OperationDescriptor	This interface provides a comprehensive description of a specific image operation.
PropertyGenerator	An interface through which properties may be computed dynamically with respect to an environment of pre-existing properties.
PropertySource	An interface encapsulating the set of operations involved in identifying and reading properties.
TileCache	A class implementing a caching mechanism for image tiles.
TileScheduler	A class implementing a mechanism for scheduling tile calculation.

Class Summary	
AreaOpImage	An abstract base class for image operators that require only a fixed rectangular source region around a source pixel in order to compute each destination pixel.
BorderExtender	An abstract superclass for classes that extend a `WritableRaster` with additional pixel data taken from a `PlanarImage`.
BorderExtenderConstant	A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with constant values.
BorderExtenderCopy	A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the edge pixels.
BorderExtenderReflect	A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the whole image.
BorderExtenderWrap	A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the whole image.
BorderExtenderZero	A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with zeros.
CanvasJAI	An extension of `java.awt.Canvas` for use with JAI.
CollectionImage	An abstract superclass for classes representing a collection of images.
CollectionOp	A node in either a rendered or a renderable image chain representing a `CollectionImage`.
ColorCube	A subclass of `LookupTableJAI` which represents a lookup table which is a color cube.
ComponentSampleModelJAI	This class represents image data which is stored such that each sample of a pixel occupies one data element of the DataBuffer.
CoordinateImage	A class representing an image that is associated with a coordinate.
CopyPropertyGenerator	Copy properties from a PlanarImage rendering.
DataBufferDouble	An extension of `DataBuffer` that stores data internally in `double` form.
DataBufferFloat	An extension of DataBuffer that stores data internally in `float` form.

FloatDoubleColorModel	A `ColorModel` class that works with pixel values that represent color and alpha information as separate samples, using float or double elements.
GraphicsJAI	A JAI wrapper for a Graphics2D object derived from a Component.
Histogram	An object for accumulating histogram information on an image.
ImageLayout	A class describing the desired layout of an OpImage.
ImageMIPMap	A class implementing the "MIP map" operation on a `RenderedImage`.
ImagePyramid	A class implementing the "Pyramid" operation on a `RenderedImage`.
ImageSequence	A class representing a sequence of images, each associated with a time stamp and a camera position.
ImageStack	A class representing a stack of images, each associated with a spatial orientation defined in a common coordinate system.
IntegerSequence	A growable sorted integer set.
Interpolation	An object encapsulating a particular algorithm for image interpolation (resampling).
InterpolationBicubic	A class representing bicubic interpolation.
InterpolationBicubic2	A class representing bicubic interpolation using a different polynomial than InterpolationBicubic.
InterpolationBilinear	A class representing bilinear interpolation.
InterpolationNearest	A class representing nearest-neighbor interpolation.
InterpolationTable	A subclass of Interpolation that uses tables to store the interpolation kernels.
JAI	A convenience class for instantiating operations.
JAI.RenderingKey	Rendering hints.
JaiI18N	
KernelJAI	A kernel, used by the Convolve, Ordered Dither, and Error Diffusion operations.
LookupTableJAI	A lookup table object associated with the "Lookup" operation.
MultiResolutionRenderableImage	A RenderableImage that produces renderings based on a set of supplied RenderedImages at various resolutions.
NoParameterDefault	A class that signifies that a parameter has no default value.
NullOpImage	A trivial `OpImage` subclass that simply transmits its source unchanged.
OperationDescriptorImpl	This class provides a concrete implementation of the `OperationDescriptor` interface, and is suitable for subclassing.
OperationGraph	OperationGraph manages a list of products belonging to a particular operation descriptor.
OperationRegistry	A class implementing the translation of operation names into instances of RenderedImageFactory, ContextualRenderedImageFactory and CollectionImageFactory.
OpImage	The parent class for all imaging operations.
ParameterBlockJAI	A convenience subclass of ParameterBlock that allows the use of default parameter values and getting/setting parameters by name.
PartialOrderNode	A node in a directed graph of operations.
PerspectiveTransform	A 2D perspective (or projective) transform, used by various OpImages.
PlanarImage	The fundamental base class representing two-dimensional images.

PointOpImage	An abstract base class for image operators that require only the (x, y) pixel from each source image in order to compute the destination pixel (x, y).
ProductOperationGraph	ProductOperationGraph manages a list of operations (image factories) belonging to a particular product.
PropertyGeneratorFromSource	A class that implements the PropertyGenerator interface.
PropertySourceImpl	A class that implements the PropertySource interface.
RasterAccessor	An adapter class for presenting image data in a ComponentSampleModel format, even if the data isn't stored that way.
RasterFactory	A convenience class for the construction of various types of `WritableRaster` and `SampleModel` objects.
RasterFormatTag	This class encapsulates the information needed for RasterAccessor to understand how a Raster is laid out.
RegistryInitData	
RemoteImage	A sub-class of `PlanarImage` which represents an image on a remote server machine.
RenderableGraphics	An implementation of `Graphics2D` with `RenderableImage` semantics.
RenderableImageAdapter	An adapter class for externally-generated RenderableImages.
RenderableOp	A JAI version of RenderableImageOp.
RenderedImageAdapter	A PlanarImage wrapper for a non-writable RenderedImage.
RenderedOp	A node in a rendered imaging chain.
ROI	The parent class for representations of a region of interest of an image.
ROIShape	A class representing a region of interest within an image as a `Shape`.
ScaleOpImage	A class extending `WarpOpImage` for use by further extension classes that perform image scaling.
SequentialImage	A class representing an image that is associated with a time stamp and a camera position.
Snapshot	A non-public class that holds a portion of the state associated with a SnapshotImage.
SnapshotImage	A class providing an arbitrary number of synchronous views of a possibly changing WritableRenderedImage.
SnapshotProxy	A proxy for Snapshot that calls Snapshot.dispose() when finalized.
SourcelessOpImage	An abstract base class for image operators that have no image sources.
StatisticsOpImage	An abstract base class for image operators that compute statistics on a given region of an image, and with a given sampling rate.
Storage	
Store	
TileCopy	A (Raster, X, Y) tuple.
TiledImage	A concrete implementation of WritableRenderedImage.
TiledImageGraphics	A concrete (i.e., non-abstract) class implementing all the methods of `Graphics2D` (and thus of `Graphics`) with a `TiledImage` as the implicit drawing canvas.
UntiledOpImage	A general class for single-source operations in which the values of all pixels in the source image contribute to the value of each pixel in the destination image.
Warp	A description of an image warp.

WarpAffine	A description of an Affine warp.
WarpCubic	A cubic-based description of an image warp.
WarpGeneralPolynomial	A general polynomial-based description of an image warp.
WarpGrid	A regular grid-based description of an image warp.
WarpOpImage	A general implementation of image warping, and a superclass for other geometric image operations.
WarpPerspective	A description of a perspective (projective) warp.
WarpPolynomial	A polynomial-based description of an image warp.
WarpQuadratic	A quadratic-based description of an image warp.
WritableRasterJAI	
WritableRenderedImageAdapter	A `PlanarImage` wrapper for a `WritableRenderedImage`.

javax.media.jai

Class AreaOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.AreaOpImage
```

public abstract class **AreaOpImage**
extends OpImage

An abstract base class for image operators that require only a fixed rectangular source region around a source pixel in order to compute each destination pixel.

The source and the destination images will occupy the same region of the plane. A given destination pixel (x, y) may be computed from the neighborhood of source pixels beginning at (x - leftPadding, y - topPadding) and extending to (x + rightPadding, y + bottomPadding) inclusive.

Since this operator needs a region around the source pixel in order to compute the destination pixel, the border destination pixels cannot be computed without any source extension. The source extension can be specified by supplying a BorderExtender that will define the pixel values of the source outside the actual source area.

If no extension is specified, the destination samples that cannot be computed will be written in the destination as zero. If the source image begins at pixel (minX, minY) and has width w and height h, the result of performing an area operation will be an image beginning at minX, minY, and having a width of w and a height of h, with the area being computed and written starting at (minX + leftPadding, minY + topPadding) and having width Math.max(w - leftPadding - rightPadding, 0) and height Math.max(h - topPadding - bottomPadding, 0).

See Also:
`BorderExtender`

Field Detail

leftPadding

`protected int leftPadding`

The number of source pixels needed to the left of the central pixel.

rightPadding

`protected int rightPadding`

The number of source pixels needed to the right of the central pixel.

topPadding

`protected int topPadding`

The number of source pixels needed above the central pixel.

bottomPadding

`protected int bottomPadding`

The number of source pixels needed below the central pixel.

extender

`protected BorderExtender extender`

The BorderExtender, may be null.

theDest

```
private java.awt.Rectangle theDest
```

Constructor Detail

AreaOpImage

```
public AreaOpImage(java.awt.image.RenderedImage source,
                   BorderExtender extender,
                   TileCache cache,
                   ImageLayout layout,
                   int leftPadding,
                   int rightPadding,
                   int topPadding,
                   int bottomPadding,
                   boolean cobbleSources)
```

Constructs an `AreaOpImage`. The output min X, min Y, width, and height are copied from the source image. The `SampleModel` and `ColorModel` of the output are set in the standard way by the `OpImage` constructor.

Additional control over the image bounds, tile grid layout, `SampleModel`, and `ColorModel` may be obtained by specifying an `ImageLayout` parameter. This parameter will be passed to the superclass constructor unchanged.

Parameters:

`source` - A `RenderedImage`.
`extender` - A BorderExtender, or null.
`cache` - a TileCache object to store tiles from this OpImage, or null. If null, a default cache will be used.
`layout` - An `ImageLayout` containing the source dimensions before padding, and optionally containing the tile grid layout, `SampleModel`, and `ColorModel`.
`leftPadding` - The desired left padding.
`rightPadding` - The desired right padding.
`topPadding` - The desired top padding.
`bottomPadding` - The desired bottom padding.
`cobbleSources` - A `boolean` indicating whether `computeRect()` expects contiguous sources.

Throws:

java.lang.IllegalArgumentException - if combining the intersected source bounds with the layout parameter results in negative output width or height.

Method Detail

getLeftPadding

```
public int getLeftPadding()
```

Returns the number of pixels needed to the left of the central pixel.

Returns:

The left padding factor.

---

getRightPadding

```
public int getRightPadding()
```

Returns the number of pixels needed to the right of the central pixel.

Returns:

The right padding factor.

---

getTopPadding

```
public int getTopPadding()
```

Returns the number of pixels needed above the central pixel.

Returns:

The top padding factor.

---

getBottomPadding

```
public int getBottomPadding()
```

Returns the number of pixels needed below the central pixel.

Returns:

The bottom padding factor.

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:

`sourceRect` - the `Rectangle` in source coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `sourceRect` is `null`.

Overrides:

mapSourceRect in class OpImage

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Returns a conservative estimate of the region of a specified source that is required in order to compute the pixels of a given destination rectangle.

Parameters:

`destRect` - the `Rectangle` in destination coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the required source region.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class OpImage

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. If source cobbling was requested at construction time, the source tile boundaries are overlayed onto the destination, cobbling is performed for areas that intersect multiple source tiles, and `computeRect(Raster[], WritableRaster, Rectangle)` is called for each of the resulting regions. Otherwise, `computeRect(PlanarImage[], WritableRaster, Rectangle)` is called once to compute the entire active area of the tile.

The image bounds may be larger than the bounds of the source image. In this case, samples for which there are no no corresponding sources are set to zero.

Parameters:

`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Returns:

The tile as a `Raster`.

Overrides:

computeTile in class OpImage

javax.media.jai

Class BorderExtender

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
```

Direct Known Subclasses:

BorderExtenderConstant, BorderExtenderCopy, BorderExtenderReflect, BorderExtenderWrap, BorderExtenderZero

---

public abstract class **BorderExtender**
extends java.lang.Object

An abstract superclass for classes that extend a `WritableRaster` with additional pixel data taken from a `PlanarImage`. Instances of `BorderExtender` are used by the `getExtendedData()` and `copyExtendedData()` methods in `PlanarImage`.

Each instance of `BorderExtender` has an `extend()` method that takes a `WritableRaster` and a `PlanarImage`. The portion of the raster that intersects the bounds of the image will already contain a copy of the image data. The remaining area is to be filled in according to the policy of the `BorderImage` subclass.

The standard subclasses of `BorderExtender` are `BorderExtenderZero`, which fills pixels with zeros; `BorderExtenderConstant`, which fills pixels with a given constant value; `BorderExtenderCopy`, which copies the edge pixels of the image; `BorderExtenderWrap`, which tiles the plane with repeating copies of the image; and `BorderExtenderReflect`, which is like `BorderExtenderWrap` except that each copy of the image is suitably reflected.

Instances of `BorderExtenderConstant` are constructed in the usual way. Instances of the other standard subclasses are obtained by means of the `createInstance()` method of this class.

`BorderExtenderCopy` is particularly useful as a way of padding image data prior to performing area or geometric operations such as convolution, scaling, and rotation.

The standard subclasses of `BorderExtender` are marked as `final` in order to allow for optimizations in their use. It is possible to write new subclasses that implement different extension policies.

See Also:

`PlanarImage.getExtendedData(java.awt.Rectangle, javax.media.jai.BorderExtender)`, `PlanarImage.copyExtendedData(java.awt.image.WritableRaster, javax.media.jai.BorderExtender)`, `BorderExtenderZero`, `BorderExtenderConstant`, `BorderExtenderCopy`, `BorderExtenderReflect`, `BorderExtenderWrap`

Field Detail

BORDER_ZERO

`public static final int` **`BORDER_ZERO`**

A constant for use in the `createInstance` method.

BORDER_COPY

`public static final int` **`BORDER_COPY`**

A constant for use in the `createInstance` method.

BORDER_REFLECT

`public static final int` **`BORDER_REFLECT`**

A constant for use in the `createInstance` method.

BORDER_WRAP

`public static final int` **`BORDER_WRAP`**

A constant for use in the `createInstance` method.

---

borderExtenderZero

```
private static final BorderExtender borderExtenderZero
```

---

borderExtenderCopy

```
private static final BorderExtender borderExtenderCopy
```

---

borderExtenderReflect

```
private static final BorderExtender borderExtenderReflect
```

---

borderExtenderWrap

```
private static final BorderExtender borderExtenderWrap
```

Constructor Detail

BorderExtender

```
public BorderExtender()
```

Method Detail

extend

```
public abstract void extend(java.awt.image.WritableRaster raster,
                            PlanarImage im)
```

Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with data derived from that `PlanarImage`.

The portion of `raster` that lies within `im.getBounds()` must not be altered. The pixels within this region should not be assumed to have any particular values.

Each subclass may implement a different policy regarding how the extension data is computed.

---

createInstance

```
public static BorderExtender createInstance(int extenderType)
```

Returns an instance of `BorderExtender` that implements a given extension policy. The policies understood by this method are:

`BORDER_ZERO`: set sample values to zero.

`BORDER_COPY`: set sample values to copies of the nearest valid pixel. For example, pixels to the left of the valid rectangle will take on the value of the valid edge pixel in the same row. Pixels both above and to the left of the valid rectangle will take on the value of the upper-left pixel.

`BORDER_REFLECT`: the output image is defined as if mirrors were placed along the edges of the source image. Thus if the left edge of the valid rectangle lies at X = 10, pixel (9, Y) will be a copy of pixel (10, Y); pixel (6, Y) will be a copy of pixel (13, Y).

`BORDER_WRAP`: the source image is tiled repeatedly in the plane.

Note that this method may not be used to create an instance of `BorderExtenderConstant`.

Any other input value will cause an `IllegalArgumentException` to be thrown.

javax.media.jai
Class BorderExtenderConstant

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
        |
        +--javax.media.jai.BorderExtenderConstant
```

---

public final class **BorderExtenderConstant**
extends BorderExtender

A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with constant values. For example, the image:

A	B	C
D	E	F
G	H	I

if extended by adding two extra rows to the top and bottom and two extra columns on the left and right sides, would become:

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	A	B	C	X	X	X	X	D	E	F	X	X	X	X	G	H	I	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

where X is the constant fill value. The set of constants is clamped to the range and precision of the data type of the Raster being filled. The number of constants used is given by the number of bands of the Raster. If the Raster has `b` bands, and there are `c` constants, constants `0` through `b - 1` are used when `b <= c`. If `b > c`, zeros are used to fill out the constants array.

See Also:
  `BorderExtender`

Field Detail

constants

`private double[] **constants**`

Constructor Detail

BorderExtenderConstant

`public **BorderExtenderConstant**(double[] constants)`

  Constructs an instance of `BorderExtenderConstant` with a given set of constants. The constants are specified as an array of `double`s.

Method Detail

clamp

```
private int clamp(int band,
                  int min,
                  int max)
```

---

extend

```
public final void extend(java.awt.image.WritableRaster raster,
                         PlanarImage im)
```

  Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with constant values.

  The portion of `raster` that lies within `im.getBounds()` is not altered.

  Overrides:
    extend in class BorderExtender

javax.media.jai
Class BorderExtenderCopy

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
        |
        +--javax.media.jai.BorderExtenderCopy
```

public class **BorderExtenderCopy**
extends BorderExtender

A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the edge pixels. For example, the image:

A	B	C
D	E	F
G	H	I

if extended by adding two extra rows to the top and bottom and two extra columns on the left and right sides, would become:

A	A	A	B	C	C	C	A	A	A	B	C	C	C	A	A	A	B	C	C	C	D	D	D	E	F	F	F	G	G	G	H	I	I	I	G	G	G	H	I	I	I	G	G	G	H	I	I	I

Although this type of extension is not particularly visually appealing, it is very useful as a way of padding source images prior to area or geometric operations, such as convolution, scaling, or rotation.

See Also:
    `BorderExtender`

Constructor Detail

BorderExtenderCopy

```
BorderExtenderCopy()
```

Method Detail

extend

```
public final void extend(java.awt.image.WritableRaster raster,
                         PlanarImage im)
```

Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with copies of the edge pixels of the image.

The portion of `raster` that lies within `im.getBounds()` is not altered.

Overrides:
    extend in class BorderExtender

javax.media.jai
Class BorderExtenderReflect

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
        |
        +--javax.media.jai.BorderExtenderReflect
```

public class **BorderExtenderReflect**
extends BorderExtender

A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the whole image. For example, the image:

|> |\

if extended by adding two extra rows to the top and bottom and one extra column on the left and right sides, would become:

<| /|	|> |\	<| /|
\\| <|	|/ |>	\\| <|
<| /|	|> |\	<| /|
\\| <|	|/ |>	\\| <|
<| /|	|> |\	<| /|

This form of extension avoids discontinuities around the edges of the image.

Constructor Detail

BorderExtenderReflect

```
BorderExtenderReflect()
```

Method Detail

flipX

```
private void flipX(java.awt.image.WritableRaster raster)
```

flipY

```
private void flipY(java.awt.image.WritableRaster raster)
```

extend

```
public final void extend(java.awt.image.WritableRaster raster,
                         PlanarImage im)
```

Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with suitably reflected copies of the entire image.

The portion of `raster` that lies within `im.getBounds()` is not altered.

Overrides:
extend in class BorderExtender

javax.media.jai

Class BorderExtenderWrap

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
        |
        +--javax.media.jai.BorderExtenderWrap
```

public class **BorderExtenderWrap**
extends BorderExtender

A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with copies of the whole image. For example, the image:

A	B	C
D	E	F
G	H	I

if extended by adding two extra rows to the top and bottom and two extra columns on the left and right sides, would become:

E	F	D	E	F	D	E	H	I	G	H	I	G	H	B	C	A	B	C	A	B	E	F	D	E	F	D	E	H	I	G	H	I	G	H	B	C	A	B	C	A	B	E	F	D	E	F	D	E

This form of extension is appropriate for data that is inherently periodic, such as the Fourier transform of an image, or a wallpaper pattern.

See Also:
`BorderExtender`

Constructor Detail

BorderExtenderWrap

```
BorderExtenderWrap()
```

Method Detail

extend

```
public final void extend(java.awt.image.WritableRaster raster,
                         PlanarImage im)
```

Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with copies of the entire image.

The portion of `raster` that lies within `im.getBounds()` is not altered.

Overrides:
extend in class BorderExtender

javax.media.jai
Class BorderExtenderZero

```
java.lang.Object
  |
  +--javax.media.jai.BorderExtender
        |
        +--javax.media.jai.BorderExtenderZero
```

public final class **BorderExtenderZero**
extends BorderExtender

A subclass of `BorderExtender` that implements border extension by filling all pixels outside of the image bounds with zeros. For example, the image:

A	B	C
D	E	F
G	H	I

if extended by adding two extra rows to the top and bottom and two extra columns on the left and right sides, would become:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	B	C	0	0	0	0	D	E	F	0	0	0	0	G	H	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

See Also:
    `BorderExtender`

Constructor Detail

BorderExtenderZero

```
BorderExtenderZero()
```

Method Detail

extend

```
public final void extend(java.awt.image.WritableRaster raster,
                         PlanarImage im)
```

Fills in the portions of a given `Raster` that lie outside the bounds of a given `PlanarImage` with zeros.

The portion of `raster` that lies within `im.getBounds()` is not altered.

Overrides:
    extend in class BorderExtender

javax.media.jai

Class CanvasJAI

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Canvas
              |
              +--javax.media.jai.CanvasJAI
```

public class **CanvasJAI**
extends java.awt.Canvas

An extension of `java.awt.Canvas` for use with JAI. `CanvasJAI` automatically returns an instance of `GraphicsJAI` from its `getGraphics()` method. This guarantees that the `update(Graphics g)` and `paint(Graphics g)` methods will receive a `GraphicsJAI` instance for accelerated rendering of `JAI` images.

In circumstances where it is not possible to use `CanvasJAI`, a similar effect may be obtained by manually calling `GraphicsJAI.createGraphicsJAI()` to "wrap" a `Graphics2D` object.

See Also:
`GraphicsJAI`

Constructor Detail

CanvasJAI

`public CanvasJAI(java.awt.GraphicsConfiguration config)`

Constructs an instance of `CanvasJAI` using the given `GraphicsConfiguration`.

Method Detail

getGraphics

`public java.awt.Graphics getGraphics()`

Returns an instance of `GraphicsJAI` for drawing to this canvas.

Overrides:
getGraphics in class java.awt.Component

javax.media.jai
Class CollectionImage

```
java.lang.Object
  |
  +--javax.media.jai.CollectionImage
```

Direct Known Subclasses:
CollectionOp, ImageSequence, ImageStack

---

public abstract class **CollectionImage**
extends java.lang.Object
implements ImageJAI, java.util.Collection

An abstract superclass for classes representing a collection of images. It may be a collection of rendered or renderable images, a collection of collections that include images. In other words, this class supports nested collections, but at the very bottom, there must be images associated with the collection objects.

Field Detail

imageCollection

`protected java.util.Collection imageCollection`

A collection of objects. It may be a collection of images of the same type, a collection of objects of the same type, each contains an image, or a collection of collections whose leaf objects are images or objects that contain images.

Constructor Detail

CollectionImage

`protected CollectionImage()`

Default constructor. The `imageCollection` parameter is `null`. Subclasses that use this constructor must either set the `imageCollection` parameter themselves, or override the methods defined in `Collection` interface. Otherwise, `NullPointerException` may be thrown.

CollectionImage

`public CollectionImage(java.util.Collection collection)`

Constructs a class that contains an image collection.

Parameters:
`collection` - A collection of objects that include images.

Throws:
NullPointerException - if `collection` is `null`.

Method Detail

getPropertyNames

`public java.lang.String[] getPropertyNames()`

Returns an array of Strings recognized as names by this property source. If no property names match, `null` will be returned. The default implementation returns `null`, i.e., no property names are recognized.

Returns:
An array of Strings giving the valid property names.

---

getPropertyNames

`public java.lang.String[] getPropertyNames(java.lang.String prefix)`

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names are recognized, or no property names match, `null` will be returned. The comparison is done in a case-independent manner.

The default implementation calls `getPropertyNames` and searches the list of names for matches.

Returns:
An array of `String`s giving the valid property names.
Throws:
NullPointerException - if `prefix` is `null`.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Returns the specified property. The default implementation returns `java.awt.Image.UndefinedProperty`.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    java.util.Collection collection)
```

Returns the specified property. The default implementation returns `java.awt.Image.UndefinedProperty`.

---

size

```
public int size()
```

Returns the number of elements in this collection.
Specified by:
size in interface java.util.Collection

---

isEmpty

```
public boolean isEmpty()
```

Returns `true` if this collection contains no elements.
Specified by:
isEmpty in interface java.util.Collection

---

contains

```
public boolean contains(java.lang.Object o)
```

Returns `true` if this collection contains the specified object.
Specified by:
contains in interface java.util.Collection

---

iterator

```
public java.util.Iterator iterator()
```

Returns an `Iterator` over the elements in this collection.
Specified by:
iterator in interface java.util.Collection

---

toArray

```
public java.lang.Object[] toArray()
```

Returns an array containing all of the elements in this collection.
Specified by:
toArray in interface java.util.Collection

---

toArray

```
public java.lang.Object[] toArray(java.lang.Object[] a)
```

Returns an array containing all of the elements in this collection whose runtime type is that of the specified array.
Specified by:
toArray in interface java.util.Collection
Throws:
ArrayStoreException - if the runtime type of the specified array is not a supertype of the runtime type of every element in this collection.

add

`public boolean **add**(java.lang.Object o)`

Adds the specified object to this collection.

Specified by:

add in interface java.util.Collection

Returns:

`true` if and only if the parameter is added to the collection.

remove

`public boolean **remove**(java.lang.Object o)`

Removes the specified object from this collection.

Specified by:

remove in interface java.util.Collection

Returns:

`true` if and only if the parameter is removed from the collection.

containsAll

`public boolean **containsAll**(java.util.Collection c)`

Returns `true` if this collection contains all of the elements in the specified collection.

Specified by:

containsAll in interface java.util.Collection

addAll

`public boolean **addAll**(java.util.Collection c)`

Adds all of the elements in the specified collection to this collection.

Specified by:

addAll in interface java.util.Collection

Returns:

`true` if this collection changed as a result of the call.

removeAll

`public boolean **removeAll**(java.util.Collection c)`

Removes all this collection’s elements that are also contained in the specified collection.

Specified by:

removeAll in interface java.util.Collection

Returns:

`true` if this collection changed as a result of the call.

retainAll

`public boolean **retainAll**(java.util.Collection c)`

Retains only the elements in this collection that are contained in the specified collection.

Specified by:

retainAll in interface java.util.Collection

Returns:

`true` if this collection changed as a result of the call.

clear

`public void **clear**()`

Removes all of the elements from this collection.

Specified by:

clear in interface java.util.Collection

javax.media.jai
Interface CollectionImageFactory

public abstract interface **CollectionImageFactory**

The `CollectionImageFactory` interface (often abbreviated CIF) is intended to be implemented by classes that wish to act as factories to produce different collection image operators.

Method Detail

create

```
public CollectionImage create(java.awt.image.renderable.ParameterBlock args,
                              java.awt.RenderingHints hints)
```

Creates a `CollectionImage` that represents the result of an operation (or chain of operations) for a given `ParameterBlock` and `RenderingHints`. If the operation is unable to handle the input arguments, this method should return `null`.

Parameters:

`args` - Input arguments to the operation, including sources and/or parameters.

`hints` - The rendering hints.

Returns:

A `CollectionImage` containing the desired output.

javax.media.jai

Class CollectionOp

```
java.lang.Object
  |
  +--javax.media.jai.CollectionImage
        |
        +--javax.media.jai.CollectionOp
```

public class **CollectionOp**
extends CollectionImage

A node in either a rendered or a renderable image chain representing a `CollectionImage`.

This class stores an `OperationRegistry` that is used to render this node, the name of the operation in the form of a `String`, a `ParameterBlock` that contains the input sources and parameters to the operation, and a `RenderingHints` that contains the hints used with the rendering.

The `OperationRegistry` may be specified at the construction time of this class, or later using the `setRegistry` method. If the registry is not specified, the default registry is used to render this node.

When any of the `Collection` methods is called on this class, this node is implicitly rendered and frozen. The result returned is the result of the rendered image collection. The `getCollection` method also causes this node to be rendered and frozen.

This node may be rendered explicitly by means of the `createInstance()` method. This method returns a `Collection` rendering without freezing the node. This allows a chain to be manipulated dynamically and rendered multiple times.

See Also:
`CollectionImage`, `RenderableOp`, `RenderedOp`

Field Detail

registry

`private OperationRegistry registry`

The `OperationRegistry` that is used to render this node.

opName

`private java.lang.String opName`

The name of the operation this node represents.

args

`private java.awt.image.renderable.ParameterBlock args`

The input arguments for this operation, including sources and/or parameters.

hints

`private java.awt.RenderingHints hints`

The rendering hints to use for this operation.

Constructor Detail

CollectionOp

```
public CollectionOp(OperationRegistry registry,
                    java.lang.String opName,
                    java.awt.image.renderable.ParameterBlock args,
                    java.awt.RenderingHints hints)
```

Constructs a `CollectionOp` that will be used to instantiate a particular collection operation from a given operation registry, an operation name, a `ParameterBlock`, and a set of rendering hints. All input parameters are saved by reference.

Parameters:
`registry` - The `OperationRegistry` to be used for instantiation. if `null`, the default registry is used.
`opName` - The operation name.
`args` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.
`hints` - The rendering hints. If `null`, it is assumed that no hints are associated with the rendering.

Throws:
NullPointerException - if `opName` is `null`.

CollectionOp

```
public CollectionOp(java.lang.String opName,
                    java.awt.image.renderable.ParameterBlock args,
                    java.awt.RenderingHints hints)
```

Constructs a `CollectionOp` that will be used to instantiate a particular collection operation from a given operation name, a `ParameterBlock`, and a set of rendering hints. The default operation registry is used. All input parameters are saved by reference.

Parameters:
`opName` - The operation name.
`args` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.
`hints` - The rendering hints. If `null`, it is assumed that no hints are associated with the rendering.

Throws:
NullPointerException - if `opName` is `null`.

CollectionOp

```
public CollectionOp(OperationRegistry registry,
                    java.lang.String opName,
                    java.awt.image.renderable.ParameterBlock args)
```

Constructs a `CollectionOp` that will be used to instantiate a particular collection operation from a given operation registry, an operation name, and a `ParameterBlock` There is no rendering hints associated with this operation. All input parameters are saved by reference.

Parameters:
`registry` - The `OperationRegistry` to be used for instantiation. if `null`, the default registry is used.
`opName` - The operation name.
`args` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.

Throws:
NullPointerException - if `opName` is `null`.

Method Detail

getRegistry

```
public OperationRegistry getRegistry()
```

Returns the `OperationRegistry` that is used by this node. If the registry had not been set, the default registry is returned.

---

setRegistry

```
public void setRegistry(OperationRegistry registry)
```

Sets the `OperationRegistry` that is used by this node. If the specified registry is `null`, the default registry is used. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:
`registry` - The new `OperationRegistry` to be set; it may be `null`.

---

getOperationName

```
public java.lang.String getOperationName()
```

Returns the name of the operation this node represents as a `String`.

---

setOperationName

```
public void setOperationName(java.lang.String opName)
```

Sets the name of the operation this node represents. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:
`opName` - The new operation name to be set.

Throws:
NullPointerException - if `opName` is `null`.

getParameterBlock

`public java.awt.image.renderable.ParameterBlock` **`getParameterBlock`**`()`

Returns the `ParameterBlock` of this node.

---

setParameterBlock

`public void` **`setParameterBlock`**`(java.awt.image.renderable.ParameterBlock pb)`

Sets the `ParameterBlock` of this node. If this node has been rendered and frozen, this method has no effect. If the speicifed new `ParameterBlock` is `null`, it is assumed that this node has no input sources and parameters. The parameter is saved by reference.

This method does not validate the content of the supplied `ParameterBlock`. The caller should ensure that the sources and parameters in the `ParameterBlock` are suitable for the operation this node represents; otherwise some form of error or exception may occur at the time of rendering.

Parameters:

`pb` - The new `ParameterBlock` to be set; it may be `null`.

---

getRenderingHints

`public java.awt.RenderingHints` **`getRenderingHints`**`()`

Returns the `RenderingHints` of this node. It may be `null`.

---

setRenderingHints

`public void` **`setRenderingHints`**`(java.awt.RenderingHints hints)`

Sets the `RenderingHints` of this node. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:

`hints` - The new `RenderingHints` to be set; it may be `null`.

---

getCollection

`public java.util.Collection` **`getCollection`**`()`

Returns the collection rendering associated with this operation.

---

createCollection

`private void` **`createCollection`**`()`

Creates a collection rendering if none exists.

---

createInstance

`public java.util.Collection` **`createInstance`**`()`

Instantiates a collection operator that computes the result of this `CollectionOp`.

This method does not validate the sources and parameters stored in the `ParameterBlock` against the specification of the operation this node represents. It is the responsibility of the caller to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, some kind of exception or error will occur.

---

createInstance

`private java.util.Collection` **`createInstance`**`(boolean isChainFrozen)`

This method performs the actions described by the documentation of createInstance() optionally freezing the image chain as a function of the parameter.

---

size

`public int size()`

Returns the number of elements in this collection.

Overrides:

size in class CollectionImage

---

isEmpty

`public boolean isEmpty()`

Returns `true` if this collection contains no element.

Overrides:

isEmpty in class CollectionImage

---

contains

`public boolean contains(java.lang.Object o)`

Returns `true` if this collection contains the specified object.

Overrides:

contains in class CollectionImage

---

iterator

`public java.util.Iterator iterator()`

Returns an iterator over the elements in this collection.

Overrides:

iterator in class CollectionImage

---

toArray

`public java.lang.Object[] toArray()`

Returns an array containing all of the elements in this collection.

Overrides:

toArray in class CollectionImage

---

toArray

`public java.lang.Object[] toArray(java.lang.Object[] a)`

Returns an array containing all of the elements in this collection whose runtime type is that of the specified array.

Throws:

ArrayStoreException - if the runtime type of the specified array is not a supertype of the runtime type of every element in this collection.

Overrides:

toArray in class CollectionImage

---

add

`public boolean add(java.lang.Object o)`

Adds the specified object to this collection.

Overrides:

add in class CollectionImage

---

remove

`public boolean remove(java.lang.Object o)`

Removes the specified object from this collection.

Overrides:

remove in class CollectionImage

---

containsAll

`public boolean **containsAll**(java.util.Collection c)`

Returns `true` if this collection contains all of the elements in the specified collection.
Overrides:
containsAll in class CollectionImage

---

addAll

`public boolean **addAll**(java.util.Collection c)`

Adds all of the elements in the specified collection to this collection.
Overrides:
addAll in class CollectionImage

---

removeAll

`public boolean **removeAll**(java.util.Collection c)`

Removes all this collection’s elements that are also contained in the specified collection.
Overrides:
removeAll in class CollectionImage

---

retainAll

`public boolean **retainAll**(java.util.Collection c)`

Retains only the elements in this collection that are contained in the specified collection.
Overrides:
retainAll in class CollectionImage

---

clear

`public void **clear**()`

Removes all of the elements from this collection.
Overrides:
clear in class CollectionImage

javax.media.jai

Class ColorCube

```
java.lang.Object
  |
  +--javax.media.jai.LookupTableJAI
        |
        +--javax.media.jai.ColorCube
```

public class **ColorCube**
extends LookupTableJAI

A subclass of `LookupTableJAI` which represents a lookup table which is a color cube. A color cube provides a fixed, invertible mapping between table indices and sample values. This allows the `findNearestEntry` method to be implemented more efficiently than in the general case.

All constructors are protected. The correct way to create a `ColorCube` is to use one of the static `create` methods defined in this class.

See Also:
`LookupTableJAI`

Field Detail

BYTE_496

`public static final ColorCube` **`BYTE_496`**

A `ColorCube` for dithering RGB byte data into 216 colors. The offset of this `ColorCube` is 38.

BYTE_855

`public static final ColorCube` **`BYTE_855`**

A `ColorCube` for dithering YCC byte data into 200 colors. The offset of this `ColorCube` is 54.

dimension

`private int[]` **`dimension`**

The signed array of sizes used to create the `ColorCube`.

dimsLessOne

`private int[]` **`dimsLessOne`**

An array of positive values each of whose elements is one less than the absolute value of the corresponding element of the dimension array.

multipliers

`private int[]` **`multipliers`**

An array of multipliers.

The magnitudes of the elements of the multiplier array are defined as `multipliers[0] = 1` and `multipliers[i] = multipliers[i-1]*Math.abs(dimension[i-1])` where `i > 0`. The elements are subsequently assigned the same sign (positive or negative) as the corresponding elements of the dimension array.

adjustedOffset

`private int` **`adjustedOffset`**

An offset into the lookup table, accounting for negative dimensions.

dataType

```
private int dataType
```

The data type cached to accelerate findNearestEntry().

---

numBands

```
private int numBands
```

The number of bands cached to accelerate findNearestEntry().

Constructor Detail

ColorCube

```
protected ColorCube(byte[][] data,
                    int offset)
```

Returns a multi-banded byte `ColorCube` with an index offset common to all bands.

Parameters:

`data` - The multi-banded byte data in [band][index] format.
`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

ColorCube

```
protected ColorCube(short[][] data,
                    int offset,
                    boolean isUShort)
```

Returns a multi-banded short or unsigned short `ColorCube` with an index offset common to all bands.

Parameters:

`data` - The multi-banded short data in [band][index] format.
`offset` - The common offset for all bands.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:

NullPointerException - if data is null.

ColorCube

```
protected ColorCube(int[][] data,
                    int offset)
```

Returns a multi-banded int `ColorCube` with an index offset common to all bands.

Parameters:

`data` - The multi-banded int data in [band][index] format.
`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

ColorCube

```
protected ColorCube(float[][] data,
                    int offset)
```

Returns a multi-banded float `ColorCube` with an index offset common to all bands.

Parameters:

`data` - The multi-banded float data in [band][index] format.
`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

ColorCube

```
protected ColorCube(double[][] data,
                    int offset)
```

Returns a multi-banded double `ColorCube` with an index offset common to all bands.

Parameters:

`data` - The multi-banded double data in [band][index] format.
`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

Method Detail

createColorCube

```
public static ColorCube createColorCube(int dataType,
                                        int offset,
                                        int[] dimension)
```

Returns a multi-banded `ColorCube` of a specified data type.

Parameters:

`dataType` - the data type of the `ColorCube`, one of `DataBuffer.TYPE_BYTE`, `TYPE_SHORT`, `TYPE_USHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`offset` - The common offset for all bands.
`dimension` - The signed dimension of each band.

Returns:

An appropriate `ColorCube`.

Throws:

NullPointerException - if dimension is null.

---

createColorCube

```
public static ColorCube createColorCube(int dataType,
                                        int[] dimension)
```

Returns a multi-banded `ColorCube` of a specified data type with zero offset for all bands.

Parameters:

`dataType` - The data type of the `ColorCube`.
`dimension` - The signed dimension of each band.

Returns:

An appropriate `ColorCube`.

Throws:

NullPointerException - if dimension is null.

---

createColorCubeByte

```
private static ColorCube createColorCubeByte(int offset,
                                             int[] dimension)
```

Returns a multi-banded byte `ColorCube` with an index offset common to all bands.

Parameters:

`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A multi-banded byte `ColorCube` with offset.

---

createColorCubeShort

```
private static ColorCube createColorCubeShort(int offset,
                                              int[] dimension)
```

Returns a multi-banded short `ColorCube` with an index offset common to all bands.

Parameters:

`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:
A multi-banded short `ColorCube` with offset.

---

createColorCubeUShort

```
private static ColorCube createColorCubeUShort(int offset,
                                               int[] dimension)
```

Returns a multi-banded unsigned short `ColorCube` with an index offset common to all bands.

Parameters:
`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:
A multi-banded unsigned short `ColorCube` with offset.

---

createColorCubeInt

```
private static ColorCube createColorCubeInt(int offset,
                                            int[] dimension)
```

Returns a multi-banded int `ColorCube` with an index offset common to all bands.

Parameters:
`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:
A multi-banded int `ColorCube` with offset.

---

createColorCubeFloat

```
private static ColorCube createColorCubeFloat(int offset,
                                              int[] dimension)
```

Returns a multi-banded float `ColorCube` with an index offset common to all bands.

Parameters:
`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:
A multi-banded float `ColorCube` with offset.

---

createColorCubeDouble

```
private static ColorCube createColorCubeDouble(int offset,
                                               int[] dimension)
```

Returns a multi-banded double `ColorCube` with an index offset common to all bands.

Parameters:
`offset` - The common offset for all bands.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:
A multi-banded double `ColorCube`.

---

createDataArray

```
private static java.lang.Object createDataArray(int dataType,
                                                int offset,
                                                int[] dimension)
```

Constructs a two-dimensional array of the requested data type which represents the contents of a color cube.

Parameters:
`dataType` - The data type as defined by the static TYPE fields of `DataBuffer`, e.g., `DataBuffer.TYPE_BYTE`.
`offset` - The initial offset into the data array.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be

increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional array of the requested data type laid out in color cube format.

See Also:

`DataBuffer`

---

createDataArrayByte

```
private static byte[][] createDataArrayByte(int offset,
                                            int[] dimension)
```

Constructs a two-dimensional array of byte data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.
`dimension` - An array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional byte array of color cube data.

---

createDataArrayShort

```
private static short[][] createDataArrayShort(int offset,
                                              int[] dimension)
```

Constructs a two-dimensional array of short data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.
`dimension` - an array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional short array of color cube data.

---

createDataArrayUShort

```
private static short[][] createDataArrayUShort(int offset,
                                               int[] dimension)
```

Constructs a two-dimensional array of unsigned short data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.
`dimension` - an array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional short array of color cube data.

---

createDataArrayInt

```
private static int[][] createDataArrayInt(int offset,
                                          int[] dimension)
```

Constructs a two-dimensional array of int data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.
`dimension` - an array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional int array of color cube data.

---

createDataArrayFloat

```
private static float[][] createDataArrayFloat(int offset,
                                              int[] dimension)
```

Constructs a two-dimensional array of float data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.

`dimension` - an array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional float array of color cube data.

---

createDataArrayDouble

```
private static double[][] createDataArrayDouble(int offset,
                                                int[] dimension)
```

Constructs a two-dimensional array of double data which represent the contents of a color cube.

Parameters:

`offset` - The initial offset into the data array.

`dimension` - an array of signed sizes of each side of the color cube. The color ramp in each dimension will be increasing or decreasing according to whether the sign of the corresponding element of the `dimension` array is positive or negative, respectively.

Returns:

A two-dimensional double array of color cube data.

---

initFields

```
private void initFields(int offset,
                        int[] dimension)
```

Initialize the fields of a `ColorCube`.

Parameters:

`offset` - The common offset for all bands.

`dimension` - The signed dimension for each band.

---

getDimension

```
public int[] getDimension()
```

Returns the array of signed dimensions used to construct the `ColorCube`.

Returns:

the dimension array used to create the `ColorCube`.

---

getDimsLessOne

```
public int[] getDimsLessOne()
```

Returns an array containing the signed dimensions, less one.

Returns:

An array of `int`s.

---

getMultipliers

```
public int[] getMultipliers()
```

Get the multipliers as an array.

Returns:

the array of multipliers.

---

getAdjustedOffset

```
public int getAdjustedOffset()
```

Get the adjusted offset into the lookup table, accounting for negative dimensions.

Returns:

The adjusted offset.

findNearestEntry

```
public int findNearestEntry(float[] pixel)
```

Find the index of the nearest color in the color map to the pixel value argument.

Parameters:

`pixel` - a float array of all samples of a pixel.

Returns:

the index of the nearest color.

Throws:

NullPointerException - if pixel is null.

Overrides:

findNearestEntry in class LookupTableJAI

javax.media.jai
Class ComponentSampleModelJAI

```
java.lang.Object
  |
  +--java.awt.image.SampleModel
        |
        +--java.awt.image.ComponentSampleModel
              |
              +--javax.media.jai.ComponentSampleModelJAI
```

public class **ComponentSampleModelJAI**
extends java.awt.image.ComponentSampleModel

This class represents image data which is stored such that each sample of a pixel occupies one data element of the DataBuffer. It stores the N samples which make up a pixel in N separate data array elements. Different bands may be in different banks of the DataBuffer. Accessor methods are provided so that image data can be manipulated directly. This class can support different kinds of interleaving, e.g. band interleaving, scanline interleaving, and pixel interleaving. Pixel stride is the number of data array elements between two samples for the same band on the same scanline. Scanline stride is the number of data array elements between a given sample and the corresponding sample in the same column of the next scanline. Band offsets denote the number of data array elements from the first data array element of the bank of the DataBuffer holding each band to the first sample of the band. The bands are numbered from 0 to N-1. This class can represent image data for the dataTypes enumerated in java.awt.image.DataBuffer (all samples of a given ComponentSampleModel are stored with the same precision) . All strides and offsets must be non-negative.

See Also:
`ComponentSampleModel`

Constructor Detail

ComponentSampleModelJAI

```
public ComponentSampleModelJAI(int dataType,
                               int w,
                               int h,
                               int pixelStride,
                               int scanlineStride,
                               int[] bandOffsets)
```

Constructs a ComponentSampleModel with the specified parameters. The number of bands will be given by the length of the bandOffsets array. All bands will be stored in the first bank of the DataBuffer.

Parameters:
`dataType` - The data type for storing samples.
`w` - The width (in pixels) of the region of image data described.
`h` - The height (in pixels) of the region of image data described.
`pixelStride` - The pixel stride of the region of image data described.
`scanlineStride` - The line stride of the region of image data described.
`bandOffsets` - The offsets of all bands.

ComponentSampleModelJAI

```
public ComponentSampleModelJAI(int dataType,
                               int w,
                               int h,
                               int pixelStride,
                               int scanlineStride,
                               int[] bankIndices,
                               int[] bandOffsets)
```

Constructs a ComponentSampleModel with the specified parameters. The number of bands will be given by the length of the bandOffsets array. Different bands may be stored in different banks of the DataBuffer.

Parameters:
`dataType` - The data type for storing samples.
`w` - The width (in pixels) of the region of image data described.
`h` - The height (in pixels) of the region of image data described.
`pixelStride` - The pixel stride of the region of image data described.
`scanlineStride` - The line stride of the region of image data described.
`bankIndices` - The bank indices of all bands.
`bandOffsets` - The band offsets of all bands.

Method Detail

getBufferSize

```
private long getBufferSize()
```

Returns the size of the data buffer (in data elements) needed for a data buffer that matches this ComponentSampleModel.

Overrides:

getBufferSize in class java.awt.image.ComponentSampleModel

JAIorderBands

```
private int[] JAIorderBands(int[] orig,
                            int step)
```

Preserves band ordering with new step factor...

---

createCompatibleSampleModel

```
public java.awt.image.SampleModel createCompatibleSampleModel(int w,
                                                              int h)
```

Creates a new ComponentSampleModel with the specified width and height. The new SampleModel will have the same number of bands, storage data type, interleaving scheme, and pixel stride as this SampleModel.

Parameters:

`w` - The width in pixels.
`h` - The height in pixels

Overrides:

createCompatibleSampleModel in class java.awt.image.ComponentSampleModel

---

createSubsetSampleModel

```
public java.awt.image.SampleModel createSubsetSampleModel(int[] bands)
```

This creates a new ComponentSampleModel with a subset of the bands of this ComponentSampleModel. The new ComponentSampleModel can be used with any DataBuffer that the existing ComponentSampleModel can be used with. The new ComponentSampleModel/DataBuffer combination will represent an image with a subset of the bands of the original ComponentSampleModel/DataBuffer combination.

Parameters:

`bands` - subset of bands of this ComponentSampleModel

Overrides:

createSubsetSampleModel in class java.awt.image.ComponentSampleModel

---

createDataBuffer

```
public java.awt.image.DataBuffer createDataBuffer()
```

Creates a DataBuffer that corresponds to this ComponentSampleModel. The DataBuffer’s data type, number of banks, and size will be consistent with this ComponentSampleModel.

Overrides:

createDataBuffer in class java.awt.image.ComponentSampleModel

---

getDataElements

```
public java.lang.Object getDataElements(int x,
                                        int y,
                                        java.lang.Object obj,
                                        java.awt.image.DataBuffer data)
```

Returns data for a single pixel in a primitive array of type TransferType. For a ComponentSampleModel, this will be the same as the data type, and samples will be returned one per array element. Generally, obj should be passed in as null, so that the Object will be created automatically and will be of the right primitive data type.

The following code illustrates transferring data for one pixel from DataBuffer `db1`, whose storage layout is described by ComponentSampleModel `csm1`, to DataBuffer `db2`, whose storage layout is described by ComponentSampleModel `csm2`. The transfer will generally be more efficient than using getPixel/setPixel.

```
ComponentSampleModel csm1, csm2;
DataBufferInt db1, db2;
csm2.setDataElements(x, y,
               csm1.getDataElements(x, y, null, db1), db2);
```

Using getDataElements/setDataElements to transfer between two DataBuffer/SampleModel pairs is legitimate if the

SampleModels have the same number of bands, corresponding bands have the same number of bits per sample, and the TransferTypes are the same.

If obj is non-null, it should be a primitive array of type TransferType. Otherwise, a ClassCastException is thrown. An ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds, or if obj is non-null and is not large enough to hold the pixel data.

Parameters:

`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`obj` - If non-null, a primitive array in which to return the pixel data.
`data` - The DataBuffer containing the image data.

Overrides:

getDataElements in class java.awt.image.ComponentSampleModel

---

getDataElements

```
public java.lang.Object getDataElements(int x,
                                        int y,
                                        int w,
                                        int h,
                                        java.lang.Object obj,
                                        java.awt.image.DataBuffer data)
```

Returns the pixel data for the specified rectangle of pixels in a primitive array of type TransferType. For image data supported by the Java 2D API, this will be one of the dataTypes supported by java.awt.image.DataBuffer. Data may be returned in a packed format, thus increasing efficiency for data transfers. Generally, obj should be passed in as null, so that the Object will be created automatically and will be of the right primitive data type.

The following code illustrates transferring data for a rectangular region of pixels from DataBuffer `db1`, whose storage layout is described by SampleModel `sm1`, to DataBuffer `db2`, whose storage layout is described by SampleModel `sm2`. The transfer will generally be more efficient than using getPixels/setPixels.

```
       SampleModel sm1, sm2;
       DataBuffer db1, db2;
       sm2.setDataElements(x, y, w, h, sm1.getDataElements(x, y, w,
                           h, null, db1), db2);
```

Using getDataElements/setDataElements to transfer between two DataBuffer/SampleModel pairs is legitimate if the SampleModels have the same number of bands, corresponding bands have the same number of bits per sample, and the TransferTypes are the same.

If obj is non-null, it should be a primitive array of type TransferType. Otherwise, a ClassCastException is thrown. An ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds, or if obj is non-null and is not large enough to hold the pixel data.

Parameters:

`x` - The minimum X coordinate of the pixel rectangle.
`y` - The minimum Y coordinate of the pixel rectangle.
`w` - The width of the pixel rectangle.
`h` - The height of the pixel rectangle.
`obj` - If non-null, a primitive array in which to return the pixel data.
`data` - The DataBuffer containing the image data.

Overrides:

getDataElements in class java.awt.image.SampleModel

See Also:

`ComponentSampleModel.getNumDataElements()`, `SampleModel.getTransferType()`, `DataBuffer`

---

setDataElements

```
public void setDataElements(int x,
                            int y,
                            java.lang.Object obj,
                            java.awt.image.DataBuffer data)
```

Sets the data for a single pixel in the specified DataBuffer from a primitive array of type TransferType. For a ComponentSampleModel, this will be the same as the data type, and samples are transferred one per array element.

The following code illustrates transferring data for one pixel from DataBuffer `db1`, whose storage layout is described by ComponentSampleModel `csm1`, to DataBuffer `db2`, whose storage layout is described by ComponentSampleModel `csm2`. The transfer will generally be more efficient than using getPixel/setPixel.

```
       ComponentSampleModel csm1, csm2;
       DataBufferInt db1, db2;
       csm2.setDataElements(x, y, csm1.getDataElements(x, y, null, db1),
                            db2);
```

Using getDataElements/setDataElements to transfer between two DataBuffer/SampleModel pairs is legitimate if the

SampleModels have the same number of bands, corresponding bands have the same number of bits per sample, and the TransferTypes are the same.

obj must be a primitive array of type TransferType. Otherwise, a ClassCastException is thrown. An ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds, or if obj is not large enough to hold the pixel data.

Parameters:

`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`obj` - A primitive array containing pixel data.
`data` - The DataBuffer containing the image data.

Overrides:

setDataElements in class java.awt.image.ComponentSampleModel

---

setDataElements

```
public void setDataElements(int x,
                            int y,
                            int w,
                            int h,
                            java.lang.Object obj,
                            java.awt.image.DataBuffer data)
```

Sets the data for a rectangle of pixels in the specified DataBuffer from a primitive array of type TransferType. For image data supported by the Java 2D API, this will be one of the dataTypes supported by java.awt.image.DataBuffer. Data in the array may be in a packed format, thus increasing efficiency for data transfers.

The following code illustrates transferring data for a rectangular region of pixels from DataBuffer `db1`, whose storage layout is described by SampleModel `sm1`, to DataBuffer `db2`, whose storage layout is described by SampleModel `sm2`. The transfer will generally be more efficient than using getPixels/setPixels.

```
        SampleModel sm1, sm2;
        DataBuffer db1, db2;
        sm2.setDataElements(x, y, w, h, sm1.getDataElements(x, y, w, h,
                            null, db1), db2);
```

Using getDataElements/setDataElements to transfer between two DataBuffer/SampleModel pairs is legitimate if the SampleModels have the same number of bands, corresponding bands have the same number of bits per sample, and the TransferTypes are the same.

obj must be a primitive array of type TransferType. Otherwise, a ClassCastException is thrown. An ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds, or if obj is not large enough to hold the pixel data.

Parameters:

`x` - The minimum X coordinate of the pixel rectangle.
`y` - The minimum Y coordinate of the pixel rectangle.
`w` - The width of the pixel rectangle.
`h` - The height of the pixel rectangle.
`obj` - A primitive array containing pixel data.
`data` - The DataBuffer containing the image data.

Overrides:

setDataElements in class java.awt.image.SampleModel

See Also:

`ComponentSampleModel.getNumDataElements()`, `SampleModel.getTransferType()`, `DataBuffer`

---

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      float s,
                      java.awt.image.DataBuffer data)
```

Sets a sample in the specified band for the pixel located at (x,y) in the DataBuffer using a float for input. ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds.

Parameters:

`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`b` - The band to set.
`s` - The input sample as a float.
`data` - The DataBuffer containing the image data.

Throws:
ArrayIndexOutOfBoundsException - if coordinates are not in bounds
Overrides:
setSample in class java.awt.image.SampleModel

---

getSampleFloat

```
public float getSampleFloat(int x,
                            int y,
                            int b,
                            java.awt.image.DataBuffer data)
```

Returns the sample in a specified band for the pixel located at (x,y) as a float. ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds.
Parameters:
`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`b` - The band to return.
`data` - The DataBuffer containing the image data.
Returns:
sample The floating point sample value
Overrides:
getSampleFloat in class java.awt.image.SampleModel

---

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      double s,
                      java.awt.image.DataBuffer data)
```

Sets a sample in the specified band for the pixel located at (x,y) in the DataBuffer using a double for input. ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds.
Parameters:
`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`b` - The band to set.
`s` - The input sample as a double.
`data` - The DataBuffer containing the image data.
Throws:
ArrayIndexOutOfBoundsException - if coordinates are not in bounds
Overrides:
setSample in class java.awt.image.SampleModel

---

getSampleDouble

```
public double getSampleDouble(int x,
                              int y,
                              int b,
                              java.awt.image.DataBuffer data)
```

Returns the sample in a specified band for a pixel located at (x,y) as a double. ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds.
Parameters:
`x` - The X coordinate of the pixel location.
`y` - The Y coordinate of the pixel location.
`b` - The band to return.
`data` - The DataBuffer containing the image data.
Returns:
sample The double sample value
Overrides:
getSampleDouble in class java.awt.image.SampleModel

---

getPixels

```
public double[] getPixels(int x,
                          int y,
                          int w,
                          int h,
                          double[] dArray,
                          java.awt.image.DataBuffer data)
```

Returns all samples for a rectangle of pixels in a double array, one sample per array element. ArrayIndexOutOfBoundsException may be thrown if the coordinates are not in bounds.

Parameters:

`x` - The X coordinate of the upper left pixel location.
`y` - The Y coordinate of the upper left pixel location.
`w` - The width of the pixel rectangle.
`h` - The height of the pixel rectangle.
`dArray` - If non-null, returns the samples in this array.
`data` - The DataBuffer containing the image data.

Overrides:

getPixels in class java.awt.image.SampleModel

---

toString

```
public java.lang.String toString()
```

Returns a String containing the values of all valid fields.

Overrides:

toString in class java.lang.Object

javax.media.jai

Class CoordinateImage

```
java.lang.Object
  |
  +--javax.media.jai.CoordinateImage
```

public class **CoordinateImage**
extends java.lang.Object

A class representing an image that is associated with a coordinate. This class is used with `ImageStack`.

See Also:
`ImageStack`

Field Detail

image

`public PlanarImage image`

The image.

coordinate

`public java.lang.Object coordinate`

The coordinate associated with the image. The type of this parameter is `Object` so that the application may choose any class to represent a coordinate based on the individual’s needs.

Constructor Detail

CoordinateImage

```
public CoordinateImage(PlanarImage pi,
                       java.lang.Object c)
```

Constructor.

Throws:
NullPointerException - if `pi` is `null`.
NullPointerException - if `c` is `null`.

javax.media.jai

Class CopyPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.CopyPropertyGenerator
```

class **CopyPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

Copy properties from a PlanarImage rendering.

Field Detail

im

```
PlanarImage im
```

Constructor Detail

CopyPropertyGenerator

```
public CopyPropertyGenerator(PlanarImage im)
```

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Specified by:
getProperty in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Specified by:
getProperty in interface PropertyGenerator

javax.media.jai
Class DataBufferDouble

```
java.lang.Object
  |
  +--java.awt.image.DataBuffer
        |
        +--javax.media.jai.DataBufferDouble
```

public class **DataBufferDouble**
extends java.awt.image.DataBuffer

An extension of `DataBuffer` that stores data internally in `double` form.

See Also:
`DataBuffer`

Field Detail

bankdata

`protected double[][] bankdata`

The array of data banks.

data

`protected double[] data`

A reference to the default data bank.

Constructor Detail

DataBufferDouble

`public DataBufferDouble(int size)`

Constructs a `double`-based `DataBuffer` with a specified size.

Parameters:
`size` - The number of elements in the `DataBuffer`.

DataBufferDouble

```
public DataBufferDouble(int size,
                        int numBanks)
```

Constructs a `double`-based `DataBuffer` with a specified number of banks, all of which are of a specified size.

Parameters:
`size` - The number of elements in each bank of the `DataBuffer`.
`numBanks` - The number of banks in the `DataBuffer`.

DataBufferDouble

```
public DataBufferDouble(double[] dataArray,
                        int size)
```

Constructs a `double`-based `DataBuffer` with the specified data array. Only the first `size` elements are available for use by this `DataBuffer`. The array must be large enough to hold `size` elements.

Parameters:
`dataArray` - An array of `double`s to be used as the first and only bank of this `DataBuffer`.
`size` - The number of elements of the array to be used.

DataBufferDouble

```
public DataBufferDouble(double[] dataArray,
                        int size,
                        int offset)
```

Constructs a `double`-based `DataBuffer` with the specified data array. Only the elements between `offset` and `offset + size - 1` are available for use by this `DataBuffer`. The array must be large enough to hold `offset + size` elements.

Parameters:

`dataArray` - An array of `double`s to be used as the first and only bank of this `DataBuffer`.
`size` - The number of elements of the array to be used.
`offset` - The offset of the first element of the array that will be used.

DataBufferDouble

```
public DataBufferDouble(double[][] dataArray,
                        int size)
```

Constructs a `double`-based `DataBuffer` with the specified data arrays. Only the first `size` elements of each array are available for use by this `DataBuffer`. The number of banks will be equal `to dataArray.length`.

Parameters:

`dataArray` - An array of arrays of `double`s to be used as the banks of this `DataBuffer`.
`size` - The number of elements of each array to be used.

DataBufferDouble

```
public DataBufferDouble(double[][] dataArray,
                        int size,
                        int[] offsets)
```

Constructs a `double`-based `DataBuffer` with the specified data arrays, size, and per-bank offsets. The number of banks is equal to dataArray.length. Each array must be at least as large as `size plus the corresponding offset. There must be an entry in the offsets array for each data array.`

`Parameters:`

`dataArray - An array of arrays of doubles to be used as the banks of this DataBuffer.`
`size - The number of elements of each array to be used.`
`offsets - An array of integer offsets, one for each bank.`

Method Detail

getData

```
public double[] getData()
```

Returns the default (first) `double` data array.

---

getData

```
public double[] getData(int bank)
```

Returns the data array for the specified bank.

---

getBankData

```
public double[][] getBankData()
```

Returns the data array for all banks.

---

getElem

```
public int getElem(int i)
```

Returns the requested data array element from the first (default) bank as an `int`.

Parameters:

`i` - The desired data array element.

Returns:

The data entry as an `int`.

Overrides:
getElem in class java.awt.image.DataBuffer

---

getElem

```
public int getElem(int bank,
                   int i)
```

Returns the requested data array element from the specified bank as an `int`.

Parameters:
`bank` - The bank number.
`i` - The desired data array element.

Returns:
The data entry as an `int`.

Overrides:
getElem in class java.awt.image.DataBuffer

---

setElem

```
public void setElem(int i,
                    int val)
```

Sets the requested data array element in the first (default) bank to the given `int`.

Parameters:
`i` - The desired data array element.
`val` - The value to be set.

Overrides:
setElem in class java.awt.image.DataBuffer

---

setElem

```
public void setElem(int bank,
                    int i,
                    int val)
```

Sets the requested data array element in the specified bank to the given `int`.

Parameters:
`bank` - The bank number.
`i` - The desired data array element.
`val` - The value to be set.

Overrides:
setElem in class java.awt.image.DataBuffer

---

getElemFloat

```
public float getElemFloat(int i)
```

Returns the requested data array element from the first (default) bank as a `float`.

Parameters:
`i` - The desired data array element.

Returns:
The data entry as a `float`.

Overrides:
getElemFloat in class java.awt.image.DataBuffer

---

getElemFloat

```
public float getElemFloat(int bank,
                          int i)
```

Returns the requested data array element from the specified bank as a `float`.

Parameters:
`bank` - The bank number.
`i` - The desired data array element.

Returns:
The data entry as a `float`.

Overrides:
getElemFloat in class java.awt.image.DataBuffer

setElemFloat

```
public void setElemFloat(int i,
                         float val)
```

Sets the requested data array element in the first (default) bank to the given `float`.

Parameters:

`i` - The desired data array element.
`val` - The value to be set.

Overrides:

setElemFloat in class java.awt.image.DataBuffer

setElemFloat

```
public void setElemFloat(int bank,
                         int i,
                         float val)
```

Sets the requested data array element in the specified bank to the given `float`.

Parameters:

`bank` - The bank number.
`i` - The desired data array element.
`val` - The value to be set.

Overrides:

setElemFloat in class java.awt.image.DataBuffer

getElemDouble

```
public double getElemDouble(int i)
```

Returns the requested data array element from the first (default) bank as a `double`.

Parameters:

`i` - The desired data array element.

Returns:

The data entry as a `double`.

Overrides:

getElemDouble in class java.awt.image.DataBuffer

getElemDouble

```
public double getElemDouble(int bank,
                            int i)
```

Returns the requested data array element from the specified bank as a `double`.

Parameters:

`bank` - The bank number.
`i` - The desired data array element.

Returns:

The data entry as a `double`.

Overrides:

getElemDouble in class java.awt.image.DataBuffer

setElemDouble

```
public void setElemDouble(int i,
                          double val)
```

Sets the requested data array element in the first (default) bank to the given `double`.

Parameters:

`i` - The desired data array element.
`val` - The value to be set.

Overrides:

setElemDouble in class java.awt.image.DataBuffer

setElemDouble

```
public void setElemDouble(int bank,
                          int i,
                          double val)
```

Sets the requested data array element in the specified bank to the given `double`.

Parameters:

`bank` - The bank number.
`i` - The desired data array element.
`val` - The value to be set.

Overrides:

setElemDouble in class java.awt.image.DataBuffer

javax.media.jai

Class DataBufferFloat

```
java.lang.Object
  |
  +--java.awt.image.DataBuffer
        |
        +--javax.media.jai.DataBufferFloat
```

public class **DataBufferFloat**
extends java.awt.image.DataBuffer

An extension of DataBuffer that stores data internally in `float` form.

See Also:
    `DataBuffer`

Field Detail

bankdata

`protected float[][] bankdata`

The array of data banks.

data

`protected float[] data`

A reference to the default data bank.

Constructor Detail

DataBufferFloat

`public DataBufferFloat(int size)`

Constructs a `float`-based `DataBuffer` with a specified size.

Parameters:
`size` - The number of elements in the DataBuffer.

DataBufferFloat

```
public DataBufferFloat(int size,
                       int numBanks)
```

Constructs a `float`-based `DataBuffer` with a specified number of banks, all of which are of a specified size.

Parameters:
`size` - The number of elements in each bank of the `DataBuffer`.
`numBanks` - The number of banks in the `DataBuffer`.

DataBufferFloat

```
public DataBufferFloat(float[] dataArray,
                       int size)
```

Constructs a `float`-based `DataBuffer` with the specified data array. Only the first `size` elements are available for use by this `DataBuffer`. The array must be large enough to hold `size` elements.

Parameters:
`dataArray` - An array of `float`s to be used as the first and only bank of this `DataBuffer`.
`size` - The number of elements of the array to be used.

DataBufferFloat

```
public DataBufferFloat(float[] dataArray,
                       int size,
                       int offset)
```

Constructs a `float`-based `DataBuffer` with the specified data array. Only the elements between `offset` and `offset + size - 1` are available for use by this `DataBuffer`. The array must be large enough to hold `offset + size` elements.

Parameters:

`dataArray` - An array of `float`s to be used as the first and only bank of this `DataBuffer`.
`size` - The number of elements of the array to be used.
`offset` - The offset of the first element of the array that will be used.

DataBufferFloat

```
public DataBufferFloat(float[][] dataArray,
                       int size)
```

Constructs a `float`-based `DataBuffer` with the specified data arrays. Only the first `size` elements of each array are available for use by this `DataBuffer`. The number of banks will be equal to `dataArray.length`.

Parameters:

`dataArray` - An array of arrays of `float`s to be used as the banks of this `DataBuffer`.
`size` - The number of elements of each array to be used.

DataBufferFloat

```
public DataBufferFloat(float[][] dataArray,
                       int size,
                       int[] offsets)
```

Constructs a `float`-based `DataBuffer` with the specified data arrays, size, and per-bank offsets. The number of banks is equal to `dataArray.length`. Each array must be at least as large as `size` plus the corresponding offset. There must be an entry in the offsets array for each data array.

Parameters:

`dataArray` - An array of arrays of `float`s to be used as the banks of this `DataBuffer`.
`size` - The number of elements of each array to be used.
`offsets` - An array of integer offsets, one for each bank.

Method Detail

getData

```
public float[] getData()
```

Returns the default (first) `float` data array.

---

getData

```
public float[] getData(int bank)
```

Returns the data array for the specified bank.

---

getBankData

```
public float[][] getBankData()
```

Returns the data array for all banks.

---

getElem

```
public int getElem(int i)
```

Returns the requested data array element from the first (default) bank as an `int`.

Parameters:

`i` - The desired data array element.

Returns:

The data entry as an `int`.

Overrides:

getElem in class java.awt.image.DataBuffer

getElem

```
public int getElem(int bank,
                   int i)
```

Returns the requested data array element from the specified bank as an `int`.

Parameters:

`bank` - The bank number.

`i` - The desired data array element.

Returns:

The data entry as an `int`.

Overrides:

getElem in class java.awt.image.DataBuffer

setElem

```
public void setElem(int i,
                    int val)
```

Sets the requested data array element in the first (default) bank to the given `int`.

Parameters:

`i` - The desired data array element.

`val` - The value to be set.

Overrides:

setElem in class java.awt.image.DataBuffer

setElem

```
public void setElem(int bank,
                    int i,
                    int val)
```

Sets the requested data array element in the specified bank to the given `int`.

Parameters:

`bank` - The bank number.

`i` - The desired data array element.

`val` - The value to be set.

Overrides:

setElem in class java.awt.image.DataBuffer

getElemFloat

```
public float getElemFloat(int i)
```

Returns the requested data array element from the first (default) bank as a `float`.

Parameters:

`i` - The desired data array element.

Returns:

The data entry as a `float`.

Overrides:

getElemFloat in class java.awt.image.DataBuffer

getElemFloat

```
public float getElemFloat(int bank,
                          int i)
```

Returns the requested data array element from the specified bank as a `float`.

Parameters:

`bank` - The bank number.

`i` - The desired data array element.

Returns:

The data entry as a `float`.

Overrides:

getElemFloat in class java.awt.image.DataBuffer

setElemFloat

```
public void setElemFloat(int i,
                         float val)
```

Sets the requested data array element in the first (default) bank to the given `float`.

Parameters:

`i` - The desired data array element.

`val` - The value to be set.

Overrides:

setElemFloat in class java.awt.image.DataBuffer

---

setElemFloat

```
public void setElemFloat(int bank,
                         int i,
                         float val)
```

Sets the requested data array element in the specified bank to the given `float`.

Parameters:

`bank` - The bank number.

`i` - The desired data array element.

`val` - The value to be set.

Overrides:

setElemFloat in class java.awt.image.DataBuffer

---

getElemDouble

```
public double getElemDouble(int i)
```

Returns the requested data array element from the first (default) bank as a `double`.

Parameters:

`i` - The desired data array element.

Returns:

The data entry as a `double`.

Overrides:

getElemDouble in class java.awt.image.DataBuffer

---

getElemDouble

```
public double getElemDouble(int bank,
                            int i)
```

Returns the requested data array element from the specified bank as a `double`.

Parameters:

`bank` - The bank number.

`i` - The desired data array element.

Returns:

The data entry as a `double`.

Overrides:

getElemDouble in class java.awt.image.DataBuffer

---

setElemDouble

```
public void setElemDouble(int i,
                          double val)
```

Sets the requested data array element in the first (default) bank to the given `double`.

Parameters:

`i` - The desired data array element.

`val` - The value to be set.

Overrides:

setElemDouble in class java.awt.image.DataBuffer

---

setElemDouble

```
public void setElemDouble(int bank,
                          int i,
                          double val)
```

Sets the requested data array element in the specified bank to the given `double`.

Parameters:

`bank` - The bank number.
`i` - The desired data array element.
`val` - The value to be set.

Overrides:

setElemDouble in class java.awt.image.DataBuffer

javax.media.jai

Class FloatDoubleColorModel

```
java.lang.Object
  |
  +--java.awt.image.ColorModel
        |
        +--java.awt.image.ComponentColorModel
              |
              +--javax.media.jai.FloatDoubleColorModel
```

public class **FloatDoubleColorModel**
extends java.awt.image.ComponentColorModel

A `ColorModel` class that works with pixel values that represent color and alpha information as separate samples, using float or double elements. This class can be used with an arbitrary `ColorSpace`. The number of color samples in the pixel values must be same as the number of color components in the `ColorSpace`. There may be a single alpha sample.

Sample values are taken as ranging from 0.0 to 1.0; that is, when converting to 8-bit RGB, a multiplication by 255 is performed and values outside of the range 0-255 are clamped at the closest endpoint.

For maximum efficiency, pixel data being interpreted by this class should be in the sRGB color space. This will result in only the trivial conversion (scaling by 255 and dividing by any premultiplied alpha) to be performed. Other color spaces require more general conversions.

For those methods that use a primitive array pixel representation of type `transferType`, the array length is the same as the number of color and alpha samples. Color samples are stored first in the array followed by the alpha sample, if present. The order of the color samples is specified by the `ColorSpace`. Typically, this order reflects the name of the color space type. For example, for `TYPE_RGB`, index 0 corresponds to red, index 1 to green, and index 2 to blue. The transfer types supported are `DataBuffer.TYPE_FLOAT`, `DataBuffer.TYPE_DOUBLE`.

The translation from pixel values to color/alpha components for display or processing purposes is a one-to-one correspondence of samples to components.

Methods that use a single int pixel representation throw an `IllegalArgumentException`.

A `FloatDoubleColorModel` can be used in conjunction with a `ComponentSampleModelJAI`.

See Also:
`ColorModel`, `ColorSpace`, `ComponentSampleModel`, `ComponentSampleModelJAI`

Field Detail

colorSpace

`java.awt.color.ColorSpace` **`colorSpace`**

colorSpaceType

`int` **`colorSpaceType`**

numColorComponents

`int` **`numColorComponents`**

numComponents

`int` **`numComponents`**

transparency

`int` **`transparency`**

hasAlpha

```
boolean hasAlpha
```

---

isAlphaPremultiplied

```
boolean isAlphaPremultiplied
```

Constructor Detail

FloatDoubleColorModel

```
public FloatDoubleColorModel(java.awt.color.ColorSpace colorSpace,
                             boolean hasAlpha,
                             boolean isAlphaPremultiplied,
                             int transparency,
                             int transferType)
```

Constructs a `ComponentColorModel` from the specified parameters. Color components will be in the specified `ColorSpace`. `hasAlpha` indicates whether alpha information is present. If `hasAlpha` is true, then the boolean `isAlphaPremultiplied` specifies how to interpret color and alpha samples in pixel values. If the boolean is `true`, color samples are assumed to have been multiplied by the alpha sample. The `transparency` specifies what alpha values can be represented by this color model. The `transferType` is the type of primitive array used to represent pixel values.

Parameters:

`colorSpace` - The `ColorSpace` associated with this color model.
`hasAlpha` - If true, this color model supports alpha.
`isAlphaPremultiplied` - If true, alpha is premultiplied.
`transparency` - Specifies what alpha values can be represented by this color model.
`transferType` - Specifies the type of primitive array used to represent pixel values, one of DataBuffer.TYPE_FLOAT or TYPE_DOUBLE.

Throws:

java.lang.IllegalArgumentException - If the transfer type is not DataBuffer.TYPE_FLOAT or TYPE_DOUBLE.

See Also:

`ColorSpace`, `Transparency`

Method Detail

bitsHelper

```
private static int[] bitsHelper(int transferType,
                                java.awt.color.ColorSpace colorSpace,
                                boolean hasAlpha)
```

---

getRed

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

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:

getRed in class java.awt.image.ComponentColorModel

---

getGreen

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

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:

getGreen in class java.awt.image.ComponentColorModel

---

getBlue

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

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:
getBlue in class java.awt.image.ComponentColorModel

---

getAlpha

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

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:
getAlpha in class java.awt.image.ComponentColorModel

---

getRGB

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

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:
getRGB in class java.awt.image.ComponentColorModel

---

clamp

```
private int clamp(float value)
```

---

clamp

```
private int clamp(double value)
```

---

getSample

```
private int getSample(java.lang.Object inData,
                      int sample)
```

---

getRed

```
public int getRed(java.lang.Object inData)
```

Returns the red color component for the specified pixel, scaled from 0 to 255 in the default RGB ColorSpace, sRGB. A color conversion is done if necessary. The `pixel` value is specified by an array of data elements of type `transferType` passed in as an object reference. The returned value will be a non pre-multiplied value. If the alpha is premultiplied, this method divides it out before returning the value (if the alpha value is 0, the red value will be 0).

Parameters:
`inData` - The pixel from which you want to get the red color component, specified by an array of data elements of type `transferType`.

Returns:
The red color component for the specified pixel, as an int.

Throws:
ClassCastException - If `inData` is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - if `inData` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:
getRed in class java.awt.image.ComponentColorModel

---

getGreen

```
public int getGreen(java.lang.Object inData)
```

Returns the green color component for the specified pixel, scaled from 0 to 255 in the default RGB ColorSpace, sRGB. A color conversion is done if necessary. The `pixel` value is specified by an array of data elements of type `transferType` passed in as an object reference. The returned value will be a non pre-multiplied value. If the alpha is premultiplied, this method divides it out before returning the value (if the alpha value is 0, the green value will be 0).

Parameters:
`inData` - The pixel from which you want to get the green color component, specified by an array of data elements of type `transferType`.

Returns:
The green color component for the specified pixel, as an int.

Throws:

ClassCastException - If `inData` is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - if `inData` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:

getGreen in class java.awt.image.ComponentColorModel

---

getBlue

`public int getBlue(java.lang.Object inData)`

Returns the blue color component for the specified pixel, scaled from 0 to 255 in the default RGB ColorSpace, sRGB. A color conversion is done if necessary. The `pixel` value is specified by an array of data elements of type `transferType` passed in as an object reference. The returned value will be a non pre-multiplied value. If the alpha is premultiplied, this method divides it out before returning the value (if the alpha value is 0, the blue value will be 0).

Parameters:

`inData` - The pixel from which you want to get the blue color component, specified by an array of data elements of type `transferType`.

Returns:

The blue color component for the specified pixel, as an int.

Throws:

ClassCastException - If `inData` is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - if `inData` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:

getBlue in class java.awt.image.ComponentColorModel

---

getAlpha

`public int getAlpha(java.lang.Object inData)`

Returns the alpha component for the specified pixel, scaled from 0 to 255. The pixel value is specified by an array of data elements of type `transferType` passed in as an object reference. If the `ColorModel` does not have alpha, 255 is returned.

Parameters:

`inData` - The pixel from which you want to get the alpha component, specified by an array of data elements of type `transferType`.

Returns:

The alpha component for the specified pixel, as an int.

Throws:

NullPointerException - if `inData` is `null` and the `colorModel` has alpha.
ClassCastException - If `inData` is not a primitive array of type `transferType` and the `ColorModel` has alpha.
ArrayIndexOutOfBoundsException - if `inData` is not large enough to hold a pixel value for this `ColorModel` and the `ColorModel` has alpha.

Overrides:

getAlpha in class java.awt.image.ComponentColorModel

---

getRGB

`public int getRGB(java.lang.Object inData)`

Returns the color/alpha components for the specified pixel in the default RGB color model format. A color conversion is done if necessary. The pixel value is specified by an array of data elements of type `transferType` passed in as an object reference. The returned value is in a non pre-multiplied format. If the alpha is premultiplied, this method divides it out of the color components (if the alpha value is 0, the color values will be 0).

Parameters:

`inData` - The pixel from which you want to get the color/alpha components, specified by an array of data elements of type `transferType`.

Returns:

The color/alpha components for the specified pixel, as an int.

Throws:

ClassCastException - If `inData` is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - if `inData` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:

getRGB in class java.awt.image.ComponentColorModel

---

getDataElements

```
public java.lang.Object getDataElements(int rgb,
                                        java.lang.Object pixel)
```

Returns a data element array representation of a pixel in this `ColorModel`, given an integer pixel representation in the default RGB color model. This array can then be passed to the `setDataElements` method of a `WritableRaster` object. If the `pixel` parameter is null, a new array is allocated.

Parameters:

`rgb` - An ARGB value packed into an int.
`pixel` - The float or double array representation of the pixel.

Throws:

ClassCastException - If `pixel` is not null and is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - If `pixel` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:

getDataElements in class java.awt.image.ComponentColorModel

---

getComponents

```
public int[] getComponents(int pixel,
                           int[] components,
                           int offset)
```

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:

getComponents in class java.awt.image.ComponentColorModel

---

getComponents

```
public int[] getComponents(java.lang.Object pixel,
                           int[] components,
                           int offset)
```

Throws an `IllegalArgumentException` since the pixel values cannot be placed into an `int` array.

Overrides:

getComponents in class java.awt.image.ComponentColorModel

---

getDataElement

```
public int getDataElement(int[] components,
                          int offset)
```

Throws an `IllegalArgumentException`, since pixel values for this `ColorModel` are not conveniently representable as a single `int`.

Overrides:

getDataElement in class java.awt.image.ComponentColorModel

---

getDataElements

```
public java.lang.Object getDataElements(int[] components,
                                        int offset,
                                        java.lang.Object obj)
```

Returns a data element array representation of a pixel in this `ColorModel`, given an array of unnormalized color/alpha components. This array can then be passed to the `setDataElements` method of a `WritableRaster` object.

Parameters:

`components` - An array of unnormalized color/alpha components.
`offset` - The integer offset into the `components` array.
`obj` - The object in which to store the data element array representation of the pixel. If `obj` variable is null, a new array is allocated. If `obj` is not null, it must be a primitive array of type `transferType`. An `ArrayIndexOutOfBoundsException` is thrown if `obj` is not large enough to hold a pixel value for this `ColorModel`.

Returns:

The data element array representation of a pixel in this `ColorModel`.

Throws:

java.lang.IllegalArgumentException - If the components array is not large enough to hold all the color and alpha components (starting at offset).
ClassCastException - If `obj` is not null and is not a primitive array of type `transferType`.
ArrayIndexOutOfBoundsException - If `obj` is not large enough to hold a pixel value for this `ColorModel`.

Overrides:
getDataElements in class java.awt.image.ComponentColorModel

---

coerceData

```
public java.awt.image.ColorModel coerceData(java.awt.image.WritableRaster raster,
                                              boolean isAlphaPremultiplied)
```

Forces the raster data to match the state specified in the `isAlphaPremultiplied` variable, assuming the data is currently correctly described by this `ColorModel`. It may multiply or divide the color raster data by alpha, or do nothing if the data is in the correct state. If the data needs to be coerced, this method also returns an instance of `FloatDoubleColorModel` with the `isAlphaPremultiplied` flag set appropriately.

Throws:
java.lang.IllegalArgumentException - if transfer type of `raster` is not the same as that of this `FloatDoubleColorModel`.

Overrides:
coerceData in class java.awt.image.ComponentColorModel

---

isCompatibleRaster

```
public boolean isCompatibleRaster(java.awt.image.Raster raster)
```

Returns `true` if the supplied `Raster`'s `SampleModel` is compatible with this `FloatDoubleColorModel`.

Parameters:
`raster` - a `Raster`to be checked for compatibility.

Overrides:
isCompatibleRaster in class java.awt.image.ComponentColorModel

---

createCompatibleWritableRaster

```
public java.awt.image.WritableRaster createCompatibleWritableRaster(int w,
                                                                    int h)
```

Creates a `WritableRaster` with the specified width and height, that has a data layout (`SampleModel`) compatible with this `ColorModel`. The returned `WritableRaster`'s `SampleModel` will be an instance of `ComponentSampleModel`.

Parameters:
`w` - The width of the `WritableRaster` you want to create.
`h` - The height of the `WritableRaster` you want to create.

Returns:
A `WritableRaster` that is compatible with this `ColorModel`.

Overrides:
createCompatibleWritableRaster in class java.awt.image.ComponentColorModel

See Also:
`WritableRaster`, `SampleModel`

---

createCompatibleSampleModel

```
public java.awt.image.SampleModel createCompatibleSampleModel(int w,
                                                              int h)
```

Creates a `SampleModel` with the specified width and height that has a data layout compatible with this `ColorModel`. The returned `SampleModel` will be an instance of `ComponentSampleModel`.

Parameters:
`w` - The width of the `SampleModel` you want to create.
`h` - The height of the `SampleModel` you want to create.

Returns:
A `SampleModel` that is compatible with this `ColorModel`.

Overrides:
createCompatibleSampleModel in class java.awt.image.ComponentColorModel

See Also:
`SampleModel`, `ComponentSampleModel`

---

isCompatibleSampleModel

`public boolean **isCompatibleSampleModel**(java.awt.image.SampleModel sm)`

Checks whether or not the specified `SampleModel` is compatible with this `ColorModel`. A `SampleModel` is compatible if it is an instance of `ComponentSampleModel`, has the sample number of bands as the total nomber of components (including alpha) in the `ColorSpace` used by this `ColorModel`, and has the same data type (float or double) as this `ColorModel`.

Parameters:

`sm` - The `SampleModel` to test for compatibility.

Returns:

`true` if the `SampleModel` is compatible with this `ColorModel`, `false` if it is not.

Overrides:

isCompatibleSampleModel in class java.awt.image.ComponentColorModel

See Also:

`SampleModel`, `ComponentSampleModel`

---

toString

`public java.lang.String **toString**()`

Returns a String containing the values of all valid fields.

Overrides:

toString in class java.awt.image.ColorModel

javax.media.jai

Class GraphicsJAI

```
java.lang.Object
  |
  +--java.awt.Graphics
        |
        +--java.awt.Graphics2D
              |
              +--javax.media.jai.GraphicsJAI
```

public class **GraphicsJAI**
extends java.awt.Graphics2D

A JAI wrapper for a Graphics2D object derived from a Component. When drawing JAI images to a Component such as a Canvas, a new GraphicsJAI may be constructed to wrap the Graphics2D object provided by that Component. This GraphicsJAI object may provide acceleration for calls to drawRenderedImage(), drawRenderableImage(), and possibly other methods.

If it is possible to use a CanvasJAI object instead of a generic Canvas, or other Canvas subclass, then the Graphics objects obtained from getGraphics() or received as an argument in paint() will automatically be instances of GraphicsJAI.

The portion of the `GraphicsJAI` interface that deals with adding and retrieving new hardware-specific implementations has not been finalized and does not appear in the current API.

See Also:
`CanvasJAI`

Field Detail

g

`java.awt.Graphics2D **g**`

component

`java.awt.Component **component**`

Constructor Detail

GraphicsJAI

```
protected GraphicsJAI(java.awt.Graphics2D g,
                      java.awt.Component component)
```

Constructs a new instance of `GraphicsJAI` that wraps a given instance of `Graphics2D` for drawing to a given `Component`.

Method Detail

createGraphicsJAI

```
public static GraphicsJAI createGraphicsJAI(java.awt.Graphics2D g,
                                            java.awt.Component component)
```

Returns an instance of `GraphicsJAI` suitable for rendering to the given `Component` via the given `Graphics2D` instance.

If one is available, his method will select a hardware-specific implementation, that is specialized for the display device containing the component.

create

```
public java.awt.Graphics create()
```

Creates a new `GraphicsJAI` object that is a copy of this `GraphicsJAI` object.

Overrides:
create in class java.awt.Graphics

See Also:
`Graphics.create()`

getColor

`public java.awt.Color **getColor**()`

See comments in java.awt.Graphics.
Overrides:
getColor in class java.awt.Graphics
See Also:
`Graphics.getColor()`

setColor

`public void **setColor**(java.awt.Color c)`

See comments in java.awt.Graphics.
Overrides:
setColor in class java.awt.Graphics
See Also:
`Graphics.setColor(Color)`

setPaintMode

`public void **setPaintMode**()`

See comments in java.awt.Graphics.
Overrides:
setPaintMode in class java.awt.Graphics
See Also:
`Graphics.setPaintMode()`

setXORMode

`public void **setXORMode**(java.awt.Color c1)`

See comments in java.awt.Graphics.
Overrides:
setXORMode in class java.awt.Graphics
See Also:
`Graphics.setXORMode(Color)`

getFont

`public java.awt.Font **getFont**()`

See comments in java.awt.Graphics.
Overrides:
getFont in class java.awt.Graphics
See Also:
`Graphics.getFont()`

setFont

`public void **setFont**(java.awt.Font font)`

See comments in java.awt.Graphics.
Overrides:
setFont in class java.awt.Graphics
See Also:
`Graphics.setFont(Font)`

getFontMetrics

`public java.awt.FontMetrics **getFontMetrics**(java.awt.Font f)`

See comments in java.awt.Graphics.
Overrides:
getFontMetrics in class java.awt.Graphics

See Also:
    `Graphics.getFontMetrics(Font)`

---

getClipBounds

```
public java.awt.Rectangle getClipBounds()
```

See comments in java.awt.Graphics.
Overrides:
    getClipBounds in class java.awt.Graphics
See Also:
    `Graphics.getClipBounds()`

---

clipRect

```
public void clipRect(int x,
                     int y,
                     int width,
                     int height)
```

See comments in java.awt.Graphics.
Overrides:
    clipRect in class java.awt.Graphics
See Also:
    `Graphics.clipRect(int, int, int, int)`

---

setClip

```
public void setClip(int x,
                    int y,
                    int width,
                    int height)
```

See comments in java.awt.Graphics.
Overrides:
    setClip in class java.awt.Graphics
See Also:
    `Graphics.setClip(int, int, int, int)`

---

getClip

```
public java.awt.Shape getClip()
```

See comments in java.awt.Graphics.
Overrides:
    getClip in class java.awt.Graphics
See Also:
    `Graphics.getClip()`

---

setClip

```
public void setClip(java.awt.Shape clip)
```

See comments in java.awt.Graphics.
Overrides:
    setClip in class java.awt.Graphics
See Also:
    `Graphics.setClip(Shape)`

---

copyArea

```
public void copyArea(int x,
                     int y,
                     int width,
                     int height,
                     int dx,
                     int dy)
```

See comments in java.awt.Graphics.
Overrides:
copyArea in class java.awt.Graphics
See Also:
`Graphics.copyArea(int, int, int, int, int, int)`

---

drawLine

```
public void drawLine(int x1,
                     int y1,
                     int x2,
                     int y2)
```

See comments in java.awt.Graphics.
Overrides:
drawLine in class java.awt.Graphics
See Also:
`Graphics.drawLine(int, int, int, int)`

---

fillRect

```
public void fillRect(int x,
                     int y,
                     int width,
                     int height)
```

See comments in java.awt.Graphics.
Overrides:
fillRect in class java.awt.Graphics
See Also:
`Graphics.fillRect(int, int, int, int)`

---

clearRect

```
public void clearRect(int x,
                      int y,
                      int width,
                      int height)
```

See comments in java.awt.Graphics.
Overrides:
clearRect in class java.awt.Graphics
See Also:
`Graphics.clearRect(int, int, int, int)`

---

drawRoundRect

```
public void drawRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

See comments in java.awt.Graphics.
Overrides:
drawRoundRect in class java.awt.Graphics
See Also:
`Graphics.drawRoundRect(int, int, int, int, int, int)`

---

fillRoundRect

```
public void fillRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

See comments in java.awt.Graphics.
Overrides:
fillRoundRect in class java.awt.Graphics
See Also:
`Graphics.fillRoundRect(int, int, int, int, int, int)`

---

drawOval

```
public void drawOval(int x,
                     int y,
                     int width,
                     int height)
```

See comments in java.awt.Graphics.
Overrides:
drawOval in class java.awt.Graphics
See Also:
`Graphics.drawOval(int, int, int, int)`

---

fillOval

```
public void fillOval(int x,
                     int y,
                     int width,
                     int height)
```

See comments in java.awt.Graphics.
Overrides:
fillOval in class java.awt.Graphics
See Also:
`Graphics.fillOval(int, int, int, int)`

---

drawArc

```
public void drawArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

See comments in java.awt.Graphics.
Overrides:
drawArc in class java.awt.Graphics
See Also:
`Graphics.drawArc(int, int, int, int, int, int)`

---

fillArc

```
public void fillArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

See comments in java.awt.Graphics.
Overrides:
fillArc in class java.awt.Graphics
See Also:
`Graphics.fillArc(int, int, int, int, int, int)`

---

drawPolyline

```
public void drawPolyline(int[] xPoints,
                         int[] yPoints,
                         int nPoints)
```

See comments in java.awt.Graphics.
Overrides:
drawPolyline in class java.awt.Graphics
See Also:
`Graphics.drawPolyline(int[], int[], int)`

---

drawPolygon

```
public void drawPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

See comments in java.awt.Graphics.
Overrides:
drawPolygon in class java.awt.Graphics
See Also:
`Graphics.drawPolygon(int[], int[], int)`

---

fillPolygon

```
public void fillPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

See comments in java.awt.Graphics.
Overrides:
fillPolygon in class java.awt.Graphics
See Also:
`Graphics.fillPolygon(int[], int[], int)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics
See Also:
`Graphics.drawImage(Image, int, int, ImageObserver)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics
See Also:
`Graphics.drawImage(Image, int, int, int, int, ImageObserver)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics

See Also:
`Graphics.drawImage(Image, int, int, Color, ImageObserver)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics
See Also:
`Graphics.drawImage(Image, int, int, int, int, Color, ImageObserver)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics
See Also:
`Graphics.drawImage(Image, int, int, int, int, int, int, int, int, ImageObserver)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

See comments in java.awt.Graphics.
Overrides:
drawImage in class java.awt.Graphics
See Also:
`Graphics.drawImage(Image, int, int, int, int, int, int, int, int, Color, ImageObserver)`

---

dispose

```
public void dispose()
```

See comments in java.awt.Graphics.
Overrides:
dispose in class java.awt.Graphics
See Also:
`Graphics.dispose()`

---

draw

```
public void draw(java.awt.Shape s)
```

See comments in java.awt.Graphics2D.

Overrides:

draw in class java.awt.Graphics2D

See Also:

`Graphics2D.draw(Shape)`

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         java.awt.geom.AffineTransform xform,
                         java.awt.image.ImageObserver obs)
```

See comments in java.awt.Graphics2D.

Overrides:

drawImage in class java.awt.Graphics2D

See Also:

`Graphics2D.drawImage(Image, AffineTransform, ImageObserver)`

---

drawImage

```
public void drawImage(java.awt.image.BufferedImage img,
                      java.awt.image.BufferedImageOp op,
                      int x,
                      int y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawImage in class java.awt.Graphics2D

See Also:

`Graphics2D.drawImage(BufferedImage, BufferedImageOp, int, int)`

---

drawRenderedImage

```
public void drawRenderedImage(java.awt.image.RenderedImage img,
                              java.awt.geom.AffineTransform xform)
```

See comments in java.awt.Graphics2D.

Overrides:

drawRenderedImage in class java.awt.Graphics2D

See Also:

`Graphics2D.drawRenderedImage(RenderedImage, AffineTransform)`

---

drawRenderableImage

```
public void drawRenderableImage(java.awt.image.renderable.RenderableImage img,
                                java.awt.geom.AffineTransform xform)
```

See comments in java.awt.Graphics2D.

Overrides:

drawRenderableImage in class java.awt.Graphics2D

See Also:

`Graphics2D.drawRenderableImage(RenderableImage, AffineTransform)`

---

drawString

```
public void drawString(java.lang.String str,
                       int x,
                       int y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawString in class java.awt.Graphics2D

See Also:

`Graphics2D.drawString(String, int, int)`

drawString

```
public void drawString(java.lang.String s,
                       float x,
                       float y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawString in class java.awt.Graphics2D

See Also:

`Graphics2D.drawString(String, float, float)`

---

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       int x,
                       int y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawString in class java.awt.Graphics2D

See Also:

`Graphics2D.drawString(AttributedCharacterIterator, int, int)`

---

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       float x,
                       float y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawString in class java.awt.Graphics2D

See Also:

`Graphics2D.drawString(AttributedCharacterIterator, float, float)`

---

drawGlyphVector

```
public void drawGlyphVector(java.awt.font.GlyphVector g,
                            float x,
                            float y)
```

See comments in java.awt.Graphics2D.

Overrides:

drawGlyphVector in class java.awt.Graphics2D

See Also:

`Graphics2D.drawGlyphVector(GlyphVector, float, float)`

---

fill

```
public void fill(java.awt.Shape s)
```

See comments in java.awt.Graphics2D.

Overrides:

fill in class java.awt.Graphics2D

See Also:

`Graphics2D.fill(Shape)`

---

hit

```
public boolean hit(java.awt.Rectangle rect,
                   java.awt.Shape s,
                   boolean onStroke)
```

See comments in java.awt.Graphics2D.

Overrides:

hit in class java.awt.Graphics2D

See Also:

`Graphics2D.hit(Rectangle, Shape, boolean)`

getDeviceConfiguration

```
public java.awt.GraphicsConfiguration getDeviceConfiguration()
```

See comments in java.awt.Graphics2D.

Overrides:
getDeviceConfiguration in class java.awt.Graphics2D

See Also:
`Graphics2D.getDeviceConfiguration()`

setComposite

```
public void setComposite(java.awt.Composite comp)
```

See comments in java.awt.Graphics2D.

Overrides:
setComposite in class java.awt.Graphics2D

See Also:
`Graphics2D.setComposite(Composite)`

setPaint

```
public void setPaint(java.awt.Paint paint)
```

See comments in java.awt.Graphics2D.

Overrides:
setPaint in class java.awt.Graphics2D

See Also:
`Graphics2D.setPaint(Paint)`

setStroke

```
public void setStroke(java.awt.Stroke s)
```

See comments in java.awt.Graphics2D.

Overrides:
setStroke in class java.awt.Graphics2D

See Also:
`Graphics2D.setStroke(Stroke)`

setRenderingHint

```
public void setRenderingHint(java.awt.RenderingHints.Key hintKey,
                             java.lang.Object hintValue)
```

See comments in java.awt.Graphics2D.

Overrides:
setRenderingHint in class java.awt.Graphics2D

See Also:
`Graphics2D.setRenderingHint(RenderingHints.Key, Object)`

getRenderingHint

```
public java.lang.Object getRenderingHint(java.awt.RenderingHints.Key hintKey)
```

See comments in java.awt.Graphics2D.

Overrides:
getRenderingHint in class java.awt.Graphics2D

See Also:
`Graphics2D.getRenderingHint(RenderingHints.Key)`

setRenderingHints

```
public void setRenderingHints(java.util.Map hints)
```

See comments in java.awt.Graphics2D.

Overrides:
setRenderingHints in class java.awt.Graphics2D

See Also:
`Graphics2D.setRenderingHints(Map)`

---

addRenderingHints

```
public void addRenderingHints(java.util.Map hints)
```

See comments in java.awt.Graphics2D.
Overrides:
addRenderingHints in class java.awt.Graphics2D
See Also:
`Graphics2D.addRenderingHints(Map)`

---

getRenderingHints

```
public java.awt.RenderingHints getRenderingHints()
```

See comments in java.awt.Graphics2D.
Overrides:
getRenderingHints in class java.awt.Graphics2D
See Also:
`Graphics2D.getRenderingHints()`

---

translate

```
public void translate(int x,
                      int y)
```

See comments in java.awt.Graphics2D.
Overrides:
translate in class java.awt.Graphics2D
See Also:
`Graphics2D.translate(int, int)`

---

translate

```
public void translate(double tx,
                      double ty)
```

See comments in java.awt.Graphics2D.
Overrides:
translate in class java.awt.Graphics2D
See Also:
`Graphics2D.translate(double, double)`

---

rotate

```
public void rotate(double theta)
```

See comments in java.awt.Graphics2D.
Overrides:
rotate in class java.awt.Graphics2D
See Also:
`Graphics2D.rotate(double)`

---

rotate

```
public void rotate(double theta,
                   double x,
                   double y)
```

See comments in java.awt.Graphics2D.
Overrides:
rotate in class java.awt.Graphics2D
See Also:
`Graphics2D.rotate(double, double, double)`

---

scale

```
public void scale(double sx,
                  double sy)
```

See comments in java.awt.Graphics2D.

Overrides:

scale in class java.awt.Graphics2D

See Also:

`Graphics2D.scale(double, double)`

---

shear

```
public void shear(double shx,
                  double shy)
```

See comments in java.awt.Graphics2D.

Overrides:

shear in class java.awt.Graphics2D

See Also:

`Graphics2D.shear(double, double)`

---

transform

```
public void transform(java.awt.geom.AffineTransform Tx)
```

See comments in java.awt.Graphics2D.

Overrides:

transform in class java.awt.Graphics2D

See Also:

`Graphics2D.transform(AffineTransform)`

---

setTransform

```
public void setTransform(java.awt.geom.AffineTransform Tx)
```

See comments in java.awt.Graphics2D.

Overrides:

setTransform in class java.awt.Graphics2D

See Also:

`Graphics2D.setTransform(AffineTransform)`

---

getTransform

```
public java.awt.geom.AffineTransform getTransform()
```

See comments in java.awt.Graphics2D.

Overrides:

getTransform in class java.awt.Graphics2D

See Also:

`Graphics2D.getTransform()`

---

getPaint

```
public java.awt.Paint getPaint()
```

See comments in java.awt.Graphics2D.

Overrides:

getPaint in class java.awt.Graphics2D

See Also:

`Graphics2D.getPaint()`

---

getComposite

```
public java.awt.Composite getComposite()
```

See comments in java.awt.Graphics2D.

Overrides:

getComposite in class java.awt.Graphics2D

See Also:
`Graphics2D.getComposite()`

---

setBackground

`public void **setBackground**(java.awt.Color color)`

See comments in java.awt.Graphics2D.

Overrides:
setBackground in class java.awt.Graphics2D

See Also:
`Graphics2D.setBackground(Color)`

---

getBackground

`public java.awt.Color **getBackground**()`

See comments in java.awt.Graphics2D.

Overrides:
getBackground in class java.awt.Graphics2D

See Also:
`Graphics2D.getBackground()`

---

getStroke

`public java.awt.Stroke **getStroke**()`

See comments in java.awt.Graphics2D.

Overrides:
getStroke in class java.awt.Graphics2D

See Also:
`Graphics2D.getStroke()`

---

clip

`public void **clip**(java.awt.Shape s)`

See comments in java.awt.Graphics2D.

Overrides:
clip in class java.awt.Graphics2D

See Also:
`Graphics2D.clip(Shape)`

---

getFontRenderContext

`public java.awt.font.FontRenderContext **getFontRenderContext**()`

See comments in java.awt.Graphics2D.

Overrides:
getFontRenderContext in class java.awt.Graphics2D

See Also:
`Graphics2D.getFontRenderContext()`

javax.media.jai

Class Histogram

```
java.lang.Object
  |
  +--javax.media.jai.Histogram
```

public class **Histogram**
extends java.lang.Object
implements java.io.Serializable

An object for accumulating histogram information on an image.

A histogram counts the number of image samples whose values lie within a given range of values, or "bin." The source image may be of any data type. Furthermore, the set of pixels counted may be limited by the use of a region of interest (ROI), and by horizontal and vertical subsampling factors. These factors allow the accuracy of the histogram to be traded for speed of computation.

The Histogram class is intended to be used mainly by the "Histogram" operation, which takes care of the details of taking a histogram of an entire (tiled) image.

See Also:
`HistogramDescriptor`

Field Detail

numBands

`private int numBands`

The number of bands in the image which the histogram is taken.

numBins

`private int[] numBins`

The number of bins used for each band of the image.

lowValue

`private double[] lowValue`

The lowest pixel value of the image checked for each band.

highValue

`private double[] highValue`

The highest pixel value of the image checked for each band.

binWidth

`private double[] binWidth`

bins

`private int[][] bins`

The bins for each band, used to hold information about pixel vlaues.

Constructor Detail

Histogram

```
public Histogram(int[] numBins,
                 double[] lowValue,
                 double[] highValue)
```

Constructs a Histogram that may be used to accumulate data within a given range for each band of an image. The legal pixel range and the number of bins may be controlled separately. If binWidth is defined as (highValue - lowValue)/numBins, bin i will count pixel values in the range from `lowValue + i*binWidth` <= x < `lowValue + (i + 1)*binWidth`. Pixels that have values outside the range of lowValue and highValue, that is `(pixel < lowValue && pixel >= highValue)`, are ignored.

Parameters:

`numBins` - The number of bins for each band of the image; numBins.length must be equal to the number of bands of the image which the histogram is taken.
`lowValue` - The lowest pixel value checked for each band.
`highValue` - The highest pixel value checked for each band. Note when counting the pixel values, this highValue is not included based on the above formula.

Throws:

NullPointerException - if numBins is null.
NullPointerException - if lowValue is null.
NullPointerException - if highValue is null.
java.lang.IllegalArgumentException - if either lowValue of highValue does not have the same number of elements as numBins

Method Detail

getNumBands

`public int getNumBands()`

Returns the number of bands of the histogram.

---

getNumBins

`public int[] getNumBins()`

Returns the number of bins of the histogram for all bands.

---

getNumBins

`public int getNumBins(int band)`

Returns the number of bins of the histogram for a specified band.
Throws:
ArrayIndexOutOfBoundsException - if an invalid band is specified.

---

getLowValue

`public double[] getLowValue()`

Returns the lowest value checked for all bands.

---

getLowValue

`public double getLowValue(int band)`

Returns the lowest value checked for a specified band.
Throws:
ArrayIndexOutOfBoundsException - if an invalid band is specified.

---

getHighValue

`public double[] getHighValue()`

Returns the highest value checked for all bands.

---

getHighValue

`public double getHighValue(int band)`

Returns the highest value checked for a specified band.
Throws:
ArrayIndexOutOfBoundsException - if an invalid band is specified.

getBins

```
public int[][] getBins()
```

Returns the bins of the histogram for all bands.

getBins

```
public int[] getBins(int band)
```

Returns the bins of the histogram for a specified band.

Throws:

ArrayIndexOutOfBoundsException - if an invalid band is specified.

getBinSize

```
public int getBinSize(int band,
                      int bin)
```

Returns the number of pixel values found in a given bin for a given band.

Throws:

ArrayIndexOutOfBoundsException - if an invalid band is specified.
ArrayIndexOutOfBoundsException - if an invalid bin is specified.

getBinLowValue

```
public double getBinLowValue(int band,
                             int bin)
```

Returns the lowest pixel value found in a given bin for a given band.

Throws:

ArrayIndexOutOfBoundsException - if an invalid band is specified.

clearHistogram

```
public void clearHistogram()
```

Resets the counts of all bins to zero.

countPixels

```
public void countPixels(java.awt.image.Raster pixels,
                        ROI roi,
                        int xStart,
                        int yStart,
                        int xPeriod,
                        int yPeriod)
```

Adds the pixels of a Raster that lie within a given region of interest (ROI) to the histogram. The set of pixels is further reduced by subsampling factors in the horizontal and vertical directions. The set of pixels to be accumulated may be obtained by intersecting the grid `(xStart + i*xPeriod, yStart + j*yPeriod); i, j >= 0` with the region of interest and the bounding rectangale of the Raster.

Parameters:

`pixels` - A Raster containing pixels to be histogrammed.
`roi` - The region of interest, as a ROI.
`xStart` - The initial X sample coordinate.
`yStart` - The initial Y sample coordinate.
`xPeriod` - The X sampling rate.
`yPeriod` - The Y sampling rate.

Throws:

NullPointerException - if pixels is null.
java.lang.IllegalArgumentException - if the source raster and the histogram object do not match in number of bands.

countPixelsByte

```
private void countPixelsByte(RasterAccessor source,
                             java.awt.Rectangle rect,
                             int xPeriod,
                             int yPeriod)
```

---

countPixelsUShort

```
private void countPixelsUShort(RasterAccessor source,
                               java.awt.Rectangle rect,
                               int xPeriod,
                               int yPeriod)
```

---

countPixelsShort

```
private void countPixelsShort(RasterAccessor source,
                              java.awt.Rectangle rect,
                              int xPeriod,
                              int yPeriod)
```

---

countPixelsInt

```
private void countPixelsInt(RasterAccessor source,
                            java.awt.Rectangle rect,
                            int xPeriod,
                            int yPeriod)
```

---

countPixelsFloat

```
private void countPixelsFloat(RasterAccessor source,
                              java.awt.Rectangle rect,
                              int xPeriod,
                              int yPeriod)
```

---

countPixelsDouble

```
private void countPixelsDouble(RasterAccessor source,
                               java.awt.Rectangle rect,
                               int xPeriod,
                               int yPeriod)
```

---

startPosition

```
private int startPosition(int pos,
                          int start,
                          int Period)
```

javax.media.jai
Interface ImageFunction

public abstract interface **ImageFunction**

ImageFunction is a common interface for vector-valued functions which are to be evaluated at positions in the X-Y coordinate system. At each position the value of such a function may contain one or more elements each of which may be complex.

Method Detail

isComplex

```
public boolean isComplex()
```

Returns whether or not each value’s elements are complex.

getNumElements

```
public int getNumElements()
```

Returns the number of elements per value at each position.

getElements

```
public void getElements(float startX,
                        float startY,
                        float deltaX,
                        float deltaY,
                        int countX,
                        int countY,
                        int element,
                        float[] real,
                        float[] imag)
```

Returns all values of a given element for a specified set of coordinates. An ArrayIndexOutOfBoundsException may be thrown if the length of the supplied array(s) is insufficient.

Parameters:

`startX` - The X coordinate of the upper left location to evaluate.
`startY` - The Y coordinate of the upper left location to evaluate.
`deltaX` - The horizontal increment.
`deltaY` - The vertical increment.
`countX` - The number of points in the horizontal direction.
`countY` - The number of points in the vertical direction.
`real` - A pre-allocated float array of length at least countX*countY in which the real parts of all elements will be returned.
`imag` - A pre-allocated float array of length at least countX*countY in which the imaginary parts of all elements will be returned; may be null for real data, i.e., when `isComplex()` returns false.

Throws:

ArrayIndexOutOfBoundsException - if the length of the supplied array(s) is insufficient.

getElements

```
public void getElements(double startX,
                        double startY,
                        double deltaX,
                        double deltaY,
                        int countX,
                        int countY,
                        int element,
                        double[] real,
                        double[] imag)
```

Returns all values of a given element for a specified set of coordinates. An ArrayIndexOutOfBoundsException may be thrown if the length of the supplied array(s) is insufficient.

Parameters:

`startX` - The X coordinate of the upper left location to evaluate.
`startY` - The Y coordinate of the upper left location to evaluate.
`deltaX` - The horizontal increment.

`deltaY` - The vertical increment.
`countX` - The number of points in the horizontal direction.
`countY` - The number of points in the vertical direction.
`real` - A pre-allocated double array of length at least countX*countY in which the real parts of all elements will be returned.
`imag` - A pre-allocated double array of length at least countX*countY in which the imaginary parts of all elements will be returned; may be null for real data, i.e., when `isComplex()` returns false.

Throws:

ArrayIndexOutOfBoundsException - if the length of the supplied array(s) is insufficient.

javax.media.jai

Interface ImageJAI

All Known Implementing Classes:

CollectionImage, ImageMIPMap, PlanarImage

---

public abstract interface **ImageJAI**
extends PropertySource

An interface implemented by all JAI image classes.

---

javax.media.jai
Class ImageLayout

```
java.lang.Object
  |
  +--javax.media.jai.ImageLayout
```

public class **ImageLayout**
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable

A class describing the desired layout of an OpImage.

The ImageLayout class encapsulates three types of information about an image:

- The image bounds, comprising the min X and Y coordinates, image width, and image height;
- The tile grid layout, comprising the tile grid X and Y offsets, the tile width, and the tile height; and
- The SampleModel and ColorModel of the image.

Each of these parameters may be set individually, or left unset. An unset parameter will cause the corresponding value of a given RenderedImage to be used. For example, the code:

```
 ImageLayout layout;
 RenderedImage im;

 int width = layout.getTileWidth(im);
```

will return the tile width of the ImageLayout if it is set, or the tile width of the image `im` if it is not.

ImageLayout objects are primarily intended to be passed as part of the renderingHints argument of the create() method of RenderedImageFactory. The create() method may remove parameter settings that it cannot deal with, prior to passing the ImageLayout to any OpImage constructors. New OpImage subclasses are not required to accept an ImageLayout parameter, but most will at least need to synthesize one to be passed up the constructor chain.

Methods that modify the state of an ImageLayout return a reference to 'this' following the change. This allows multiple modifications to be made in a single expression. This provides a way of modifying an ImageLayout within a superclass constructor call.

Field Detail

MIN_X_MASK

`public static final int` **`MIN_X_MASK`**

A bitmask to specify the validity of minX.

MIN_Y_MASK

`public static final int` **`MIN_Y_MASK`**

A bitmask to specify the validity of minY.

WIDTH_MASK

`public static final int` **`WIDTH_MASK`**

A bitmask to specify the validity of width.

HEIGHT_MASK

`public static final int` **`HEIGHT_MASK`**

A bitmask to specify the validity of height.

TILE_GRID_X_OFFSET_MASK

`public static final int` **`TILE_GRID_X_OFFSET_MASK`**

A bitmask to specify the validity of tileGridXOffset.

TILE_GRID_Y_OFFSET_MASK

`public static final int` **`TILE_GRID_Y_OFFSET_MASK`**

A bitmask to specify the validity of tileGridYOffset.

TILE_WIDTH_MASK

`public static final int` **`TILE_WIDTH_MASK`**

A bitmask to specify the validity of tileWidth.

TILE_HEIGHT_MASK

`public static final int` **`TILE_HEIGHT_MASK`**

A bitmask to specify the validity of tileHeight.

SAMPLE_MODEL_MASK

`public static final int` **`SAMPLE_MODEL_MASK`**

A bitmask to specify the validity of sampleModel.

COLOR_MODEL_MASK

`public static final int` **`COLOR_MODEL_MASK`**

A bitmask to specify the validity of colorModel.

---

minX

`int` **`minX`**

The image’s minimum X coordinate.

---

minY

`int` **`minY`**

The image’s minimum Y coordinate.

---

width

`int` **`width`**

The image’s width.

---

height

`int` **`height`**

The image’s height.

---

tileGridXOffset

`int` **`tileGridXOffset`**

The X coordinate of tile (0, 0).

---

tileGridYOffset

`int` **`tileGridYOffset`**

The Y coordinate of tile (0, 0).

---

tileWidth

`int tileWidth`

The width of a tile.

---

tileHeight

`int tileHeight`

The height of a tile.

---

sampleModel

`transient java.awt.image.SampleModel sampleModel`

The image’s SampleModel.

---

colorModel

`transient java.awt.image.ColorModel colorModel`

The image’s ColorModel.

---

validMask

`protected int validMask`

The ’or’-ed together valid bitmasks.

Constructor Detail

ImageLayout

`public ImageLayout()`

Constructs an ImageLayout with no parameters set.

ImageLayout

```
public ImageLayout(int minX,
                   int minY,
                   int width,
                   int height,
                   int tileGridXOffset,
                   int tileGridYOffset,
                   int tileWidth,
                   int tileHeight,
                   java.awt.image.SampleModel sampleModel,
                   java.awt.image.ColorModel colorModel)
```

Constructs an ImageLayout with all its parameters set. The sampleModel and colorModel parameters may be set to null, but are nonetheless considered ’set’ in the sense that they will override the corresponding parameter in any RenderedImage.

Parameters:

`minX` - the image’s minimum X coordinate.
`minY` - the image’s minimum Y coordinate.
`width` - the image’s width.
`height` - the image’s height.
`tileGridXOffset` - the X coordinate of tile (0, 0).
`tileGridYOffset` - the Y coordinate of tile (0, 0).
`tileWidth` - the width of a tile.
`tileHeight` - the height of a tile.
`sampleModel` - the image’s SampleModel.
`colorModel` - the image’s ColorModel.

ImageLayout

```
public ImageLayout(int minX,
                   int minY,
                   int width,
                   int height)
```

Constructs an ImageLayout with only the image dimension parameters set.

Parameters:

`minX` - the image’s minimum X coordinate.
`minY` - the image’s minimum Y coordinate.
`width` - the image’s width.
`height` - the image’s height.

ImageLayout

```
public ImageLayout(int tileGridXOffset,
                   int tileGridYOffset,
                   int tileWidth,
                   int tileHeight,
                   java.awt.image.SampleModel sampleModel,
                   java.awt.image.ColorModel colorModel)
```

Constructs an ImageLayout with its tile grid layout, SampleModel, and ColorModel parameters set. The sampleModel and colorModel parameters may be set to null, but are nonetheless considered ’set’ in the sense that they will override the corresponding parameter in any RenderedImage.

Parameters:

`tileGridXOffset` - the X coordinate of tile (0, 0).
`tileGridYOffset` - the Y coordinate of tile (0, 0).
`tileWidth` - the width of a tile.
`tileHeight` - the height of a tile.
`sampleModel` - the image’s SampleModel.
`colorModel` - the image’s ColorModel.

ImageLayout

```
public ImageLayout(java.awt.image.RenderedImage im)
```

Constructs an ImageLayout with all its parameters set to equal those of a given RenderedImage.

Parameters:

`im` - a RenderedImage whose layout will be copied.

Method Detail

getValidMask

```
public int getValidMask()
```

Returns the ’or’-ed together bitmask indicating parameter validity. To determine the validity of a particular parameter, say tile width, test `getValidMask() & ImageLayout.TILE_WIDTH_MASK` against `0`.

To test a single mask value or set of mask values, the convenience method isValid() may be used.

Returns:

an int that is the logical ’or’ of the valid mask values, with a ’1’ bit representing the setting of a value.

---

isValid

```
public final boolean isValid(int mask)
```

Returns true if all the parameters specified by the argument are set.

Parameters:

`mask` - a bitmask.

Returns:

a boolean truth value.

---

setValid

`public ImageLayout **setValid**(int mask)`

Sets selected bits of the valid bitmask. The valid bitmask is set to the logical ’or’ of its prior value and a new value.

Parameters:

`mask` - the new mask value to be ’or’-ed with the prior value.

Returns:

a reference to this ImageLayout following the change.

---

unsetValid

`public ImageLayout **unsetValid**(int mask)`

Clears selected bits of the valid bitmask. The valid bitmask is set to the logical ’and’ of its prior value and the negation of the new mask value. This effectively subtracts from the set of valid parameters.

Parameters:

`mask` - the new mask value to be negated and ’and’-ed with the prior value.

Returns:

a reference to this ImageLayout following the change.

---

unsetImageBounds

`public ImageLayout **unsetImageBounds**()`

Marks the parameters dealing with the image bounds (minX, minY, width, and height) as being invalid.

Returns:

a reference to this ImageLayout following the change.

---

unsetTileLayout

`public ImageLayout **unsetTileLayout**()`

Marks the parameters dealing with the tile layout (tileGridXOffset, tileGridYOffset, tileWidth, and tileHeight) as being invalid.

Returns:

a reference to this ImageLayout following the change.

---

getMinX

`public int **getMinX**(java.awt.image.RenderedImage fallback)`

Returns the value of minX if it is valid, and otherwise returns the value from the supplied RenderedImage. If minX is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of minX.

---

setMinX

`public ImageLayout **setMinX**(int minX)`

Sets minX to the supplied value and marks it as valid.

Parameters:

`minX` - the minimum X coordinate of the image, as an int.

Returns:

a reference to this ImageLayout following the change.

---

getMinY

`public int **getMinY**(java.awt.image.RenderedImage fallback)`

Returns the value of minY if it is valid, and otherwise returns the value from the supplied RenderedImage. If minY is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of minY.

setMinY

`public ImageLayout setMinY(int minY)`

Sets minY to the supplied value and marks it as valid.

Parameters:

`minY` - the minimum Y coordinate of the image, as an int.

Returns:

a reference to this ImageLayout following the change.

getWidth

`public int getWidth(java.awt.image.RenderedImage fallback)`

Returns the value of width if it is valid, and otherwise returns the value from the supplied RenderedImage. If width is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of width.

setWidth

`public ImageLayout setWidth(int width)`

Sets width to the supplied value and marks it as valid.

Parameters:

`width` - the width of the image, as an int.

Returns:

a reference to this ImageLayout following the change.

getHeight

`public int getHeight(java.awt.image.RenderedImage fallback)`

Returns the value of height if it is valid, and otherwise returns the value from the supplied RenderedImage. If height is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of height.

setHeight

`public ImageLayout setHeight(int height)`

Sets height to the supplied value and marks it as valid.

Parameters:

`height` - the height of the image, as an int.

Returns:

a reference to this ImageLayout following the change.

getTileGridXOffset

`public int getTileGridXOffset(java.awt.image.RenderedImage fallback)`

Returns the value of tileGridXOffset if it is valid, and otherwise returns the value from the supplied RenderedImage. If tileGridXOffset is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of tileGridXOffset.

setTileGridXOffset

`public ImageLayout **setTileGridXOffset**(int tileGridXOffset)`

Sets tileGridXOffset to the supplied value and marks it as valid.

Parameters:

`tileGridXOffset` - the X coordinate of tile (0, 0), as an int.

Returns:

a reference to this ImageLayout following the change.

---

getTileGridYOffset

`public int **getTileGridYOffset**(java.awt.image.RenderedImage fallback)`

Returns the value of tileGridYOffset if it is valid, and otherwise returns the value from the supplied RenderedImage. If tileGridYOffset is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of tileGridYOffset.

---

setTileGridYOffset

`public ImageLayout **setTileGridYOffset**(int tileGridYOffset)`

Sets tileGridYOffset to the supplied value and marks it as valid.

Parameters:

`tileGridYOffset` - the Y coordinate of tile (0, 0), as an int.

Returns:

a reference to this ImageLayout following the change.

---

getTileWidth

`public int **getTileWidth**(java.awt.image.RenderedImage fallback)`

Returns the value of tileWidth if it is valid, and otherwise returns the value from the supplied RenderedImage. If tileWidth is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of tileWidth.

---

setTileWidth

`public ImageLayout **setTileWidth**(int tileWidth)`

Sets tileWidth to the supplied value and marks it as valid.

Parameters:

`tileWidth` - the width of a tile, as an int.

Returns:

a reference to this ImageLayout following the change.

---

getTileHeight

`public int **getTileHeight**(java.awt.image.RenderedImage fallback)`

Returns the value of tileHeight if it is valid, and otherwise returns the value from the supplied RenderedImage. If tileHeight is not valid and fallback is null, 0 is returned.

Parameters:

`fallback` - the RenderedImage fallback.

Returns:

the appropriate value of tileHeight.

---

setTileHeight

`public ImageLayout **setTileHeight**(int tileHeight)`

Sets tileHeight to the supplied value and marks it as valid.

Parameters:
: `tileHeight` - the height of a tile, as an int.

Returns:
: a reference to this ImageLayout following the change.

---

getSampleModel

`public java.awt.image.SampleModel` **`getSampleModel`**`(java.awt.image.RenderedImage fallback)`

Returns the value of sampleModel if it is valid, and otherwise returns the value from the supplied RenderedImage. If sampleModel is not valid and fallback is null, null is returned.

Parameters:
: `fallback` - the RenderedImage fallback.

Returns:
: the appropriate value of sampleModel.

---

setSampleModel

`public ImageLayout` **`setSampleModel`**`(java.awt.image.SampleModel sampleModel)`

Sets sampleModel to the supplied value and marks it as valid.

Parameters:
: `sampleModel` - the new SampleModel.

Returns:
: a reference to this ImageLayout following the change.

---

getColorModel

`public java.awt.image.ColorModel` **`getColorModel`**`(java.awt.image.RenderedImage fallback)`

Returns the value of colorModel if it is valid, and otherwise returns the value from the supplied RenderedImage. If colorModel is not valid and fallback is null, null is returned.

Parameters:
: `fallback` - the RenderedImage fallback.

Returns:
: the appropriate value of colorModel.

---

setColorModel

`public ImageLayout` **`setColorModel`**`(java.awt.image.ColorModel colorModel)`

Sets colorModel to the supplied value and marks it as valid.

Parameters:
: `colorModel` - the new ColorModel.

Returns:
: a reference to this ImageLayout following the change.

---

toString

`public java.lang.String` **`toString`**`()`

Returns a String containing the values of all valid fields.

Overrides:
: toString in class java.lang.Object

---

clone

`public java.lang.Object` **`clone`**`()`

Returns a clone of the ImageLayout as an Object.

Overrides:
: clone in class java.lang.Object

---

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the ImageLayout.

Throws:

java.io.IOException -

---

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the ImageLayout.

Throws:

java.io.IOException -

javax.media.jai
Class ImageMIPMap

```
java.lang.Object
  |
  +--javax.media.jai.ImageMIPMap
```

Direct Known Subclasses:
ImagePyramid

---

public class **ImageMIPMap**
extends java.lang.Object
implements ImageJAI

A class implementing the "MIP map" operation on a `RenderedImage`. Given a `RenderedImage`, which represents the image at the highest resolution level, the images at each lower resolution levels may be derived by performing a specific chain of operations to down sample the image at the next higher resolution level repeatedly. The highest resolution level is defined as level 0.

The `downSampler` is a chain of operations that is used to derive the image at the next lower resolution level from the image at the current resolution level. That is, given an image at resolution level `i`, the `downSampler` is used to obtain the image at resolution level `i+1`. The chain may contain one or more operation nodes; however, each node must be a `RenderedOp`. The parameter points to the last node in the chain. The very first node in the chain must be a `RenderedOp` that takes one `RenderedImage` as its source. All other nodes may have multiple sources. When traversing back up the chain, if a node has more than one source, the first source, `source0`, is used to move up the chain. This parameter is saved by reference.

See Also:
`ImagePyramid`

Field Detail

highestImage

`protected java.awt.image.RenderedImage` **`highestImage`**

The image with the highest resolution.

---

currentImage

`protected java.awt.image.RenderedImage` **`currentImage`**

The image at the current resolution level.

---

currentLevel

`protected int` **`currentLevel`**

The current resolution level.

---

downSampler

`protected RenderedOp` **`downSampler`**

The operation chain used to derive the lower resolution images.

Constructor Detail

ImageMIPMap

`protected` **`ImageMIPMap`**`()`

The default constructor.

ImageMIPMap

```
public ImageMIPMap(java.awt.image.RenderedImage image,
                   java.awt.geom.AffineTransform transform,
                   Interpolation interpolation)
```

Constructor. The down sampler is an "affine" operation that uses the supplied `AffineTransform` and `Interpolation` objects. All input parameters are saved by reference.

Parameters:

`image` - The image with the highest resolution.
`transform` - An affine matrix used with an "affine" operation to derive the lower resolution images.
`interpolation` - The interpolation method for the "affine" operation. It may be `null`, in which case the default "nearest neighbor" interpolation method is used.

Throws:

java.lang.IllegalArgumentException - if `image` is `null`.
java.lang.IllegalArgumentException - if `transform` is `null`.

ImageMIPMap

```
public ImageMIPMap(java.awt.image.RenderedImage image,
                   RenderedOp downSampler)
```

Constructor. The `downSampler` points to the last operation node in the `RenderedOp` chain. The very first operation in the chain must not have any source images specified; that is, its number of sources must be 0. All input parameters are saved by reference.

Parameters:

`image` - The image with the highest resolution.
`downSampler` - The operation chain used to derive the lower resolution images. No validation is done on the first operation in the chain.

Throws:

NullPointerException - if `image` is `null`.
NullPointerException - if `downSampler` is `null`.

ImageMIPMap

```
public ImageMIPMap(RenderedOp downSampler)
```

Constructs a new `ImageMIPMap` from a `RenderedOp` chain. The `downSampler` points to the last operation node in the `RenderedOp` chain. The source image is determined by traversing up the chain; starting at the bottom node, given by the `downSample` parameter, we move to the first source of the node and repeat until we find either a sourceless `RenderedOp` or any other type of `RenderedImage`. The `downSampler` parameter is saved by reference and should not be modified during the lifetime of any `ImageMIPMap` referring to it.

Parameters:

`downSampler` - The operation chain used to derive the lower resolution images. The source of the first node in this chain is taken as the image with the highest resolution.

Throws:

NullPointerException - if `downSampler` is `null`.
java.lang.IllegalArgumentException - if `downSampler` has no sources.
java.lang.IllegalArgumentException - if an object other than a `RenderedImage` is found in the `downSampler` chain.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns an array of `String`s recognized as names by this property source. If no property names match, `null` will be returned.

The default implementation returns `null`, i.e., no property names are recognized.

Returns:

An array of `String`s giving the valid property names.

---

getPropertyNames

```
public java.lang.String[] getPropertyNames(java.lang.String prefix)
```

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names are recognized, or no property names match, `null` will be returned. The comparison is done in a case-independent manner.

Returns:
An array of `String`s giving the valid property names.
Throws:
NullPointerException - if `prefix` is `null`.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Returns the specified property. The default implementation returns `java.awt.Image.UndefinedProperty`.
Parameters:
`name` - The name of the property.
Returns:
The value of the property, as an Object.

---

getCurrentLevel

```
public int getCurrentLevel()
```

Returns the current resolution level. The highest resolution level is defined as level 0.

---

getCurrentImage

```
public java.awt.image.RenderedImage getCurrentImage()
```

Returns the image at the current resolution level.

---

getImage

```
public java.awt.image.RenderedImage getImage(int level)
```

Returns the image at the specified resolution level. The requested level must be greater than or equal to 0 or `null` will be returned.

---

getDownImage

```
public java.awt.image.RenderedImage getDownImage()
```

Returns the image at the next lower resolution level, obtained by applying the `downSampler` on the image at the current resolution level.

---

duplicate

```
protected RenderedOp duplicate(RenderedOp op,
                               java.util.Vector images)
```

Duplicates a `RenderedOp` chain. Each node in the chain must be a `RenderedOp`. The `op` parameter points to the last `RenderedOp` in the chain. The very first op in the chain must have no sources and its source will be set to the supplied image vector. When traversing up the chain, if any node has more than one source, the first source will be used. The first source of each node is duplicated; all other sources are copied by reference.
Throws:
NullPointerException - if `op` is `null`.
NullPointerException - if `images` is `null`.

---

duplicate

```
protected RenderedOp duplicate(RenderedOp op,
                               java.awt.image.RenderedImage image)
```

Duplicates a `RenderedOp` chain. Each node in the chain must be a `RenderedOp`. The `op` parameter points to the last `RenderedOp` in the chain. The very first op in the chain must have no sources but should take 1 `RenderedImage` as its source and this source will be set to the supplied image. When traversing up the chain, if any node has more than one source, the first source will be used. The first source of each node is duplicated; all other sources are copied by reference.
Throws:
NullPointerException - if `op` is `null`.
NullPointerException - if `image` is `null`.

---

duplicate

```
protected RenderedOp duplicate(RenderedOp op,
                               java.awt.image.RenderedImage image0,
                               java.awt.image.RenderedImage image1)
```

Duplicates a `RenderedOp` chain. Each node in the chain must be a `RenderedOp`. The `op` parameter points to the last `RenderedOp` in the chain. The very first op in the chain must have no sources but should take 2 `RenderedImage`s as its sources and these sources will be set to the supplied images. When traversing up the chain, if any node has more than one source, the first source will be used. The first source of each node is duplicated; all other sources are copied by reference.

Throws:

NullPointerException - if `op` is `null`.
NullPointerException - if `image0` is `null`.
NullPointerException - if `image1` is `null`.

---

getAsRenderable

```
public java.awt.image.renderable.RenderableImage getAsRenderable(int numImages,
                                                                 float minX,
                                                                 float minY,
                                                                 float height)
```

Returns the current image as a `RenderableImage`. This method returns a `MultiResolutionRenderableImage`. The `numImages` parameter indicates the number of `RenderedImage`s used to construct the `MultiResolutionRenderableImage`. Starting with the current image, the images are obtained by finding the necessary numbers of lower resolution images using the `downSampler`. The current level and current image will not be changed.

The `numImages` should be greater than or equal to 1. If a value of less than 1 is specified, this method uses 1 image, which is the current image.

Throws:

java.lang.IllegalArgumentException - if `height` is less than 0.

See Also:

`MultiResolutionRenderableImage`

---

getAsRenderable

```
public java.awt.image.renderable.RenderableImage getAsRenderable()
```

Returns the current image as a `RenderableImage`. This method returns a `MultiResolutionRenderableImage` with the current image as the only source image, minX and minY set to 0.0, and height set to 1.0.

See Also:

`MultiResolutionRenderableImage`

javax.media.jai
Class ImagePyramid

```
java.lang.Object
  |
  +--javax.media.jai.ImageMIPMap
        |
        +--javax.media.jai.ImagePyramid
```

public class **ImagePyramid**
extends ImageMIPMap

A class implementing the "Pyramid" operation on a `RenderedImage`. Given a `RenderedImage` which represents the image at the highest resolution level, the images at lower resolution levels may be derived by performing a specific chain of operations to down sample the image at the higher resolution level repeatedly. Similarly, once an image at a lower resolution level is obtained, the images at higher resolution levels may be retrieved by performing a specific chain of operations to up sample the image at the lower resolution level repeatedly.

When an image is down sampled, the image at the higher resolution level is lost. However, the different image between the original image and the image obtained by up sampling the down sampled result image is saved. This different image, combined with the up sampling operations is used to retrieve the image at a higher resolution level from the image at a lower resolution level.

This is a bi-directional operation. A user may request an image at any resolution level greater than or equal to the highest resolution level, which is defined as level 0.

The `downSampler` is a chain of operations that is used to derive the image at the next lower resolution level from the image at the current resolution level. That is, given an image at resolution level `i`, `downSampler` is used to obtain the image at resolution level `i+1`. The chain may contain one or more operation nodes; however, each node must be a `RenderedOp`. The parameter points to the last node in the chain. The very first node in the chain must be a `RenderedOp` that takes one `RenderedImage` as its source. All other nodes may have multiple sources. When traversing back up the chain, if a node has more than one source, the first source, `source0`, is used to move up the chain. This parameter is saved by reference.

The `upSampler` is a chain of operations that is used to derive the image at the next higher resolution level from the image at the current resolution level. That is, given an image at resolution level `i`, `upSampler` is used to obtain the image at resolution level `i-1`. The requirement for this parameter is similar to that of the `downSampler` parameter.

The `differencer` is a chain of operations that is used to find the difference between an image at a particular resolution level and the image obtained by first down sampling that image then up sampling the result image of the down sampling operations. The chain may contain one or more operation nodes; however, each node must be a `RenderedOp`. The parameter points to the last node in the chain. The very first node in the chain must be a `RenderedOp` that takes two `RenderedImage`s as its sources. When traversing back up the chain, if a node has more than one source, the first source, `source0`, is used to move up the chain. This parameter is saved by reference.

The `combiner` is a chain of operations that is used to combine the result image of the up sampling operations and the different image saved to retrieve an image at a higher resolution level. The requirement for this parameter is similar to that of the `differencer` parameter.

See Also:
`ImageMIPMap`

Field Detail

upSampler

`protected RenderedOp` **`upSampler`**

The operation chain used to derive the higher resolution images.

differencer

`protected RenderedOp` **`differencer`**

The operation chain used to differ two images.

combiner

```
protected RenderedOp combiner
```

The operation chain used to combine two images.

---

diffImages

```
private java.util.Vector diffImages
```

The saved different images.

Constructor Detail

ImagePyramid

```
protected ImagePyramid()
```

The default constructor.

ImagePyramid

```
public ImagePyramid(java.awt.image.RenderedImage image,
                    RenderedOp downSampler,
                    RenderedOp upSampler,
                    RenderedOp differencer,
                    RenderedOp combiner)
```

Constructor. The `RenderedOp` parameters point to the last operation node in each chain. The first operation in each chain must not have any source images specified; that is, its number of sources must be 0. All input parameters are saved by reference.

Parameters:

`image` - The image with the highest resolution.
`downSampler` - The operation chain used to derive the lower resolution images.
`upSampler` - The operation chain used to derive the higher resolution images.
`differencer` - The operation chain used to differ two images.
`combiner` - The operation chain used to combine two images.

Throws:

NullPointerException - if `image` is `null`.
NullPointerException - if `downSampler` is `null`.
NullPointerException - if `upSampler` is `null`.
NullPointerException - if `differencer` is `null`.
NullPointerException - if `combiner` is `null`.

ImagePyramid

```
public ImagePyramid(RenderedOp downSampler,
                    RenderedOp upSampler,
                    RenderedOp differencer,
                    RenderedOp combiner)
```

Constructor. The `RenderedOp` parameters point to the last operation node in each chain. The first operation in the `downSampler` chain must have the image with the highest resolution as its source. The first operation in all other chains must not have any source images specified; that is, its number of sources must be 0. All input parameters are saved by reference.

Parameters:

`downSampler` - The operation chain used to derive the lower resolution images.
`upSampler` - The operation chain used to derive the higher resolution images.
`differencer` - The operation chain used to differ two images.
`combiner` - The operation chain used to combine two images.

Throws:

NullPointerException - if `downSampler` is `null`.
NullPointerException - if `upSampler` is `null`.
NullPointerException - if `differencer` is `null`.
NullPointerException - if `combiner` is `null`.
java.lang.IllegalArgumentException - if `downSampler` has no sources.
java.lang.IllegalArgumentException - if an object other than a `RenderedImage` is found in the `downSampler` chain.

Method Detail

getImage

`public java.awt.image.RenderedImage **getImage**(int level)`

Returns the image at the specified resolution level. The requested level must be greater than or equal to 0 or `null` will be returned. The image is obtained by either down sampling or up sampling the current image.

Overrides:

getImage in class ImageMIPMap

---

getDownImage

`public java.awt.image.RenderedImage **getDownImage**()`

Returns the image at the next lower resolution level, obtained by applying the `downSampler` on the image at the current resolution level.

Overrides:

getDownImage in class ImageMIPMap

---

getUpImage

`public java.awt.image.RenderedImage **getUpImage**()`

Returns the image at the previous higher resolution level, If the current image is already at level 0, then the current image will be returned without further up sampling.

The image is obtained by first up sampling the current image, then combine the result image with the previously saved different image using the `combiner` op chain.

---

getDiffImage

`public java.awt.image.RenderedImage **getDiffImage**()`

Returns the difference image between the current image and the image obtained by first down sampling the current image then up sampling the result image of down sampling. This is done using the `differencer` op chain. The current level and current image will not be changed.

javax.media.jai

Class ImageSequence

```
java.lang.Object
  |
  +--javax.media.jai.CollectionImage
        |
        +--javax.media.jai.ImageSequence
```

---

public class **ImageSequence**
extends CollectionImage

A class representing a sequence of images, each associated with a time stamp and a camera position. The images are of the type `javax.media.jai.PlanarImage`; the time stamps are of the type `float`; the camera positions are of the type `java.lang.Object`. The tuple (image, time stamp, camera position) is represented by class `javax.media.jai.SequentialImage`.

This class can be used to represent video or time-lapse photography.

See Also:
`PlanarImage`, `SequentialImage`

Constructor Detail

ImageSequence

`protected` **`ImageSequence`**`()`

The default constrctor.

ImageSequence

`public` **`ImageSequence`**`(java.util.Collection images)`

Constructs a class that represents a sequence of images.

Parameters:
`images` - A collection of `SequentialImage`.

Throws:
NullPointerException - if `images` is `null`.

Method Detail

getImage

`public PlanarImage` **`getImage`**`(float ts)`

Returns the image associated with the specified time stamp, or `null` if no match is found.

---

getImage

`public PlanarImage` **`getImage`**`(java.lang.Object cp)`

Returns the image associated with the specified camera position, or `null` if `cp` is `null` or if no match is found.

---

getTimeStamp

`public float` **`getTimeStamp`**`(PlanarImage pi)`

Returns the time stamp associated with the specified image, or `-Float.MAX_VALUE` if `pi` is `null` or if no match is found.

---

getCameraPosition

`public java.lang.Object` **`getCameraPosition`**`(PlanarImage pi)`

Returns the camera position associated with the specified image, or `null` if `pi` is `null` or if no match is found.

---

add

`public boolean **add**(java.lang.Object o)`

Adds a `SequentialImage` to this collection. If the specified image is `null`, it is not added to the collection.

Returns:

`true` if and only if the `SequentialImage` is added to the collection.

Overrides:

add in class CollectionImage

---

remove

`public boolean **remove**(PlanarImage pi)`

Removes the `SequentialImage` that contains the specified image from this collection.

Returns:

`true` if and only if a `SequentialImage` with the specified image is removed from the collection.

---

remove

`public boolean **remove**(float ts)`

Removes the `SequentialImage` that contains the specified time stamp from this collection.

Returns:

`true` if and only if a `SequentialImage` with the specified time stamp is removed from the collection.

---

remove

`public boolean **remove**(java.lang.Object cp)`

Removes the `SequentialImage` that contains the specified camera position from this collection.

Returns:

`true` if and only if a `SequentialImage` with the specified camera position is removed from the collection.

Overrides:

remove in class CollectionImage

javax.media.jai
Class ImageStack

```
java.lang.Object
  |
  +--javax.media.jai.CollectionImage
        |
        +--javax.media.jai.ImageStack
```

public abstract class **ImageStack**
extends CollectionImage

A class representing a stack of images, each associated with a spatial orientation defined in a common coordinate system. The images are of the type `javax.media.jai.PlanarImage`; the coordinates are of the type `java.lang.Object`. The tuple (image, coordinate) is represented by class `javax.media.jai.CoordinateImage`.

This class can be used to represent medical or geophysical images.

See Also:
 `PlanarImage`

Constructor Detail

ImageStack

`protected` **`ImageStack`**`()`

The default constructor.

ImageStack

`public` **`ImageStack`**`(java.util.Collection images)`

Constructor.
Parameters:
 `images` - A collection of `CoordinateImage`.
Throws:
 NullPointerException - if `images` is `null`.

Method Detail

getImage

`public PlanarImage` **`getImage`**`(java.lang.Object c)`

Returns the image associated with the specified coordinate, or `null` if `c` is `null` or if no match is found.

getCoordinate

`public java.lang.Object` **`getCoordinate`**`(PlanarImage pi)`

Returns the coordinate associated with the specified image, or `null` if `pi` is `null` or if no match is found.

add

`public boolean` **`add`**`(java.lang.Object o)`

Adds a `CoordinateImage` to this collection. If the specified image is `null`, it is not added to the collection.
Returns:
 true if and only if the `CoordinateImage` is added to the collection.
Overrides:
 add in class CollectionImage

remove

`public boolean **remove**(PlanarImage pi)`

Removes the `CoordinateImage` that contains the specified image from this collection.

Returns:

true if and only if a `CoordinateImage` containing the specified image is removed from the collection.

---

remove

`public boolean **remove**(java.lang.Object c)`

Removes the `CoordinateImage` that contains the specified coordinate from this collection.

Returns:

true if and only if a `CoordinateImage` containing the specified coordinate is removed from the collection.

Overrides:

remove in class CollectionImage

javax.media.jai

Class IntegerSequence

```
java.lang.Object
  |
  +--javax.media.jai.IntegerSequence
```

public class **IntegerSequence**
extends java.lang.Object

A growable sorted integer set. Adding an integer to the sequence results in it being placed into the sequence in sorted order. Adding an integer that is already part of the sequence has no effect.

This structure is used by various subclasses of `OpImage` to keep track of horizontal and vertical source splits. Each instance of IntegerSequence provides an internal enumeration by means of which the elements of the sequence may be accessed in order. The enumerator is initialized by the `startEnumeration` method, and the `hasMoreElements` and `nextElement` methods allow looping through the elements. Only one enumeration at a time is supported. Calling `insert()` from multiple threads is not supported.

Field Detail

min

`private int min`

Lower bound of the valid integer range.

max

`private int max`

Upper bound of the valid integer range.

DEFAULT_CAPACITY

`private static final int DEFAULT_CAPACITY`

The default initial capacity of iArray.

iArray

`private int[] iArray`

The array storing the unsorted integer values.

capacity

`private int capacity`

The capacity of iArray.

numElts

`private int numElts`

The number of (non-unique) elements actually stored in iArray.

isSorted

`private boolean isSorted`

True if iArray has been sorted and purged of duplicates.

currentIndex

`private int currentIndex`

The current element of the iteration.

Constructor Detail

IntegerSequence

```
public IntegerSequence(int min,
                       int max)
```

Constructs a sequence bounded by an inclusive range of values.

IntegerSequence

`public IntegerSequence()`

Constructs a sequence that may contain any integer value.

Method Detail

insert

`public void insert(int element)`

Inserts an integer into the sequence. If the value falls out of the desired range, it will be silently rejected. Inserting an element that is already a member of the sequence has no effect.

Parameters:

`element` - The `int` to be inserted.

---

startEnumeration

`public void startEnumeration()`

Resets the iterator to the beginning of the sequence.

---

hasMoreElements

`public boolean hasMoreElements()`

Returns true if more elements are available to be iterated over.

---

nextElement

`public int nextElement()`

Returns the next element of the iteration in ascending order. If the end of the array has been reached, a `java.util.NoSuchElementException will be thrown.`

Throws:

`java.util.NoSuchElementException - if the end of the array has been reached.`

---

getNumElements

`public int getNumElements()`

`Returns the number of elements contained within this IntegerSequence.`

---

toString

`public java.lang.String toString()`

`Returns a String representation of the sequence for debugging.`

Overrides:

`toString in class java.lang.Object`

javax.media.jai
Class Interpolation

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
```

Direct Known Subclasses:
InterpolationBilinear, InterpolationNearest, InterpolationTable

---

public abstract class **Interpolation**
extends java.lang.Object
implements java.io.Serializable

An object encapsulating a particular algorithm for image interpolation (resampling). An Interpolation captures the notion of performing sampling on a regular grid of pixels using a local neighborhood. It is intended to be used by operations that resample their sources, including affine mapping and warping.

Resampling is the action of computing a pixel value at a possibly non-integral position of an image. The image defines pixel values at integer lattice points, and it is up to the resampler to produce a reasonable value for positions not falling on the lattice. A number of techniques are used in practice, the most common being nearest-neighbor, which simply takes the value of the closest lattice point; bilinear, which interpolates linearly between the four closest lattice points; and bicubic, which applies a piecewise polynomial function to a 4x4 neighborhood of nearby points. The area over which a resampling function needs to be computed is referred to as its support; thus the standard resampling functions have supports of 1, 4, and 16 pixels respectively. Mathematically, the ideal resampling function for a band-limited image (one containing no energy above a given frequency) is the sinc function, equal to sin(x)/x. This has practical limitations, in particular its infinite support, which lead to the use of the standard approximations described above.

Other interpolation functions may be required to solve problems other than the resampling of band-limited image data. When shrinking an image, it is common to use a function that combines area averaging with resampling in order to remove undesirable high frequencies as part of the interpolation process. Other application areas may use interpolating functions that operate under other assumptions about image data, such as taking the maximum value of a 2x2 neighborhood. The interpolation class provides a framework in which a variety of interpolation schemes may be expressed.

Many interpolations are separable, that is, they may be equivalently rewritten as a horizontal interpolation followed by a vertical one (or vice versa). In practice, some precision may be lost by the rounding and truncation that takes place between the passes. The Interpolation class assumes separability and implements all vertical interpolation methods in terms of corresponding horizontal methods, and defines isSeparable() to return true. A subclass may override these methods to provide distinct implementations of horizontal and vertical interpolation. Some subclasses may implement the two-dimensional interpolation methods directly, yielding more precise results, while others may implement these using a two-pass approach.

A minimal Interpolation subclass must call the Interpolation constructor (super()) and then set at least the following fields.

```
leftPadding
rightPadding
topPadding
bottomPadding
width
height
subsampleBitsH
subsampleBitsV
```

It must also implement at least the following methods.

```
int interpolateH(int[] samples, int xfrac)
float interpolateH(float[] samples, float xfrac)
double interpolateH(double[] samples, float xfrac)
```

All other methods are defined in terms of these methods for ease of implementation of new Interpolation subclasses.

Since interpolation is generally performed for every pixel of a destination image, efficiency is important. In particular, passing source samples by means of arrays is likely to be unacceptably slow. Accordingly, methods are provided for the common cases of 2x1, 1x2, 4x1, 1x4, 2x2, and 4x4 input grids. These methods are defined in the superclass to package their arguments into arrays and forward the call to the array versions, in order to simplify implementation. They should be called only on Interpolation objects with the correct width and height. In other words, an implementor of an InterpolationSubclass may implement "interpolateH(int s0, int s1, int xfrac)" assuming that the interpolation width is in fact equal to 2, and does not need to enforce this constraint.

The fractional position of interpolation (xfrac, yfrac) is always between 0.0 and 1.0 (not including 1.0). For integral image data, the fraction is represented as a scaled integer between 0 and $2^n - 1$, where n is a small integer. The value of n in the horizontal and vertical directions may be obtained by calling getSubsampleBitsH() and getSubsampleBitsV(). In general, code that makes use of an externally-provided Interpolation object must query that object to determine its desired positional precision.

For float and double images, a float between 0.0F and 1.0F (not including 1.0F) is used as a positional specifier in the interest of greater accuracy.

It is important to understand that the subsampleBits precision is used only to indicate the scaling implicit in the fractional locations and then only with integral image data types. An implementation is not required to actually quantize its interpolation coefficients to match the specified subsampling precision.

The diagrams below illustrate the pixels involved in one-dimensional interpolation. Point s0 is the interpolation kernel key position.

```
          Horizontal              Vertical

    s_    s0 .  s1    s2            s_
             ^
            xfrac                   s0
                                     .< yfrac
                                    s1

                                    s2
```

The diagram below illustrates the pixels involved in two-dimensional interpolation. Point s00 is the interpolation kernel key position.

```
               s__    s_0    s_1    s_2



               s0_    s00    s01    s02

                          .             < yfrac

               s1_    s10    s11    s12



               s2_    s20    s21    s22
                          ^
                         xfrac
```

The subclasses of Interpolation include InterpolationNearest, InterpolationBilinear, InterpolationBicubic, and InterpolationBicubic2 (a variant defined by a different polynomial function). These subclasses are marked 'final,' so users may identify them by name (using 'instanceof') and write specialized code for them. This may also allow inlining to occur on some virtual machines. These classes do provide correct, if less than optimal code for performing their interpolations, so it is possible to use any Interpolation object in a generic manner. The Sun-provided InterpolationBilinear and InterpolationBicubic classes provide a more optimal implementation while using the same semantics.

The InterpolationTable class is a subclass of Interpolation that divides the set of subsample positions into a fixed number of "bins" and stores a kernel for each bin. InterpolationBicubic and InterpolationBicubic2 are implemented in terms of InterpolationTable since a direct implementation is very expensive.

See Also:

`InterpolationNearest`, `InterpolationBilinear`, `InterpolationBicubic`, `InterpolationBicubic2`, `InterpolationTable`

Field Detail

INTERP_NEAREST

`public static final int` **`INTERP_NEAREST`**

A constant specifying interpolation by the InterpolationNearest class.

INTERP_BILINEAR

`public static final int` **`INTERP_BILINEAR`**

A constant specifying interpolation by the InterpolationBilinear class.

INTERP_BICUBIC

`public static final int` **`INTERP_BICUBIC`**

A constant specifying interpolation by the InterpolationBicubic class.

INTERP_BICUBIC_2

`public static final int` **`INTERP_BICUBIC_2`**

A constant specifying interpolation by the InterpolationBicubic2 class.

---

nearestInstance

`private static final Interpolation` **`nearestInstance`**

---

bilinearInstance

`private static final Interpolation` **`bilinearInstance`**

---

bicubicInstance

`private static final Interpolation` **`bicubicInstance`**

---

bicubic2Instance

`private static final Interpolation` **`bicubic2Instance`**

---

leftPadding

`protected int` **`leftPadding`**

The number of pixels lying to the left of the interpolation kernel key position.

---

rightPadding

`protected int` **`rightPadding`**

The number of pixels lying to the right of the interpolation kernel key position.

---

topPadding

`protected int` **`topPadding`**

The number of pixels lying above the interpolation kernel key position.

---

bottomPadding

`protected int` **`bottomPadding`**

The number of pixels lying below the interpolation kernel key position.

---

subsampleBitsH

`protected int` **`subsampleBitsH`**

The numbers of bits used for the horizontal subsample position. This value determines how integer fractional positons are to be interpreted.

---

subsampleBitsV

`protected int` **`subsampleBitsV`**

The numbers of bits used for the vertical subsample position. This value determines how integer fractional positons are to be interpreted.

width

```
protected int width
```

The width of the interpolation kernel in pixels.

height

```
protected int height
```

The height of the interpolation kernel in pixels.

Constructor Detail

Interpolation

```
public Interpolation()
```

Construct Interpolation object with no fields set. This constructor should only be invoked by subclasses which will subsequently set all fields themselves.

Interpolation

```
public Interpolation(int width,
                     int height,
                     int leftPadding,
                     int rightPadding,
                     int topPadding,
                     int bottomPadding,
                     int subsampleBitsH,
                     int subsampleBitsV)
```

Construct interpolation object with all parameters set. Subclasses must supply all parameters.

Method Detail

getInstance

```
public static Interpolation getInstance(int type)
```

Creates an interpolation of one of the standard types. This is intended strictly as a convenience method.

Parameters:

`type` - one of: INTERP_NEAREST, INTERP_BILINEAR, INTERP_BICUBIC, or INTERP_BICUBIC_2

Returns:

an appropriate Interpolation object.

Throws:

java.lang.IllegalArgumentException - if an unrecognized type is supplied.

getLeftPadding

```
public int getLeftPadding()
```

Returns the number of samples required to the left of the center.

getRightPadding

```
public int getRightPadding()
```

Returns the number of samples required to the right of the center.

getTopPadding

```
public int getTopPadding()
```

Returns the number of samples required above the center.

getBottomPadding

```
public int getBottomPadding()
```

Returns the number of samples required below the center.

---

getWidth

```
public int getWidth()
```

Returns the number of samples required for horizontal resampling.

---

getHeight

```
public int getHeight()
```

Returns the number of samples required for vertical resampling.

---

isSeparable

```
public boolean isSeparable()
```

Returns true if the interpolation can be performed in a separable manner, that is, by performing a separate pass in each dimension. It is the caller’s responsibility to deal with issues of precision. By default, true is returned.

---

getSubsampleBitsH

```
public int getSubsampleBitsH()
```

Returns the number of bits used to index subsample positions in the horizontal direction. All integral ’xfrac’ parameters should range between 0 and $2^{(getSubsampleBitsH())}$ - 1.

In general, the caller is responsible for determining the number of subsample bits of any Interpolation object it receives and setting up its position variables accordingly. Some Interpolation objects allow the number of bits to be set at construction time.

---

getSubsampleBitsV

```
public int getSubsampleBitsV()
```

Returns the number of bits used to index subsample positions in the vertical direction. All integral ’yfrac’ parameters should range between 0 and $2^{(getSubsampleBitsV())}$ - 1.

---

interpolateH

```
public abstract int interpolateH(int[] samples,
                                 int xfrac)
```

Performs horizontal interpolation on a 1-dimensional array of integral samples.

An implementation is not required to actually quantize its interpolation coefficients to match the specified subsampling precision. However, the supplied value of xfrac (or yfrac) must match the precision of its corresponding subsampleBits. For example, with a subsampleBitsH value of 8, xfrac must lie between 0 and 255.

Parameters:

`samples` - an array of ints.

`xfrac` - the subsample position, multiplied by $2^{(subsampleBitsH)}$.

Returns:

the interpolated value as an int.

---

interpolateV

```
public int interpolateV(int[] samples,
                        int yfrac)
```

Performs vertical interpolation on a 1-dimensional array of integral samples.

By default, vertical interpolation is defined to be the same as horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`samples` - an array of ints.

`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.

Returns:
the interpolated value as an int.
See Also:
`interpolateH(int[], int)`

---

interpolate

```
public int interpolate(int[][] samples,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 2-dimensional array of integral samples. By default, this is implemented using a two-pass approach.

Parameters:
`samples` - a two-dimensional array of ints.
`xfrac` - the X subsample position, multiplied by $2^{(subsampleBitsH)}$.
`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.
Returns:
the interpolated value as an int.
See Also:
`interpolateH(int[], int)`

---

interpolateH

```
public int interpolateH(int s0,
                        int s1,
                        int xfrac)
```

Performs horizontal interpolation on a pair of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 2 and leftPadding == 0.

Parameters:
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the subsample position, multiplied by $2^{(subsampleBitsH)}$.
Returns:
the interpolated value as an int.
See Also:
`interpolateH(int[], int)`

---

interpolateH

```
public int interpolateH(int s_,
                        int s0,
                        int s1,
                        int s2,
                        int xfrac)
```

Performs horizontal interpolation on a quadruple of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 4 and leftPadding == 1.

Parameters:
`s_` - the sample to the left of the central sample.
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`s2` - the sample to the right of s1.
`xfrac` - the subsample position, multiplied by $2^{(subsampleBitsH)}$.
Returns:
the interpolated value as an int.
See Also:
`interpolateH(int[], int)`

---

interpolateV

```
public int interpolateV(int s0,
                        int s1,
                        int yfrac)
```

Performs vertical interpolation on a pair of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 2 and topPadding == 0.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.

Returns:

the interpolated value as an int.

See Also:

`interpolateH(int[], int)`

---

interpolateV

```
public int interpolateV(int s_,
                        int s0,
                        int s1,
                        int s2,
                        int yfrac)
```

Performs vertical interpolation on a quadruple of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 4 and topPadding == 1.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.

Returns:

the interpolated value as an int.

See Also:

`interpolateH(int[], int)`

---

interpolate

```
public int interpolate(int s00,
                       int s01,
                       int s10,
                       int s11,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 2x2 grid of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == height == 2 and leftPadding == topPadding == 0.

Parameters:

`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, multiplied by $2^{(subsampleBitsH)}$.
`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.

Returns:

the interpolated value as an int.

See Also:

`interpolateH(int[], int)`

interpolate

```
public int interpolate(int s__,
                       int s_0,
                       int s_1,
                       int s_2,
                       int s0_,
                       int s00,
                       int s01,
                       int s02,
                       int s1_,
                       int s10,
                       int s11,
                       int s12,
                       int s2_,
                       int s20,
                       int s21,
                       int s22,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 4x4 grid of integral samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == height == 4 and leftPadding == topPadding == 1.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, multiplied by $2^{(subsampleBitsH)}$.
`yfrac` - the Y subsample position, multiplied by $2^{(subsampleBitsV)}$.

Returns:

the interpolated value as an int.

See Also:

`interpolateH(int[], int)`

interpolateH

```
public abstract float interpolateH(float[] samples,
                                   float xfrac)
```

Performs horizontal interpolation on a 1-dimensional array of floating-point samples representing a row of samples. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`samples` - an array of floats.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

interpolateV

```
public float interpolateV(float[] samples,
                          float yfrac)
```

Performs vertical interpolation on a 1-dimensional array of floating-point samples representing a column of samples. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`samples` - an array of floats.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolate

```
public float interpolate(float[][] samples,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 2-dimensional array of floating-point samples. By default, this is implemented using a two-pass approach. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`samples` - an array of floats.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolateH

```
public float interpolateH(float s0,
                          float s1,
                          float xfrac)
```

Performs horizontal interpolation on a pair of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 2 and leftPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolateH

```
public float interpolateH(float s_,
                          float s0,
                          float s1,
                          float s2,
                          float xfrac)
```

Performs horizontal interpolation on a quadruple of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 4 and leftPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s_` - the sample to the left of the central sample.
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`s2` - the sample to the right of s1.
`xfrac` - the subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolateV

```
public float interpolateV(float s0,
                          float s1,
                          float yfrac)
```

Performs vertical interpolation on a pair of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 2 and topPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolateV

```
public float interpolateV(float s_,
                          float s0,
                          float s1,
                          float s2,
                          float yfrac)
```

Performs vertical interpolation on a quadruple of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 4 and topPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolate

```
public float interpolate(float s00,
                         float s01,
                         float s10,
                         float s11,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 2x2 grid of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == height == 2 and leftPadding == topPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolate

```
public float interpolate(float s__,
                         float s_0,
                         float s_1,
                         float s_2,
                         float s0_,
                         float s00,
                         float s01,
                         float s02,
                         float s1_,
                         float s10,
                         float s11,
                         float s12,
                         float s2_,
                         float s20,
                         float s21,
                         float s22,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 4x4 grid of floating-point samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == height == 4 and leftPadding == topPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

---

interpolateH

```
public abstract double interpolateH(double[] samples,
                                    float xfrac)
```

Performs horizontal interpolation on a 1-dimensional array of double samples representing a row of samples. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`samples` - an array of doubles.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolateV

```
public double interpolateV(double[] samples,
                           float yfrac)
```

Performs vertical interpolation on a 1-dimensional array of double samples representing a column of samples. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`samples` - an array of doubles.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolate

```
public double interpolate(double[][] samples,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 2-dimensional array of double samples. By default, this is implemented using a two-pass approach. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`samples` - an array of doubles.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolateH

```
public double interpolateH(double s0,
                           double s1,
                           float xfrac)
```

Performs horizontal interpolation on a pair of double samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 2 and leftPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolateH

```
public double interpolateH(double s_,
                           double s0,
                           double s1,
                           double s2,
                           float xfrac)
```

Performs horizontal interpolation on a quadruple of double samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == 4 and leftPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s_` - the sample to the left of the central sample.
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`s2` - the sample to the right of s1.
`xfrac` - the subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolateV

```
public double interpolateV(double s0,
                           double s1,
                           float yfrac)
```

Performs vertical interpolation on a pair of double samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 2 and topPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolateV

```
public double interpolateV(double s_,
                           double s0,
                           double s1,
                           double s2,
                           float yfrac)
```

Performs vertical interpolation on a quadruple of double samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if height == 4 and topPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

By default, vertical interpolation is identical to horizontal interpolation. Subclasses may choose to implement them differently.

Parameters:

`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolate

```
public double interpolate(double s00,
                          double s01,
                          double s10,
                          double s11,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 2x2 grid of double samples. Subclasses may implement this method to provide a speed improvement over the array method. This base class method merely calls the array method. It should only be called if width == height == 2 and leftPadding == topPadding == 0. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

---

interpolate

```
public double interpolate(double s__,
                          double s_0,
                          double s_1,
                          double s_2,
                          double s0_,
                          double s00,
                          double s01,
                          double s02,
                          double s1_,
                          double s10,
                          double s11,
                          double s12,
                          double s2_,
                          double s20,
                          double s21,
                          double s22,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 4x4 grid of double samples. It should only be called if width == height == 4 and leftPadding == topPadding == 1. The setting of subsampleBits need not have any effect on the interpolation accuracy of an implementation of this method.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

javax.media.jai

Class InterpolationBicubic

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
        |
        +--javax.media.jai.InterpolationTable
              |
              +--javax.media.jai.InterpolationBicubic
```

public final class **InterpolationBicubic**
extends InterpolationTable

A class representing bicubic interpolation.

InterpolationBicubic is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
 r(x) = (a + 2)|x|^3 - (a + 3)|x|^2         +  1 , 0 <= |x| < 1
 r(x) =       a|x|^3 -      5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
 r(x) = 0                                        , otherwise
```

with 'a' set to -0.5.

A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

This implementation creates an `InterpolationTable` whose integer coefficients have eight bits of precision to the right of the binary point.

The diagrams below illustrate the pixels involved in one-dimensional interpolation.

```
          Horizontal              Vertical

    s_    s0 .  s1    s2            s_
              ^
            xfrac                   s0
                                     .< yfrac
                                    s1

                                    s2
```

The diagram below illustrates the pixels involved in two-dimensional interpolation.

```
                s__    s_0    s_1    s_2



               s0_    s00    s01    s02

                          .             < yfrac

               s1_    s10    s11    s12



               s2_    s20    s21    s22
                          ^
                         xfrac
```

The class is marked 'final' so that it may be more easily inlined.

See Also:
    `Interpolation`

Field Detail

PRECISION_BITS

```
private static final int PRECISION_BITS
```

A

`private static final float **A**`

A3

`private static final float A3`

A2

`private static final float A2`

A0

`private static final float A0`

B3

`private static final float B3`

B2

`private static final float B2`

B1

`private static final float B1`

B0

`private static final float B0`

Constructor Detail

InterpolationBicubic

`public InterpolationBicubic(int subsampleBits)`

Constructs an InterpolationBicubic with a given subsample precision, in bits. This precision is applied to both axes.

This implementation creates an `InterpolationTable` whose integer coefficients have eight bits of precision to the right of the binary point.

Parameters:

`subsampleBits` - the subsample precision.

Method Detail

dataHelper

`private static float[] dataHelper(int subsampleBits)`

---

bicubic

`private static float bicubic(float x)`

Returns the bicubic polynomial value at a certain value of x.

javax.media.jai
Class InterpolationBicubic2

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
        |
        +--javax.media.jai.InterpolationTable
              |
              +--javax.media.jai.InterpolationBicubic2
```

public final class **InterpolationBicubic2**
extends InterpolationTable

A class representing bicubic interpolation using a different polynomial than InterpolationBicubic.

InterpolationBicubic2 is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
 r(x) = (a + 2)|x|^3 - (a + 3)|x|^2         +  1 , 0 <= |x| < 1
 r(x) =       a|x|^3 -      5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
 r(x) = 0                                        , otherwise
```

with ’a’ set to -1.0.

A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

This implementation creates an `InterpolationTable` whose integer coefficients have eight bits of precision to the right of the binary point.

The diagrams below illustrate the pixels involved in one-dimensional interpolation.

```
          Horizontal              Vertical

    s_    s0 .  s1    s2            s_
              ^
            xfrac                   s0
                                     .< yfrac
                                    s1

                                    s2
```

The diagram below illustrates the pixels involved in two-dimensional interpolation.

```
                s__    s_0    s_1    s_2



               s0_    s00    s01    s02

                          .             < yfrac

               s1_    s10    s11    s12



               s2_    s20    s21    s22
                          ^
                         xfrac
```

The class is marked ’final’ so that it may be more easily inlined.

Field Detail

PRECISION_BITS

`private static final int` **`PRECISION_BITS`**

A

`private static final float **A**`

A3

`private static final float A3`

A2

`private static final float A2`

A0

`private static final float A0`

B3

`private static final float B3`

B2

`private static final float B2`

B1

`private static final float B1`

B0

`private static final float B0`

Constructor Detail

InterpolationBicubic2

`public InterpolationBicubic2(int subsampleBits)`

Constructs an InterpolationBicubic2 with a given subsample precision, in bits. This precision is applied to both axes.

This implementation creates an `InterpolationTable` whose integer coefficients have eight bits of precision to the right of the binary point.

Parameters:

`subsampleBits` - the subsample precision.

Method Detail

dataHelper

`private static float[] dataHelper(int subsampleBits)`

---

bicubic

`private static float bicubic(float x)`

Returns the bicubic polynomial value at a certain value of x.

javax.media.jai

Class InterpolationBilinear

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
        |
        +--javax.media.jai.InterpolationBilinear
```

public final class **InterpolationBilinear**
extends Interpolation

A class representing bilinear interpolation. The class is marked 'final' so it may be either automatically or manually inlined.

Bilinear interpolation requires a neighborhood extending one pixel to the right and below the central sample. If the subsample position is given by (u, v), the resampled pixel value will be:

```
 (1 - v)*[(1 - u)*p00 + u*p01] + v*[(1 - u)*p10 + u*p11]
```

A neighborhood extending one sample to the right of, and one sample below the central sample is required to perform bilinear interpolation. This implementation maintains equal subsampleBits in x and y.

The diagrams below illustrate the pixels involved in one-dimensional bilinear interpolation.

```
          Horizontal              Vertical

          s0 .  s1                  s0
               ^                     .< yfrac
            xfrac                   s1
```

The diagram below illustrates the pixels involved in two-dimensional bilinear interpolation.

```
                       s00    s01

                          .      < yfrac

                      s10    s11
                          ^
                         xfrac
```

The class is marked 'final' so that it may be more easily inlined.

Field Detail

one

`private int one`

round

`private int round`

shift

`private int shift`

round2

`private int round2`

shift2

`private int shift2`

DEFAULT_SUBSAMPLE_BITS

```
static final int DEFAULT_SUBSAMPLE_BITS
```

Constructor Detail

InterpolationBilinear

```
public InterpolationBilinear(int subsampleBits)
```

Constructs an InterpolationBilinear with a given subsample precision, in bits. This precision is applied to both axes.

Parameters:

`subsampleBits` - the subsample precision.

InterpolationBilinear

```
public InterpolationBilinear()
```

Constructs an InterpolationBilinear with the default subsample precision.

Method Detail

interpolateH

```
public final int interpolateH(int[] samples,
                              int xfrac)
```

Performs horizontal interpolation on a one-dimensional array of integral samples.

Parameters:

`samples` - an array of ints.
`xfrac` - the subsample position, multiplied by 2^(subsampleBits).

Returns:

the interpolated value as an int.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final int interpolateV(int[] samples,
                              int yfrac)
```

Performs vertical interpolation on a one-dimensional array of integral samples.

Parameters:

`samples` - an array of ints.
`yfrac` - the Y subsample position, multiplied by 2^(subsampleBits).

Returns:

the interpolated value as an int.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public final int interpolate(int[][] samples,
                             int xfrac,
                             int yfrac)
```

Performs interpolation on a two-dimensional array of integral samples.

Parameters:

`samples` - a two-dimensional array of ints.
`xfrac` - the X subsample position, multiplied by 2^(subsampleBits).
`yfrac` - the Y subsample position, multiplied by 2^(subsampleBits).

Returns:

the interpolated value as an int.

Overrides:

interpolate in class Interpolation

---

interpolateH

```
public final int interpolateH(int s0,
                              int s1,
                              int xfrac)
```

Performs horizontal interpolation on a pair of integral samples. This method may be used instead of the array version for speed.

Parameters:

`s0` - the central sample.

`s1` - the sample to the right of the central sample.

`xfrac` - the subsample position, multiplied by 2^(subsampleBits).

Returns:

the interpolated value as an int.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final int interpolateV(int s0,
                              int s1,
                              int yfrac)
```

Performs vertical interpolation on a pair of integral samples. This method may be used instead of the array version for speed.

Parameters:

`s0` - the central sample.

`s1` - the sample below the central sample.

`yfrac` - the Y subsample position, multiplied by 2^(subsampleBits).

Returns:

the interpolated value as an int.

Overrides:

interpolateV in class Interpolation

---

interpolateH

```
public final int interpolateH(int s_,
                              int s0,
                              int s1,
                              int s2,
                              int xfrac)
```

Performs horizontal interpolation on a quadruple of integral samples. The outlying samples are ignored.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final int interpolateV(int s_,
                              int s0,
                              int s1,
                              int s2,
                              int yfrac)
```

Performs vertical interpolation on a quadruple of integral samples. The outlying samples are ignored.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public final int interpolate(int s00,
                             int s01,
                             int s10,
                             int s11,
                             int xfrac,
                             int yfrac)
```

Performs interpolation on a 2x2 grid of integral samples.

Parameters:

`s00` - the central sample.

`s01` - the sample to the right of the central sample.

`s10` - the sample below the central sample.

`s11` - the sample below and to the right of the central sample.

`xfrac` - the X subsample position, multiplied by 2^(subsampleBits).
`yfrac` - the Y subsample position, multiplied by 2^(subsampleBits).

Returns:
the interpolated value as an int.

Overrides:
interpolate in class Interpolation

---

interpolate

```
public final int interpolate(int s__,
                             int s_0,
                             int s_1,
                             int s_2,
                             int s0_,
                             int s00,
                             int s01,
                             int s02,
                             int s1_,
                             int s10,
                             int s11,
                             int s12,
                             int s2_,
                             int s20,
                             int s21,
                             int s22,
                             int xfrac,
                             int yfrac)
```

Performs interpolation on a 4x4 grid of integral samples. The outlying samples are ignored.

Overrides:
interpolate in class Interpolation

---

interpolateH

```
public final float interpolateH(float[] samples,
                                float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of floating-point samples.

Parameters:
`samples` - an array of floats.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a float.

Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final float interpolateV(float[] samples,
                                float yfrac)
```

Performs vertical interpolation on a one-dimensional array of floating-point samples.

Parameters:
`samples` - an array of floats.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a float.

Overrides:
interpolateV in class Interpolation

---

interpolate

```
public final float interpolate(float[][] samples,
                               float xfrac,
                               float yfrac)
```

Performs interpolation on a two-dimensional array of floating-point samples.

Parameters:
`samples` - an array of floats.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a float.
Overrides:
interpolate in class Interpolation

---

interpolateH

```
public final float interpolateH(float s0,
                                float s1,
                                float xfrac)
```

Performs horizontal interpolation on a horizontal pair of floating-point samples. This method may be used instead of the array version for speed.
Parameters:
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a float.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final float interpolateV(float s0,
                                float s1,
                                float yfrac)
```

Performs vertical interpolation on a vertical pair of floating-point samples. This method may be used instead of the array version for speed.
Parameters:
`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a float.
Overrides:
interpolateV in class Interpolation

---

interpolateH

```
public final float interpolateH(float s_,
                                float s0,
                                float s1,
                                float s2,
                                float frac)
```

Performs horizontal interpolation on a horizontal quad of floating-point samples. The outlying samples are ignored.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final float interpolateV(float s_,
                                float s0,
                                float s1,
                                float s2,
                                float frac)
```

Performs vertical interpolation on a horizontal quad of floating-point samples. The outlying samples are ignored.
Overrides:
interpolateV in class Interpolation

---

interpolate

```
public final float interpolate(float s00,
                               float s01,
                               float s10,
                               float s11,
                               float xfrac,
                               float yfrac)
```

Performs interpolation on a 2x2 grid of floating-point samples.
Parameters:
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a float.
Overrides:
interpolate in class Interpolation

---

interpolate

```
public final float interpolate(float s__,
                               float s_0,
                               float s_1,
                               float s_2,
                               float s0_,
                               float s00,
                               float s01,
                               float s02,
                               float s1_,
                               float s10,
                               float s11,
                               float s12,
                               float s2_,
                               float s20,
                               float s21,
                               float s22,
                               float xfrac,
                               float yfrac)
```

Performs interpolation on a 4x4 grid. The outlying samples are ignored.
Overrides:
interpolate in class Interpolation

---

interpolateH

```
public final double interpolateH(double[] samples,
                                 float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of double samples.
Parameters:
`samples` - an array of doubles.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final double interpolateV(double[] samples,
                                 float yfrac)
```

Performs vertical interpolation on a one-dimensional array of double samples.
Parameters:
`samples` - an array of doubles.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Overrides:
interpolateV in class Interpolation

---

interpolate

```
public final double interpolate(double[][] samples,
                                float xfrac,
                                float yfrac)
```

Performs interpolation on a two-dimensional array of double samples.

Parameters:

`samples` - an array of doubles.

`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

Overrides:

interpolate in class Interpolation

---

interpolateH

```
public final double interpolateH(double s0,
                                 double s1,
                                 float xfrac)
```

Performs horizontal interpolation on a horizontal pair of double samples. This method may be used instead of the array version for speed.

Parameters:

`s0` - the central sample.

`s1` - the sample to the right of the central sample.

`xfrac` - the subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final double interpolateV(double s0,
                                 double s1,
                                 float yfrac)
```

Performs vertical interpolation on a vertical pair of double samples. This method may be used instead of the array version for speed.

Parameters:

`s0` - the central sample.

`s1` - the sample below the central sample.

`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

Overrides:

interpolateV in class Interpolation

---

interpolateH

```
public final double interpolateH(double s_,
                                 double s0,
                                 double s1,
                                 double s2,
                                 float xfrac)
```

Performs interpolation on a horizontal quad of double samples. The outlying samples are ignored.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final double interpolateV(double s_,
                                 double s0,
                                 double s1,
                                 double s2,
                                 float yfrac)
```

Performs vertical interpolation on a vertical quad of double samples. The outlying samples are ignored.
Overrides:
interpolateV in class Interpolation

---

interpolate

```
public final double interpolate(double s00,
                                double s01,
                                double s10,
                                double s11,
                                float xfrac,
                                float yfrac)
```

Performs interpolation on a 2x2 grid of double samples.
Parameters:
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Overrides:
interpolate in class Interpolation

---

interpolate

```
public final double interpolate(double s__,
                                double s_0,
                                double s_1,
                                double s_2,
                                double s0_,
                                double s00,
                                double s01,
                                double s02,
                                double s1_,
                                double s10,
                                double s11,
                                double s12,
                                double s2_,
                                double s20,
                                double s21,
                                double s22,
                                float xfrac,
                                float yfrac)
```

Performs interpolation on a 4x4 grid. The outlying samples are ignored.
Overrides:
interpolate in class Interpolation

javax.media.jai
Class InterpolationNearest

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
        |
        +--javax.media.jai.InterpolationNearest
```

public final class **InterpolationNearest**
extends Interpolation

A class representing nearest-neighbor interpolation. Since nearest-neighbor interpolation is simply pixel copying, and not really interpolation at all, most code that performs nearest-neighbor sampling will want to use special-purpose code. However, this class is provided both as a way to specify such interpolation, with the consumer making use of ’instanceof’ to detect the particular class, and as a way to force general Interpolation users to use nearest-neighbor sampling.

Neighborhoods of sizes 2x1, 1x2, 2x2, 4x1, 1x4, 4x4, Nx1 and 1xN, that is, all the interpolate() methods defined in the Interpolation class, are supported in the interest of simplifying code that handles a number of types of interpolation. In each case, the central sample is returned and the rest are ignored.

The class is marked ’final’ so that it may be more easily inlined.

Constructor Detail

InterpolationNearest

```
public InterpolationNearest()
```

Constructs an `InterpolationNearest`. The return value of `getSubsampleBitsH()` and `getSubsampleBitsV()` will be 0.

Method Detail

interpolateH

```
public final int interpolateH(int[] samples,
                              int xfrac)
```

Performs horizontal interpolation on a one-dimensional array of integral samples. The central sample (samples[0]) is returned.

Overrides:
interpolateH in class Interpolation

interpolateV

```
public final int interpolateV(int[] samples,
                              int yfrac)
```

Performs vertical interpolation on a one-dimensional array of integral samples. The central sample (samples[0]) is returned.

Overrides:
interpolateV in class Interpolation

interpolate

```
public final int interpolate(int[][] samples,
                             int xfrac,
                             int yfrac)
```

Performs interpolation on a two-dimensional array of integral samples. The central sample (samples[0][0]) is returned.

Overrides:
interpolate in class Interpolation

interpolateH

```
public final int interpolateH(int s0,
                              int s1,
                              int xfrac)
```

Performs horizontal interpolation on a pair of integral samples. The central sample (s0) is returned.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final int interpolateV(int s0,
                              int s1,
                              int yfrac)
```

Performs vertical interpolation on a pair of integral samples. The central sample (s0) is returned.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public int interpolate(int s00,
                       int s01,
                       int s10,
                       int s11,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 2x2 grid of integral samples. The central sample (s00) is returned.

Overrides:

interpolate in class Interpolation

---

interpolate

```
public int interpolate(int s__,
                       int s_0,
                       int s_1,
                       int s_2,
                       int s0_,
                       int s00,
                       int s01,
                       int s02,
                       int s1_,
                       int s10,
                       int s11,
                       int s12,
                       int s2_,
                       int s20,
                       int s21,
                       int s22,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 4x4 grid of integral samples. The central sample (s00) is returned.

Overrides:

interpolate in class Interpolation

---

interpolateH

```
public final float interpolateH(float[] samples,
                                float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of floating-point samples. The central sample (s0) is returned.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final float interpolateV(float[] samples,
                                float yfrac)
```

Performs vertical interpolation on a one-dimensional array of floating-point samples. The central sample (s0) is returned.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public final float interpolate(float[][] samples,
                               float xfrac,
                               float yfrac)
```

Performs interpolation on a two-dimensional array of floating-point samples. The central sample (samples[0][0]) is returned.

Overrides:

interpolate in class Interpolation

---

interpolateH

```
public final float interpolateH(float s0,
                                float s1,
                                float xfrac)
```

Performs horizontal interpolation on a pair of floating-point samples. The central sample (s0) is returned.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public final float interpolateV(float s0,
                                float s1,
                                float yfrac)
```

Performs vertical interpolation on a pair of floating-point samples. The central sample (s0) is returned.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public float interpolate(float s00,
                         float s01,
                         float s10,
                         float s11,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 2x2 grid of floating-point samples. The central sample (s00) is returned.

Overrides:

interpolate in class Interpolation

---

interpolate

```
public float interpolate(float s__,
                         float s_0,
                         float s_1,
                         float s_2,
                         float s0_,
                         float s00,
                         float s01,
                         float s02,
                         float s1_,
                         float s10,
                         float s11,
                         float s12,
                         float s2_,
                         float s20,
                         float s21,
                         float s22,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 4x4 grid of floating-point samples. The central sample (s00) is returned.
Overrides:
interpolate in class Interpolation

---

interpolateH

```
public final double interpolateH(double[] samples,
                                 float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of double samples. The central sample (s0) is returned.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final double interpolateV(double[] samples,
                                 float yfrac)
```

Performs vertical interpolation on a one-dimensional array of double samples. The central sample (s0) is returned.
Overrides:
interpolateV in class Interpolation

---

interpolate

```
public final double interpolate(double[][] samples,
                                float xfrac,
                                float yfrac)
```

Performs interpolation on a two-dimensional array of double samples. The central sample (samples[0][0]) is returned.
Overrides:
interpolate in class Interpolation

---

interpolateH

```
public final double interpolateH(double s0,
                                 double s1,
                                 float xfrac)
```

Performs horizontal interpolation on a pair of double samples. The central sample (s0) is returned.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public final double interpolateV(double s0,
                                 double s1,
                                 float yfrac)
```

Performs vertical interpolation on a pair of double samples. The central sample (s0) is returned.
Overrides:
interpolateV in class Interpolation

---

interpolate

```
public double interpolate(double s00,
                          double s01,
                          double s10,
                          double s11,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 2x2 grid of double samples. The central sample (s00) is returned.
Overrides:
interpolate in class Interpolation

---

interpolate

```
public double interpolate(double s__,
                          double s_0,
                          double s_1,
                          double s_2,
                          double s0_,
                          double s00,
                          double s01,
                          double s02,
                          double s1_,
                          double s10,
                          double s11,
                          double s12,
                          double s2_,
                          double s20,
                          double s21,
                          double s22,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 4x4 grid of double samples. The central sample (s00) is returned.

Overrides:

interpolate in class Interpolation

javax.media.jai

Class InterpolationTable

```
java.lang.Object
  |
  +--javax.media.jai.Interpolation
        |
        +--javax.media.jai.InterpolationTable
```

Direct Known Subclasses:

InterpolationBicubic, InterpolationBicubic2

public class **InterpolationTable**
extends Interpolation

A subclass of Interpolation that uses tables to store the interpolation kernels. The set of subpixel positions is broken up into a fixed number of "bins" and a distinct kernel is used for each bin. The number of bins must be a power of two.

An InterpolationTable defines a separable interpolation, with a set of kernels for each dimension. The number of bins may vary between the two dimensions. The kernels are stored in double precision, floating- and fixed-point form. The fixed point representation has a user-specified fractional precision. It is the user’s responsibility to specify an appropriate level of precision that will not cause overflow when accumulating the results of a convolution against a set of source pixels, using 32-bit integer arithmetic.

Field Detail

precisionBits

`protected int` **`precisionBits`**

The number of fractional bits used to describe filter coefficients.

round

`private int` **`round`**

The number 1/2 with precisionBits of fractional precision.

numSubsamplesH

`private int` **`numSubsamplesH`**

The number of horizontal subpixel positions within a pixel.

numSubsamplesV

`private int` **`numSubsamplesV`**

The number of vertical subpixel positions within a pixel.

dataHd

`protected double[]` **`dataHd`**

The horizontal coefficient data in double format.

dataVd

`protected double[]` **`dataVd`**

The vertical coefficient data in double format.

dataHf

```
protected float[] dataHf
```

The horizontal coefficient data in floating-point format.

---

dataVf

```
protected float[] dataVf
```

The vertical coefficient data in floating-point format.

---

dataHi

```
protected int[] dataHi
```

The horizontal coefficient data in fixed-point format.

---

dataVi

```
protected int[] dataVi
```

The vertical coefficient data in fixed-point format.

Constructor Detail

InterpolationTable

```
public InterpolationTable(int leftPadding,
                          int topPadding,
                          int width,
                          int height,
                          int subsampleBitsH,
                          int subsampleBitsV,
                          int precisionBits,
                          int[] dataH,
                          int[] dataV)
```

Constructs an InterpolationTable with specified horizontal and vertical extents (support), number of horizontal and vertical bins, fixed-point fractional precision, and int kernel entries. The kernel data values are organized as $2^{subsampleBits}$ entries each containing width ints.

dataH and dataV are required to contain width * $2^{subsampleBitsH}$ and height * $2^{subsampleBitsV}$ entries respectively, otherwise an IllegalArgumentException will be thrown.

If dataV is null, it is assumed to be a copy of dataH and the topPadding, height, and subsampleBitsV parameters are ignored.

Parameters:

`leftPadding` - The number of samples to the left of the central sample to be used during horizontal resampling.
`topPadding` - The number of samples above the central sample to be used during vertical resampling.
`width` - the width of a horizontal resampling kernel.
`height` - the height of a vertical resampling kernel. Ignored if dataV is null.
`subsampleBitsH` - the log (base 2) of the number of horizontal subsample positions.
`subsampleBitsV` - the log (base 2) of the number of vertical subsample positions. Ignored if dataV is null.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values. The same value is used for both horizontal and vertical resampling.
`dataH` - the horizontal table entries, as an int array of $2^{subsampleBitsH}$ entries each of length width.
`dataV` - the vertical table entries, as an int array of $2^{subsampleBitsV}$ entries each of length height, or null. If null, the dataH table is used for vertical interpolation as well and the topPadding, height, and subsampleBitsV parameters are ignored.

Throws:

java.lang.IllegalArgumentException - if the size of the data arrays are incorrect.

InterpolationTable

```
public InterpolationTable(int padding,
                          int width,
                          int subsampleBits,
                          int precisionBits,
                          int[] data)
```

Constructs an InterpolationTable with identical horizontal and vertical resampling kernels.
Parameters:
`padding` - The number of samples to the left or above the central sample to be used during resampling.
`width` - the width or height of a resampling kernel.
`subsampleBits` - the log (base 2) of the number of subsample positions.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values.
`data` - the kernel entries, as an int array of width*$2^{\mathtt{subsampleBits}}$ entries

---

InterpolationTable

```
public InterpolationTable(int leftPadding,
                          int topPadding,
                          int width,
                          int height,
                          int subsampleBitsH,
                          int subsampleBitsV,
                          int precisionBits,
                          float[] dataH,
                          float[] dataV)
```

Constructs an InterpolationTable with specified horizontal and vertical extents (support), number of horizontal and vertical bins, fixed-point fractional precision, and float kernel entries. The kernel data values are organized as $2^{\mathtt{subsampleBits}}$ entries each containing width floats.

dataH and dataV are required to contain width * $2^{\mathtt{subsampleBitsH}}$ and height * $2^{\mathtt{subsampleBitsV}}$ entries respectively, otherwise an IllegalArgumentException will be thrown.

If dataV is null, it is assumed to be a copy of dataH and the topPadding, height, and subsampleBitsV parameters are ignored.
Parameters:
`leftPadding` - The number of samples to the left of the central sample to be used during horizontal resampling.
`topPadding` - The number of samples above the central sample to be used during vertical resampling.
`width` - the width of a horizontal resampling kernel.
`height` - the height of a vertical resampling kernel. Ignored if dataV is null.
`subsampleBitsH` - the log (base 2) of the number of horizontal subsample positions.
`subsampleBitsV` - the log (base 2) of the number of vertical subsample positions. Ignored if dataV is null.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values. The same value is used for both horizontal and vertical resampling.
`dataH` - the horizontal table entries, as a float array of $2^{\mathtt{subsampleBitsH}}$ entries each of length width.
`dataV` - the vertical table entries, as a float array of $2^{\mathtt{subsampleBitsV}}$ entries each of length height, or null. If null, the dataH table is used for vertical interpolation as well and the topPadding, height, and subsampleBitsV parameters are ignored.
Throws:
java.lang.IllegalArgumentException - if the size of the data arrays are incorrect.

InterpolationTable

```
public InterpolationTable(int padding,
                          int width,
                          int subsampleBits,
                          int precisionBits,
                          float[] data)
```

Constructs an InterpolationTable with identical horizontal and vertical resampling kernels.
Parameters:
`padding` - The number of samples to the left or above the central sample to be used during resampling.
`width` - the width or height of a resampling kernel.
`subsampleBits` - the log (base 2) of the number of subsample positions.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values.
`data` - the kernel entries, as a float array of width*$2^{\mathtt{subsampleBits}}$ entries

---

InterpolationTable

```
public InterpolationTable(int leftPadding,
                          int topPadding,
                          int width,
                          int height,
                          int subsampleBitsH,
                          int subsampleBitsV,
                          int precisionBits,
                          double[] dataH,
                          double[] dataV)
```

Constructs an InterpolationTable with specified horizontal and vertical extents (support), number of horizontal and vertical bins, fixed-point fractional precision, and double kernel entries. The kernel data values are organized as $2^{\texttt{subsampleBits}}$ entries each containing width doubles.

dataH and dataV are required to contain width * $2^{\texttt{subsampleBitsH}}$ and height * $2^{\texttt{subsampleBitsV}}$ entries respectively, otherwise an IllegalArgumentException will be thrown.

If dataV is null, it is assumed to be a copy of dataH and the topPadding, height, and subsampleBitsV parameters are ignored.

Parameters:

`leftPadding` - The number of samples to the left of the central sample to be used during horizontal resampling.
`topPadding` - The number of samples above the central sample to be used during vertical resampling.
`width` - the width of a horizontal resampling kernel.
`height` - the height of a vertical resampling kernel. Ignored if dataV is null.
`subsampleBitsH` - the log (base 2) of the number of horizontal subsample positions.
`subsampleBitsV` - the log (base 2) of the number of vertical subsample positions. Ignored if dataV is null.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values. The same value is used for both horizontal and vertical resampling.
`dataH` - the horizontal table entries, as a double array of $2^{\texttt{subsampleBitsH}}$ entries each of length width.
`dataV` - the vertical table entries, as a double array of $2^{\texttt{subsampleBitsV}}$ entries each of length height, or null. If null, the dataH table is used for vertical interpolation as well and the topPadding, height, and subsampleBitsV parameters are ignored.

InterpolationTable

```
public InterpolationTable(int padding,
                          int width,
                          int subsampleBits,
                          int precisionBits,
                          double[] data)
```

Constructs an InterpolationTable with identical horizontal and vertical resampling kernels.

Parameters:

`padding` - The number of samples to the left or above the central sample to be used during resampling.
`width` - the width or height of a resampling kernel.
`subsampleBits` - the log (base 2) of the number of subsample positions.
`precisionBits` - the number of bits of fractional precision to be used when resampling integral sample values.
`data` - the kernel entries, as a double array of width*$2^{\texttt{subsampleBitsH}}$ entries

Method Detail

getPrecisionBits

```
public int getPrecisionBits()
```

Returns the number of bits of fractional precision used to store the fixed-point table entries.

---

getHorizontalTableData

```
public int[] getHorizontalTableData()
```

Returns the integer (fixed-point) horizontal table data. The output is an `int` array of length `getWidth() *` $2^{\texttt{getSubsampleBitsH()}}$.

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getWidth()` samples at a given fractional position (bin) `xfrac` between `0` and $2^{\texttt{getSubsampleBitsH() - 1}}$:

```
 int interpolateH(InterpolationTable interp, int[] samples, int xfrac) {
     int[] dataH = interp.getHorizontalTableData();
     int precisionBits = interp.getPrecisionBits();
     int round = 1 << (precisionBits - 1);
     int width = interp.getWidth();
     int offset = width*xfrac;

     int sum = 0;
     for (int i = 0; i < width; i++) {
         sum += dataH[offset + i]*samples[i];
     }
     return (sum + round) >> precisionBits;
 }
```

In practice, the values `dataH`, `precisionBits`, etc., may be extracted once and reused to interpolate multiple output pixels.

Returns:
An array of `int`s.

---

getVerticalTableData

`public int[] **getVerticalTableData**()`

Returns the integer (fixed-point) vertical table data. The output is an `int` array of length `getHeight() *` $2^{\texttt{getSubsampleBitsV()}}$.

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getHeight()` samples at a given fractional position (bin) `yfrac` between `0` and $2^{\texttt{getSubsampleBitsV() - 1}}$:

```
 int interpolateV(InterpolationTable interp, int[] samples, int yfrac) {
     int[] dataV = interp.getVerticalTableData();
     int precisionBits = interp.getPrecisionBits();
     int round = 1 << (precisionBits - 1);
     int height = interp.getHeight();
     int offset = height*yfrac;

     int sum = 0;
     for (int i = 0; i < height; i++) {
         sum += dataV[offset + i]*samples[i];
     }
     return (sum + round) >> precisionBits;
 }
```

In practice, the values `dataV`, `precisionBits`, etc., may be extracted once and reused to interpolate multiple output pixels.

Returns:
An array of `int`s.

---

getHorizontalTableDataFloat

`public float[] **getHorizontalTableDataFloat**()`

Returns the floating-point horizontal table data. The output is a `float` array of length `getWidth() *` $2^{\texttt{getSubsampleBitsH()}}$.

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getWidth()` floating-point samples at a given fractional position `xfrac` between `0.0F` and `1.0F`:

```
 float interpolateH(InterpolationTable interp,
                    float[] samples, float xfrac) {
     float[] dataH = interp.getHorizontalTableDataFloat();
     int width = interp.getWidth();
     int numSubsamplesH = 1 << getSubsampleBitsH();
     int ifrac = (int)(xfrac*numSubsamplesH);
     int offset = width*ifrac;

     float sum = 0.0F;
     for (int i = 0; i < width; i++) {
         sum += dataH[offset + i]*samples[i];
     }
     return sum;
 }
```

In practice, the values `dataH`, `numSubsamplesH`, etc., may be extracted once and reused to interpolate multiple output pixels.

Returns:
An array of `float`s.

---

getVerticalTableDataFloat

`public float[] **getVerticalTableDataFloat**()`

Returns the floating-point vertical table data. The output is a `float` array of length `getWidth() *` $2^{\texttt{getSubsampleBitsV()}}$.

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getHeight()` floating-point samples at a given fractional position `yfrac` between `0.0F` and `1.0F`:

```
float interpolateV(InterpolationTable interp,
                   float[] samples, float yfrac) {
    float[] dataV = interp.getVerticalTableDataFloat();
    int height = interp.getHeight();
    int numSubsamplesV = 1 << getSubsampleBitsV();
    int ifrac = (int)(yfrac*numSubsamplesV);
    int offset = height*ifrac;

    float sum = 0.0F;
    for (int i = 0; i < height; i++) {
        sum += dataV[offset + i]*samples[i];
    }
    return sum;
}
```

In practice, the values `dataV`, `numSubsamplesV`, etc., may be extracted once and reused to interpolate multiple output pixels.

Returns:

An array of `float`s.

---

getHorizontalTableDataDouble

`public double[] getHorizontalTableDataDouble()`

Returns the double horizontal table data. The output is a `double` array of length `getWidth() *` $2^{\texttt{getSubsampleBitsH()}}$.

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getWidth()` double samples at a given fractional position `xfrac` between `0.0F` and `1.0F`:

```
double interpolateH(InterpolationTable interp,
                    double[] samples, float xfrac) {
    double[] dataH = interp.getHorizontalTableDataDouble();
    int width = interp.getWidth();
    int numSubsamplesH = 1 << getSubsampleBitsH();
    int ifrac = (int)(xfrac*numSubsamplesH);
    int offset = width*ifrac;

    double sum = 0.0;
    for (int i = 0; i < width; i++) {
        sum += dataH[offset + i]*samples[i];
    }
    return sum;
}
```

In practice, the values `dataH`, `numSubsamplesH`, etc., may be extracted once and reused to interpolate multiple output pixels.

Returns:

An array of `double`s.

---

getVerticalTableDataDouble

`public double[] getVerticalTableDataDouble()`

Returns the double vertical table data. The output is a `double` array of length `getHeight() *` $2^{\texttt{getSubsampleBitsV()}}$).

The following code, given an instance `interp` of class `InterpolationTable`, will perform interpolation of a set of `getHeight()` double samples at a given fractional position `yfrac` between `0.0F` and `1.0F`:

```
double interpolateV(InterpolationTable interp,
                    double[] samples, float yfrac) {
    double[] dataV = interp.getVerticalTableDataDouble();
    int height = interp.getHeight();
    int numSubsamplesV = 1 << getSubsampleBitsV();
    int ifrac = (int)(yfrac*numSubsamplesV);
    int offset = height*ifrac;

    double sum = 0.0;
    for (int i = 0; i < height; i++) {
        sum += dataV[offset + i]*samples[i];
    }
    return sum;
}
```

In practice, the values `dataV`, `numSubsamplesV`, etc., may be extracted once and reused to interpolate multiple output pixels.
Returns:
An array of `double`s.

---

interpolateH

```
public int interpolateH(int[] samples,
                        int xfrac)
```

Performs horizontal interpolation on a one-dimensional array of integral samples. If xfrac does not lie between 0 and $2^{\mathtt{subsampleBitsH-1}}$, an ArrayIndexOutOfBoundsException may occur, where width is the width of the horizontal resampling kernel.
Parameters:
`samples` - an array of ints.
`xfrac` - the subsample position, multiplied by $2^{\mathtt{subsampleBitsH}}$.
Returns:
the interpolated value as an int.
Throws:
ArrayIndexOutOfBoundsException - if xfrac is out of bounds.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public int interpolateV(int[] samples,
                        int yfrac)
```

Performs vertical interpolation on a one-dimensional array of integral samples. If yfrac does not lie between 0 and $2^{\mathtt{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where height is the height of the vertical resampling kernel.
Parameters:
`samples` - an array of ints.
`yfrac` - the Y subsample position, multiplied by $2^{\mathtt{subsampleBitsV}}$.
Returns:
the interpolated value as an int.
Throws:
ArrayIndexOutOfBoundsException - if yfrac is out of bounds.
Overrides:
interpolateV in class Interpolation

---

interpolateH

```
public int interpolateH(int s0,
                        int s1,
                        int xfrac)
```

Performs horizontal interpolation on a pair of integral samples. This method may be used instead of the array version for speed. It should only be called if width == 2. If xfrac does not lie between 0 and $2^{\mathtt{subsampleBitsH-1}}$, an ArrayIndexOutOfBoundsException may occur, where width is the width of the horizontal resampling kernel.
Parameters:
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the subsample position, multiplied by $2^{\mathtt{subsampleBitsH}}$.
Returns:
the interpolated value as an int.
Throws:
ArrayIndexOutOfBoundsException - if xfrac is out of bounds.
Overrides:
interpolateH in class Interpolation

---

interpolateH

```
public int interpolateH(int s_,
                        int s0,
                        int s1,
                        int s2,
                        int xfrac)
```

Performs horizontal interpolation on a quadruple of integral samples. This method may be used instead of the array version for speed. It should only be called if width == 4 and leftPadding == 1. If xfrac does not lie between 0 and $2^{\text{subsampleBitsH-1}}$, an ArrayIndexOutOfBoundsException may occur, where width is the width of the horizontal resampling kernel.

Parameters:

`s_` - the sample to the left of the central sample.
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`s2` - the sample to the right of s1.
`xfrac` - the subsample position, multiplied by $2^{\text{subsampleBitsH}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if xfrac is out of bounds.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public int interpolateV(int s0,
                        int s1,
                        int yfrac)
```

Performs vertical interpolation on a pair of integral samples. This method may be used instead of the array version for speed. It should only be called if height == 2 and topPadding == 0. If yfrac does not lie between 0 and $2^{\text{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where height is the height of the vertical resampling kernel.

Parameters:

`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, multiplied by $2^{\text{subsampleBitsV}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:

interpolateV in class Interpolation

---

interpolateV

```
public int interpolateV(int s_,
                        int s0,
                        int s1,
                        int s2,
                        int yfrac)
```

Performs vertical interpolation on a quadruple of integral samples. This method may be used instead of the array version for speed. It should only be called if height == 4 and topPadding == 1. If yfrac does not lie between 0 and $2^{\text{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where height is the height of the vertical resampling kernel.

Parameters:

`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, multiplied by $2^{\text{subsampleBitsV}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public int interpolate(int s00,
                       int s01,
                       int s10,
                       int s11,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 2x2 grid of integral samples. It should only be called if width == height == 2 and leftPadding == topPadding == 0. If xfrac does not lie between 0 and $2^{\text{subsampleBitsH-1}}$, or yfrac does not lie between 0 and $2^{\text{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where width and height are the width and height of the horizontal and vertical resampling kernels respectively.

Parameters:

`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, multiplied by $2^{\text{subsampleBitsH}}$.
`yfrac` - the Y subsample position, multiplied by $2^{\text{subsampleBitsV}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:

interpolate in class Interpolation

---

interpolate

```
public int interpolate(int s__,
                       int s_0,
                       int s_1,
                       int s_2,
                       int s0_,
                       int s00,
                       int s01,
                       int s02,
                       int s1_,
                       int s10,
                       int s11,
                       int s12,
                       int s2_,
                       int s20,
                       int s21,
                       int s22,
                       int xfrac,
                       int yfrac)
```

Performs interpolation on a 4x4 grid of integral samples. It should only be called if width == height == 4 and leftPadding == topPadding == 1. If xfrac does not lie between 0 and $2^{\text{subsampleBitsH-1}}$, or yfrac does not lie between 0 and $2^{\text{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where width and height are the the width and height of the horizontal and vertical resampling kernels respectively.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, multiplied by $2^{\text{subsampleBitsH}}$.
`yfrac` - the Y subsample position, multiplied by $2^{\text{subsampleBitsV}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:

interpolate in class Interpolation

interpolateF

```
public int interpolateF(int s__,
                        int s_0,
                        int s_1,
                        int s_2,
                        int s0_,
                        int s00,
                        int s01,
                        int s02,
                        int s1_,
                        int s10,
                        int s11,
                        int s12,
                        int s2_,
                        int s20,
                        int s21,
                        int s22,
                        int xfrac,
                        int yfrac)
```

Performs interpolation on a 4x4 grid of integral samples. All internal calculations are performed in floating-point. It should only be called if width == height == 4 and leftPadding == topPadding == 1. If xfrac does not lie between 0 and $2^{\texttt{subsampleBitsH-1}}$, or yfrac does not lie between 0 and $2^{\texttt{subsampleBitsV-1}}$, an ArrayIndexOutOfBoundsException may occur, where width and height are the width and height of horizontal and vertical resampling kernels respectively.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, multiplied by $2^{\texttt{subsampleBitsH}}$.
`yfrac` - the Y subsample position, multiplied by $2^{\texttt{subsampleBitsV}}$.

Returns:

the interpolated value as an int.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

interpolateH

```
public float interpolateH(float[] samples,
                          float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of floating-point samples representing a row of samples. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`samples` - an array of floats.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if xfrac is out of bounds.

Overrides:

interpolateH in class Interpolation

interpolateV

```
public float interpolateV(float[] samples,
                          float yfrac)
```

Performs vertical interpolation on a one-dimensional array of floating-point samples representing a column of samples. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`samples` - an array of floats.

`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:

interpolateV in class Interpolation

---

interpolateH

```
public float interpolateH(float s0,
                          float s1,
                          float xfrac)
```

Performs horizontal interpolation on a pair of floating-point samples. This method may be used instead of the array version for speed. It should only be called if width == 2. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s0` - the central sample.

`s1` - the sample to the right of the central sample.

`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if xfrac is out of bounds.

Overrides:

interpolateH in class Interpolation

---

interpolateH

```
public float interpolateH(float s_,
                          float s0,
                          float s1,
                          float s2,
                          float xfrac)
```

Performs horizontal interpolation on a quadruple of floating-point samples. This method may be used instead of the array version for speed. It should only be called if width == 4 and leftPadding == 1. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s_` - the sample to the left of the central sample.

`s0` - the central sample.

`s1` - the sample to the right of the central sample.

`s2` - the sample to the right of s1.

`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if xfrac is out of bounds.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public float interpolateV(float s0,
                          float s1,
                          float yfrac)
```

Performs vertical interpolation on a pair of floating-point samples. This method may be used instead of the array version for speed. It should only be called if height == 2 and topPadding == 0. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:

interpolateV in class Interpolation

---

interpolateV

```
public float interpolateV(float s_,
                          float s0,
                          float s1,
                          float s2,
                          float yfrac)
```

Performs vertical interpolation on a quadruple of floating-point samples. This method may be used instead of the array version for speed. It should only be called if height == 4 and topPadding == 1. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:

interpolateV in class Interpolation

---

interpolate

```
public float interpolate(float s00,
                         float s01,
                         float s10,
                         float s11,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 2x2 grid of floating-point samples. It should only be called if width == height == 2 and leftPadding == topPadding == 0. If either xfrac or yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:

interpolate in class Interpolation

---

interpolate

```
public float interpolate(float s__,
                         float s_0,
                         float s_1,
                         float s_2,
                         float s0_,
                         float s00,
                         float s01,
                         float s02,
                         float s1_,
                         float s10,
                         float s11,
                         float s12,
                         float s2_,
                         float s20,
                         float s21,
                         float s22,
                         float xfrac,
                         float yfrac)
```

Performs interpolation on a 4x4 grid of floating-point samples. It should only be called if width == height == 4 and leftPadding == topPadding == 1. If either xfrac or yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a float.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:

interpolate in class Interpolation

---

interpolateH

```
public double interpolateH(double[] samples,
                           float xfrac)
```

Performs horizontal interpolation on a one-dimensional array of double samples representing a row of samples. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`samples` - an array of doubles.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

Throws:

ArrayIndexOutOfBoundsException - if xfrac is out of bounds.

Overrides:

interpolateH in class Interpolation

---

interpolateV

```
public double interpolateV(double[] samples,
                           float yfrac)
```

Performs vertical interpolation on a one-dimensional array of double samples representing a column of samples. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`samples` - an array of doubles.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a double.
Throws:
ArrayIndexOutOfBoundsException - if yfrac is out of bounds.
Overrides:
interpolateV in class Interpolation

---

interpolateH

```
public double interpolateH(double s0,
                           double s1,
                           float xfrac)
```

Performs horizontal interpolation on a pair of double samples. This method may be used instead of the array version for speed. It should only be called if width == 2. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.
Parameters:
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Throws:
ArrayIndexOutOfBoundsException - if xfrac is out of bounds.
Overrides:
interpolateH in class Interpolation

---

interpolateH

```
public double interpolateH(double s_,
                           double s0,
                           double s1,
                           double s2,
                           float xfrac)
```

Performs horizontal interpolation on a quadruple of double samples. This method may be used instead of the array version for speed. It should only be called if width == 4 and leftPadding == 1. If xfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.
Parameters:
`s_` - the sample to the left of the central sample.
`s0` - the central sample.
`s1` - the sample to the right of the central sample.
`s2` - the sample to the right of s1.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Throws:
ArrayIndexOutOfBoundsException - if xfrac is out of bounds.
Overrides:
interpolateH in class Interpolation

---

interpolateV

```
public double interpolateV(double s0,
                           double s1,
                           float yfrac)
```

Performs vertical interpolation on a pair of double samples. This method may be used instead of the array version for speed. It should only be called if height == 2 and topPadding == 0. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.
Parameters:
`s0` - the central sample.
`s1` - the sample below the central sample.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).
Returns:
the interpolated value as a double.
Throws:
ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:
interpolateV in class Interpolation

---

interpolateV

```
public double interpolateV(double s_,
                           double s0,
                           double s1,
                           double s2,
                           float yfrac)
```

Performs vertical interpolation on a quadruple of double samples. This method may be used instead of the array version for speed. It should only be called if height == 4 and topPadding == 1. If yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:
`s_` - the sample above the central sample.
`s0` - the central sample.
`s1` - the sample below the central sample.
`s2` - the sample below s1.
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a double.

Throws:
ArrayIndexOutOfBoundsException - if yfrac is out of bounds.

Overrides:
interpolateV in class Interpolation

---

interpolate

```
public double interpolate(double s00,
                          double s01,
                          double s10,
                          double s11,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 2x2 grid of double samples. It should only be called if width == height == 2 and leftPadding == topPadding == 0. If either xfrac or yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:
the interpolated value as a double.

Throws:
ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:
interpolate in class Interpolation

---

interpolate

```
public double interpolate(double s__,
                          double s_0,
                          double s_1,
                          double s_2,
                          double s0_,
                          double s00,
                          double s01,
                          double s02,
                          double s1_,
                          double s10,
                          double s11,
                          double s12,
                          double s2_,
                          double s20,
                          double s21,
                          double s22,
                          float xfrac,
                          float yfrac)
```

Performs interpolation on a 4x4 grid of double samples. It should only be called if width == height == 4 and leftPadding == topPadding == 1. If either xfrac or yfrac does not lie between the range [0.0, 1.0F), an ArrayIndexOutOfBoundsException may occur.

Parameters:

`s__` - the sample above and to the left of the central sample.
`s_0` - the sample above the central sample.
`s_1` - the sample above and one to the right of the central sample.
`s_2` - the sample above and two to the right of the central sample.
`s0_` - the sample to the left of the central sample.
`s00` - the central sample.
`s01` - the sample to the right of the central sample.
`s02` - the sample two to the right of the central sample.
`s1_` - the sample below and one to the left of the central sample.
`s10` - the sample below the central sample.
`s11` - the sample below and one to the right of the central sample.
`s12` - the sample below and two to the right of the central sample.
`s2_` - the sample two below and one to the left of the central sample.
`s20` - the sample two below the central sample.
`s21` - the sample two below and one to the right of the central sample.
`s22` - the sample two below and two to the right of the central sample.
`xfrac` - the X subsample position, in the range [0.0F, 1.0F).
`yfrac` - the Y subsample position, in the range [0.0F, 1.0F).

Returns:

the interpolated value as a double.

Throws:

ArrayIndexOutOfBoundsException - if xfrac or yfrac are out of bounds.

Overrides:

interpolate in class Interpolation

javax.media.jai
Class JAI.RenderingKey

```
java.lang.Object
  |
  +--java.awt.RenderingHints.Key
        |
        +--javax.media.jai.JAI.RenderingKey
```

---

static class **JAI.RenderingKey**
extends java.awt.RenderingHints.Key

Rendering hints.

Field Detail

objectClass

```
private java.lang.Class objectClass
```

Constructor Detail

JAI.RenderingKey

```
JAI.RenderingKey(int privateKey,
                 java.lang.Class objectClass)
```

Method Detail

isCompatibleValue

```
public boolean isCompatibleValue(java.lang.Object val)
```

Overrides:
isCompatibleValue in class java.awt.RenderingHints.Key

javax.media.jai

Class JAI

```
java.lang.Object
  |
  +--javax.media.jai.JAI
```

public final class **JAI**
extends java.lang.Object

A convenience class for instantiating operations.

This class allows programmers to use the syntax:

```
 import javax.media.jai.JAI;
 RenderedOp im = JAI.create("convolve", paramBlock, renderHints);
```

to create new images or collections by applying operators. The `create()` method returns a `RenderedOp` encapsulating the operation name, parameter block, and rendering hints. Additionally, it performs validity checking on the operation parameters.

If the `OperationDescriptor` associated with the named operation returns `true` from its `isImmediate()` method, the `JAI.createNS()` method will ask the `RenderedOp` it constructs to render itself immediately. If this rendering is `null`, `createNS()` will itself return `null` rather that returning an instance of `RenderedOp` as it normally does.

It is possible to create new instances of the JAI class in order to control each instance's registry and tile scheduler individually. Most users will want to use only the static methods of this class, which perform all operations on a default instance, which in turn makes use of a default registry. To create a new image or collection on a non-default `JAI` instance, the `createNS()` and `createCollectionNS` (NS being short for "non-static") methods are used.

The `JAI` class contains convenience methods for a number of common argument list formats. These methods perform the work of constructing a `ParameterBlock` automatically. The convenience methods are available only in `static` form and make use of the default instance. When operating with a specific instance, the general, non-static functions `createNS()` and `createCollectionNS()` should be used. All of the convenience methods operate by calling `createNS()` on the default `JAI` instance, and thus inherit the semantics of that method with regard to immediate rendering.

The registry being used by a particular instance may be inspected or set using the `getOperationRegistry()` and `setOperationRegistry()` methods. Only experienced users should attempt to set the registry.

The `TileCache` and `TileScheduler` associated with an instance may be similarly accessed.

Each instance of `JAI` contains a set of rendering hints which will be used for all image or collection creations. These hints are merged with any hints supplied to the `create` method; directly supplied hints take precedence over the common hints. When a new `JAI` instance is constructed, its hints are initialized to a copy of the hints associated with the default instance. The hints associated with any instance, including the default instance, may be manipulated using the `getRenderingHints()`, `setRenderingHints()`,

Field Detail

HINT_IMAGE_LAYOUT

`private static final int` **`HINT_IMAGE_LAYOUT`**

HINT_INTERPOLATION

`private static final int` **`HINT_INTERPOLATION`**

HINT_OPERATION_REGISTRY

`private static final int` **`HINT_OPERATION_REGISTRY`**

HINT_OPERATION_BOUND

`private static final int` **`HINT_OPERATION_BOUND`**

HINT_BORDER_EXTENDER

`private static final int` **`HINT_BORDER_EXTENDER`**

HINT_TILE_CACHE

`private static final int` **`HINT_TILE_CACHE`**

KEY_IMAGE_LAYOUT

`public static java.awt.RenderingHints.Key` **`KEY_IMAGE_LAYOUT`**

Key for `ImageLayout` object values.

KEY_INTERPOLATION

`public static java.awt.RenderingHints.Key` **`KEY_INTERPOLATION`**

Key for `Interpolation` object values.

KEY_OPERATION_REGISTRY

`public static java.awt.RenderingHints.Key` **`KEY_OPERATION_REGISTRY`**

Key for `OperationRegistry` object values.

KEY_OPERATION_BOUND

`public static java.awt.RenderingHints.Key` **`KEY_OPERATION_BOUND`**

Key for `Integer` object values representing whether the operation is compute, network, or I/O bound. The values come from the constants `OpImage.OP_COMPUTE_BOUND`, `OpImage.OP_IO_BOUND`, and `OpImage.OP_NETWORK_BOUND`.

KEY_BORDER_EXTENDER

`public static java.awt.RenderingHints.Key` **`KEY_BORDER_EXTENDER`**

Key for `BorderExtender` object values.

KEY_TILE_CACHE

`public static java.awt.RenderingHints.Key` **`KEY_TILE_CACHE`**

Key for `TileCache` object values.

---

operationRegistry

`private OperationRegistry` **`operationRegistry`**

---

tileScheduler

`private TileScheduler` **`tileScheduler`**

---

tileCache

`private TileCache` **`tileCache`**

---

renderingHints

`private java.awt.RenderingHints` **`renderingHints`**

---

defaultInstance

`private static JAI` **`defaultInstance`**

Constructor Detail

JAI

```
private JAI(OperationRegistry operationRegistry,
            TileScheduler tileScheduler,
            TileCache tileCache,
            java.awt.RenderingHints renderingHints)
```

Returns a new instance of the JAI class.

JAI

```
public JAI()
```

Returns a new instance of the JAI class. The `OperationRegistry`, `TileScheduler`, and `TileCache` will initially be references to those of the default instance. The rendering hints will be set to a clone of those of the default instance.

Method Detail

getDefaultInstance

```
public static JAI getDefaultInstance()
```

Returns the default JAI instance. This instance is used by all of the static methods of this class.

---

getOperationRegistry

```
public OperationRegistry getOperationRegistry()
```

Returns the OperationRegistry being used by this JAI instance.

---

setOperationRegistry

```
public void setOperationRegistry(OperationRegistry operationRegistry)
```

Sets the OperationRegistry to be used by this JAI instance.

---

getTileScheduler

```
public TileScheduler getTileScheduler()
```

Returns the TileScheduler being used by this JAI instance.

---

setTileScheduler

```
public void setTileScheduler(TileScheduler tileScheduler)
```

Sets the TileScheduler to be used by this JAI instance.

---

getTileCache

```
public TileCache getTileCache()
```

Returns the TileCache being used by this JAI instance.

---

setTileCache

```
public void setTileCache(TileCache tileCache)
```

Sets the TileCache to be used by this JAI instance. The `tileCache` parameter will be added to the `RenderingHints` of this `JAI` instance.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.renderable.ParameterBlock args,
                                java.awt.RenderingHints hints)
```

Creates a `RenderedOp` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`, and applying the specified hints to the destination. This method should only be used when the final result returned is a single `RenderedImage`.

The default JAI instance is used as the source of the registry and tile scheduler; that is, this method is equivalent to `getDefaultInstance().createNS(opName, args, hints)`. The functionality of this method is the same as its corresponding non-static method `createNS()`.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.
`hints` - The hints for the operation.

Returns:

A `RenderedOp` that represents the named operation, or `null` if the specified operation is in the "immediate" mode and the rendering of the `PlanarImage` failed.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the default operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the default operation registry does not support rendered image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.RenderedImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createNS

```
public RenderedOp createNS(java.lang.String opName,
                           java.awt.image.renderable.ParameterBlock args,
                           java.awt.RenderingHints hints)
```

Creates a `RenderedOp` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`, and applying the specified hints to the destination. This method should only be used when the final result returned is a single `RenderedImage`. However, the source(s) supplied may be a collection of rendered images or a collection of collections that at the very basic level include rendered images.

The supplied operation name is validated against the operation registry. The source(s) and/or parameter(s) in the `ParameterBlock` are validated against the named operation's descriptor, both in their numbers and types. Additional restrictions placed on the sources and parameters by an individual operation are also validated by calling its `OperationDescriptor.validateArguments()` method.

JAI allows a parameter to have a `null` input value, if that particular parameter has a default value specified in its operation's descriptor. In this case, the default value will replace the `null` input to be used by its `OpImage`.

JAI also allows unspecified tailing parameters, if these parameters have default values specified in the operation's descriptor. In this case, the default values again are used by the `OpImage`. However, if a parameter, which has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the `ParameterBlock`, even if it only has a value of code>null.

The rendering hints associated with this instance of `JAI` are overlaid with the hints passed to this method. That is, the set of keys will be the union of the keys from the instance's hints and the hints parameter. If the same key exists in both places, the value from the hints parameter will be used.

This version of `create` is non-static; it may be used with a specific instance of the JAI class. All of the static `create()` methods ultimately call this method, thus inheriting this method's error handling.

Since this method performs parameter checking, it may not be suitable for creating `RenderedOp` nodes meant to be passed to another host using the `RemoteImage` interface. For example, it might be necessary to refer to a file that is present only on the remote host. In such cases, it is possible to instantiate a `RenderedOp` directly, avoiding all checks.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.
`hints` - The hints for the operation.

Returns:

A `RenderedOp` that represents the named operation, or `null` if the specified operation is in the "immediate" mode and the rendering of the `PlanarImage` failed.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the current operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the current operation registry does not support rendered image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.RenderedImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified

in `args`.

---

createCollection

```
public static java.util.Collection createCollection(java.lang.String opName,
                                                    java.awt.image.renderable.ParameterBlock args,
                                                    java.awt.RenderingHints hints)
```

Creates a `Collection` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`, and applying the specified hints to the destination. This method should only be used when the final result returned is a `Collection`. (This includes `javax.media.jai.CollectionOp`s.)

The default JAI instance is used as the source of the registry and tile scheduler; that is, this method is equivalent to `getDefaultInstance().createCollectionNS(opName, args, hints)`. The functionality of this method is the same as its corresponding non-static method `createCollectionNS()`.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.
`hints` - The hints for the operation.

Returns:

A `Collection` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the default operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the default operation registry does not support rendered image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.RenderedImage` or a `javax.media.jai.CollectionImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createCollectionNS

```
public java.util.Collection createCollectionNS(java.lang.String opName,
                                               java.awt.image.renderable.ParameterBlock args,
                                               java.awt.RenderingHints hints)
```

Creates a `Collection` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`, and applying the specified hints to the destination. This method should only be used when the final result returned is a `Collection`. (This includes `javax.media.jai.CollectionOp`s.) The source(s) supplied may be a collection of rendered images or a collection of collections that at the very basic level include rendered images.

This method should be used to create a `Collection` in the rendered image mode.

The supplied operation name is validated against the operation registry. The source(s) and/or parameter(s) in the `ParameterBlock` are validated against the named operation's descriptor, both in their numbers and types. Additional restrictions placed on the sources and parameters by an individual operation are also validated by calling its `OperationDescriptor.validateArguments()` method.

JAI allows a parameter to have a `null` input value, if that particular parameter has a default value specified in its operation's descriptor. In this case, the default value will replace the `null` input to be used by its `OpImage`.

JAI also allows unspecified tailing parameters, if these parameters have default values specified in the operation's descriptor. In this case, the default values again are used by the `OpImage`. However, if a parameter, which has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the `ParameterBlock`, even if it only has a value of code>null.

The rendering hints associated with this instance of `JAI` are overlaid with the hints passed to this method. That is, the set of keys will be the union of the keys from the instance's hints and the hints parameter. If the same key exists in both places, the value from the hints parameter will be used.

This version of `createCollection` is non-static; it may be used with a specific instance of the JAI class.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.
`hints` - The hints for the operation.

Returns:

A `Collection` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation

name in the current operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the current operation registry does not support rendered image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.RenderedImage` or a `javax.media.jai.CollectionImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createTileCache

```
public static TileCache createTileCache(int tileCapacity,
                                        long memCapacity)
```

Constructs a `TileCache` with the given tile capacity in tiles and memory capacity in bytes. Users may supply an instance of `TileCache` to an operation by supplying a `RenderingHint` with a JAI.KEY_TILE_CACHE key and the desired `TileCache` instance as its value. Note that the absence of a tile cache hint will result in the use of the `TileCache` belonging to the default JAI instance. To force an operation not to perform caching, a `TileCache` instance with a tile capacity of 0 may be used. An exception will be thrown if either tileCapacity or memCapacity is negative. Attempting to set either value larger than the JVM size may result in an OutOfMemory exception.

---

createTileCache

```
public static TileCache createTileCache()
```

Constructs a `TileCache` with the default tile capacity in tiles and memory capacity in bytes. Users may supply an instance of `TileCache` to an operation by supplying a `RenderingHint` with a JAI.KEY_TILE_CACHE key and the desired `TileCache` instance as its value. Note that the absence of a tile cache hint will result in the use of the `TileCache` belonging to the default JAI instance. To force an operation not to perform caching, a `TileCache` instance with a tile capacity of 0 may be used.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.renderable.ParameterBlock args)
```

Creates a `RenderedOp` with `null` rendering hints.
Parameters:
`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param)
```

Creates a RenderedOp that takes 1 object parameter.
Parameters:
`opName` - The name of the operation.
`param` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param1,
                                java.lang.Object param2)
```

Creates a RenderedOp that takes 2 object parameters.
Parameters:
`opName` - The name of the operation.
`param1` - The first object parameter.
`param2` - The second object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param1,
                                int param2)
```

Creates a RenderedOp that takes 1 object parameter and 1 int parameter
Parameters:
`opName` - The name of the operation.
`param1` - The object parameter.
`param2` - The int parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3)
```

Creates a RenderedOp that takes 3 object parameters.
Parameters:
`opName` - The name of the operation.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                int param1,
                                int param2,
                                java.lang.Object param3)
```

Creates a RenderedOp that takes 2 int parameters and one object parameter
Parameters:
`opName` - The name of the operation.
`param1` - The first int parameter.
`param2` - The second int parameter.
`param3` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3,
                                java.lang.Object param4)
```

Creates a RenderedOp that takes 4 object parameters.
Parameters:
`opName` - The name of the operation.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.
`param4` - The fourth object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.lang.Object param1,
                                int param2,
                                java.lang.Object param3,
                                int param4)
```

Creates a RenderedOp that takes 2 object and 2 int parameters.
Parameters:
`opName` - The name of the operation.
`param1` - The first object parameter.
`param2` - The first int parameter.
`param3` - The second object parameter.
`param4` - The second int parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src)
```

Creates a RenderedOp that takes 1 RenderedImage source.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.util.Collection srcCol)
```

Creates a RenderedOp that takes 1 Collection source.

Parameters:

`opName` - The name of the operation.
`srcCol` - The Collection src parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param)
```

Creates a RenderedOp that takes 1 RenderedImage source and 1 object parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                int param)
```

Creates a RenderedOp that takes 1 RenderedImage source and 1 int parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param` - The int parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2)
```

Creates a RenderedOp that takes 1 RenderedImage source and 2 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                float param2)
```

Creates a RenderedOp that takes 1 RenderedImage source, 1 object and 1 float parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The object parameter.
`param2` - The float parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3)
```

Creates a RenderedOp that takes 1 RenderedImage source and 3 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                int param2,
                                int param3)
```

Creates a RenderedOp that takes 1 RenderedImage source, 1 object and 2 int parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The object parameter.
`param2` - The first int parameter.
`param3` - The second int parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                float param1,
                                float param2,
                                java.lang.Object param3)
```

Creates a RenderedOp that takes 1 RenderedImage source, 2 float and 1 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first float parameter.
`param2` - The second float parameter.
`param3` - The object parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3,
                                java.lang.Object param4)
```

Creates a RenderedOp that takes 1 RenderedImage source and 4 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.
`param4` - The fourth object parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                int param3,
                                int param4)
```

Creates a RenderedOp that takes 1 RenderedImage source and 2 object parameters and 2 in parameters

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The first int parameter.
`param4` - The second int parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                int param1,
                                int param2,
                                int param3,
                                int param4)
```

Creates a RenderedOp that takes 1 RenderedImage source and 4 int parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first int parameter.
`param2` - The second int parameter.
`param3` - The third int parameter.
`param4` - The fourth int parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                float param1,
                                float param2,
                                float param3,
                                java.lang.Object param4)
```

Creates a RenderedOp that takes 1 RenderedImage source, 3 float and 1 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first float parameter.
`param2` - The second float parameter.
`param3` - The third float parameter.
`param4` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3,
                                java.lang.Object param4,
                                java.lang.Object param5)
```

Creates a RenderedOp that takes 1 RenderedImage source and 5 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.
`param4` - The fourth object parameter.
`param5` - The fifth object parameter.

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                float param1,
                                float param2,
                                float param3,
                                float param4,
                                java.lang.Object param5)
```

Creates a RenderedOp that takes 1 RenderedImage source, 4 float parameters and one object parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first float parameter.
`param2` - The second float parameter.
`param3` - The third float parameter.
`param4` - The fourth float parameter.
`param5` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                float param1,
                                int param2,
                                float param3,
                                float param4,
                                java.lang.Object param5)
```

Creates a RenderedOp that takes 1 RenderedImage source, 3 float parameters, 1 int parameter and 1 object parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first float parameter.
`param2` - The int parameter.
`param3` - The second float parameter.
`param4` - The third float parameter.
`param5` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                java.lang.Object param1,
                                java.lang.Object param2,
                                java.lang.Object param3,
                                java.lang.Object param4,
                                java.lang.Object param5,
                                java.lang.Object param6)
```

Creates a RenderedOp that takes 1 RenderedImage source and 6 object parameters.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.
`param4` - The fourth object parameter.
`param5` - The fifth object parameter.
`param6` - The sixth object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                java.awt.image.RenderedImage src,
                                int param1,
                                int param2,
                                int param3,
                                int param4,
                                int param5,
                                java.lang.Object param6)
```

Creates a RenderedOp that takes 1 RenderedImage source, 5 int parameters and 1 object parameter.

Parameters:

`opName` - The name of the operation.
`src` - The RenderedImage src parameter.
`param1` - The first int parameter.
`param2` - The second int parameter.
`param3` - The third int parameter.
`param4` - The fourth int parameter.
`param5` - The fifth int parameter.
`param6` - The object parameter.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                       java.awt.image.RenderedImage src1,
                                       java.awt.image.RenderedImage src2)
```

Creates a RenderedOp that takes 2 RenderedImage sources.

Parameters:

`opName` - The name of the operation.
`src1` - The first RenderedImage src.
`src2` - The second RenderedImage src.

---

create

```
public static RenderedOp create(java.lang.String opName,
                                       java.awt.image.RenderedImage src1,
                                       java.awt.image.RenderedImage src2,
                                       java.lang.Object param1,
                                       java.lang.Object param2,
                                       java.lang.Object param3,
                                       java.lang.Object param4)
```

Creates a RenderedOp that takes 2 RenderedImage sources and 4 object parameters.

Parameters:

`opName` - The name of the operation.
`src1` - The first RenderedImage src.
`src2` - The second RenderedImage src.
`param1` - The first object parameter.
`param2` - The second object parameter.
`param3` - The third object parameter.
`param4` - The fourth object parameter.

---

createCollection

```
public static java.util.Collection createCollection(java.lang.String opName,
                                                           java.awt.image.renderable.ParameterBlock args)
```

Creates a `Collection` with `null` rendering hints.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

---

createRenderable

```
public static RenderableOp createRenderable(java.lang.String opName,
                                                   java.awt.image.renderable.ParameterBlock args)
```

Creates a `RenderableOp` that represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`. This method should only be used when the final result returned is a single `RenderdableImage`.

The default JAI instance is used as the source of the registry and tile scheduler; that is, this method is equivalent to `getDefaultInstance().createRenderableNS(opName, args)`. The functionality of this method is the same as its corresponding non-static method `createRenderableNS()`.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

Returns:

A `RenderableOp` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the default operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the default operation registry does not support renderable image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.renderable.RenderableImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createRenderableNS

```
public RenderableOp createRenderableNS(java.lang.String opName,
                                       java.awt.image.renderable.ParameterBlock args)
```

Creates a `RenderableOp` that represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`. This method should only be used when the final result returned is a single `RenderableImage`. However, the source(s) supplied may be a collection of renderable images or a collection of collections that at the very basic level include renderable images.

The supplied operation name is validated against the operation registry. The source(s) and/or parameter(s) in the `ParameterBlock` are validated against the named operation's descriptor, both in their numbers and types. Additional restrictions placed on the sources and parameters by an individual operation are also validated by calling its `OperationDescriptor.validateRenderableArguments()` method.

JAI allows a parameter to have a `null` input value, if that particular parameter has a default value specified in its operation's descriptor. In this case, the default value will replace the `null` input to be used by its `OpImage`.

JAI also allows unspecified tailing parameters, if these parameters have default values specified in the operation's descriptor. In this case, the default values again are used by the `OpImage`. However, if a parameter, which has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the `ParameterBlock`, even if it only has a value of code>null.

This version of the "createRenderable" is non-static; it may be used with a specific instance of the JAI class.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

Returns:

A `RenderableOp` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the current operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the current operation registry does not support renderable image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.renderable.RenderableImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createRenderableCollection

```
public static java.util.Collection createRenderableCollection(java.lang.String opName,
                                                              java.awt.image.renderable.ParameterBlock args)
```

Creates a `Collection` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`. This method should only be used when the final result returned is a `Collection`. (This includes `javax.media.jai.CollectionOp`s.)

The default JAI instance is used as the source of the registry and tile scheduler; that is, this method is equivalent to `getDefaultInstance().createRenderableCollectionNS(opName, args)`. The functionality of this method is the same as its corresponding non-static method `createRenderableCollectionNS()`.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

Returns:

A `Collection` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the default operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the default operation registry does not support renderable image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.renderable.RenderableImage` or a `javax.media.jai.CollectionImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

createRenderableCollectionNS

```
public java.util.Collection createRenderableCollectionNS(java.lang.String opName,
                                                         java.awt.image.renderable.ParameterBlock args)
```

Creates a `Collection` which represents the named operation, using the source(s) and/or parameter(s) specified in the `ParameterBlock`. This method should only be used when the final result returned is a `Collection`. (This includes `javax.media.jai.CollectionOp`s.) The source(s) supplied may be a collection of renderable images or a collection of collections that at the very basic level include renderable images.

This method should be used to create a `Collection` in the renderable image mode.

The supplied operation name is validated against the operation registry. The source(s) and/or parameter(s) in the `ParameterBlock` are validated against the named operation's descriptor, both in their numbers and types. Additional restrictions placed on the sources and parameters by an individual operation are also validated by calling its `OperationDescriptor.validateRenderableArguments()` method.

JAI allows a parameter to have a `null` input value, if that particular parameter has a default value specified in its operation's descriptor. In this case, the default value will replace the `null` input to be used by its `OpImage`.

JAI also allows unspecified tailing parameters, if these parameters have default values specified in the operation's descriptor. In this case, the default values again are used by the `OpImage`. However, if a parameter, which has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the `ParameterBlock`, even if it only has a value of code>null.

This version of `createRenderableCollection` is non-static; it may be used with a specific instance of the JAI class.

Parameters:

`opName` - The name of the operation.
`args` - The source(s) and/or parameter(s) for the operation.

Returns:

A `Collection` that represents the named operation.

Throws:

NullPointerException - if `opName` is `null`.
NullPointerException - if `args` is `null`.
java.lang.IllegalArgumentException - if no `OperationDescriptor` is registered under the specified operation name in the current operation registry.
java.lang.IllegalArgumentException - if the `OperationDescriptor` registered under the specified operation name in the current operation registry does not support renderable image mode.
java.lang.IllegalArgumentException - if the specified operation does not produce a `java.awt.image.renderable.RenderableImage` or a `javax.media.jai.CollectionImage`.
java.lang.IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in `args`.

---

getRenderingHints

```
public java.awt.RenderingHints getRenderingHints()
```

Returns the `RenderingHints` associated with this `JAI` instance. These rendering hints will be merged with any hints supplied as an argument to the `createNS()` method.

---

setRenderingHints

```
public void setRenderingHints(java.awt.RenderingHints hints)
```

Sets the `RenderingHints` associated with this `JAI` instance. These rendering hints will be merged with any hints supplied as an argument to the `createNS()` method.

The `hints` argument must be non-null, otherwise a `NullPointerException` will be thrown.

---

clearRenderingHints

```
public void clearRenderingHints()
```

Clears the `RenderingHints` associated with this `JAI` instance.

---

getRenderingHint

```
public java.lang.Object getRenderingHint(java.awt.RenderingHints.Key key)
```

Returns the hint value associated with a given key in this `JAI` instance, or `null` if no value is associated with the given key.

Throws:

java.lang.IllegalArgumentException - if `key` is `null`.

---

setRenderingHint

```
public void setRenderingHint(java.awt.RenderingHints.Key key,
                             java.lang.Object value)
```

Sets the hint value associated with a given key in this `JAI` instance.

Throws:

java.lang.IllegalArgumentException - if `key` is `null`.
java.lang.IllegalArgumentException - if `value` is `null`.
java.lang.IllegalArgumentException - if `value` is not of the correct type for the given hint.

---

removeRenderingHint

```
public void removeRenderingHint(java.awt.RenderingHints.Key key)
```

Removes the hint value associated with a given key in this `JAI` instance.

javax.media.jai
Class JaiI18N

```
java.lang.Object
  |
  +--javax.media.jai.JaiI18N
```

class **JaiI18N**
extends java.lang.Object

Field Detail

packageName

`static java.lang.String packageName`

Constructor Detail

JaiI18N

`JaiI18N()`

Method Detail

getString

`public static java.lang.String getString(java.lang.String key)`

javax.media.jai

Class KernelJAI

```
java.lang.Object
  |
  +--javax.media.jai.KernelJAI
```

public class **KernelJAI**
extends java.lang.Object
implements java.io.Serializable

A kernel, used by the Convolve, Ordered Dither, and Error Diffusion operations.

This class is used as an auxiliary class to perform a Convolve, Ordered Dither, or Error Diffusion operation on an image. In the latter two operations the kernel is referred to as a "dither mask" or "error filter", respectively, rather than as a kernel.

A `KernelJAI` is characterized by its width, height, and origin, or key element. The key element is the element which is placed over the current source pixel to perform convolution or error diffusion. In the case of ordered dithering an array of KernelJAI objects is actually required with there being one `KernelJAI` per band of the image to be dithered. For ordered dithering the location of the key element is in fact irrelevant.

See Also:
`ConvolveDescriptor`, `OrderedDitherDescriptor`, `ErrorDiffusionDescriptor`

Field Detail

ERROR_FILTER_FLOYD_STEINBERG

`public static final KernelJAI` **`ERROR_FILTER_FLOYD_STEINBERG`**

Floyd and Steinberg error filter (1975).

```
 (1/16 x)  [   * 7 ]
           [ 3 5 1 ]
```

ERROR_FILTER_JARVIS

`public static final KernelJAI` **`ERROR_FILTER_JARVIS`**

Jarvis, Judice, and Ninke error filter (1976).

```
           [     * 7 5 ]
 (1/48 x)  [ 3 5 7 5 3 ]
           [ 1 3 5 3 1 ]
```

ERROR_FILTER_STUCKI

`public static final KernelJAI` **`ERROR_FILTER_STUCKI`**

Stucki error filter (1981).

```
           [     * 7 5 ]
 (1/42 x)  [ 2 4 8 4 2 ]
           [ 1 2 4 2 1 ]
```

DITHER_MASK_441

`public static final KernelJAI[]` **`DITHER_MASK_441`**

4x4x1 mask useful for dithering 8-bit grayscale images to 1-bit images.

DITHER_MASK_443

`public static final KernelJAI[]` **`DITHER_MASK_443`**

4x4x3 mask useful for dithering 24-bit color images to 8-bit pseudocolor images.

GRADIENT_MASK_SOBEL_HORIZONTAL

`public static final KernelJAI` **`GRADIENT_MASK_SOBEL_HORIZONTAL`**

Gradient Mask for SOBEL_HORIZONTAL

GRADIENT_MASK_SOBEL_VERTICAL

`public static final KernelJAI` **`GRADIENT_MASK_SOBEL_VERTICAL`**

Gradient Mask for SOBEL_VERTICAL

---

width

`protected int` **`width`**

The width of the kernel.

---

height

`protected int` **`height`**

The height of the kernel.

---

xOrigin

`protected int` **`xOrigin`**

The X coordinate of the key element.

---

yOrigin

`protected int` **`yOrigin`**

The Y coordinate of the key element.

---

data

`protected float[]` **`data`**

The kernel data in row-major format.

---

dataH

`protected float[]` **`dataH`**

The horizontal data for a separable kernel

---

dataV

`protected float[]` **`dataV`**

The vertical data for a separable kernel

---

isSeparable

`protected boolean` **`isSeparable`**

True if the kernel is separable.

---

isHorizontallySymmetric

`protected boolean` **`isHorizontallySymmetric`**

True if the kernel has horizontal (Y axis) symmetry.

---

isVerticallySymmetric

```
protected boolean isVerticallySymmetric
```

True if the kernel has vertical (X axis) symmetry.

---

rotatedKernel

```
protected KernelJAI rotatedKernel
```

Variable to cache a copy of the rotated kernel

FLOAT_ZERO_TOL

```
public static final float FLOAT_ZERO_TOL
```

Constructor Detail

KernelJAI

```
public KernelJAI(int width,
                 int height,
                 int xOrigin,
                 int yOrigin,
                 float[] data)
```

Constructs a KernelJAI with the given parameters. The data array is copied.

Parameters:

`width` - the width of the kernel.
`height` - the height of the kernel.
`xOrigin` - the X coordinate of the key kernel element.
`yOrigin` - the Y coordinate of the key kernel element.
`data` - the float data in row-major format.

Throws:

NullPointerException - if data is null.
java.lang.IllegalArgumentException - if width is not a positive number.
java.lang.IllegalArgumentException - if height is not a positive number.
java.lang.IllegalArgumentException - if kernel data array does not have width * height number of elements.

KernelJAI

```
public KernelJAI(int width,
                 int height,
                 int xOrigin,
                 int yOrigin,
                 float[] dataH,
                 float[] dataV)
```

Constructs a separable KernelJAI from two float arrays. The data arrays are copied.

Parameters:

`width` - the width of the kernel.
`height` - the height of the kernel.
`xOrigin` - the X coordinate of the key kernel element.
`yOrigin` - the Y coordinate of the key kernel element.
`dataH` - the float data for the horizontal direction.
`dataV` - the float data for the vertical direction.

Throws:

NullPointerException - if dataH is null.
NullPointerException - if dataV is null.
java.lang.IllegalArgumentException - if width is not a positive number.
java.lang.IllegalArgumentException - if height is not a positive number.
java.lang.IllegalArgumentException - if dataH does not have width elements.
java.lang.IllegalArgumentException - if dataV does not have height elements.

KernelJAI

```
public KernelJAI(int width,
                 int height,
                 float[] data)
```

Constructs a kernel with the given parameters. The data array is copied. The key element is set to (trunc(width/2), trunc(height/2)).

Parameters:

`width` - the width of the kernel.
`height` - the height of the kernel.
`data` - the float data in row-major format.

Throws:

NullPointerException - if data is null.
java.lang.IllegalArgumentException - if width is not a positive number.
java.lang.IllegalArgumentException - if height is not a positive number.
java.lang.IllegalArgumentException - if data does not have width * height number of elements.

KernelJAI

```
public KernelJAI(java.awt.image.Kernel k)
```

Constructs a KernelJAI from a java.awt.image.Kernel object.

Throws:

NullPointerException - if k is null.

Method Detail

checkSeparable

```
private void checkSeparable()
```

---

classifyKernel

```
private void classifyKernel()
```

---

getWidth

```
public int getWidth()
```

Returns the width of the kernel.

---

getHeight

```
public int getHeight()
```

Returns the height of the kernel.

---

getXOrigin

```
public int getXOrigin()
```

Returns the X coordinate of the key kernel element.

---

getYOrigin

```
public int getYOrigin()
```

Returns the Y coordinate of the key kernel element.

---

getKernelData

```
public float[] getKernelData()
```

Returns a copy of the kernel data in row-major format.

---

getHorizontalKernelData

```
public float[] getHorizontalKernelData()
```

Returns the horizontal portion of the kernel if the kernel is separable, or `null` otherwise. The kernel may be tested for separability by calling `isSeparable()`.

---

getVerticalKernelData

```
public float[] getVerticalKernelData()
```

Returns the vertical portion of the kernel if the kernel is separable, or `null` otherwise. The kernel may be tested for separability by calling `isSeparable()`.

---

getElement

```
public float getElement(int xIndex,
                        int yIndex)
```

Returns a given element of the kernel.

Throws:

ArrayIndexOutOfBoundsException - if either xIndex or yIndex is an invalid index.

---

isSeparable

```
public boolean isSeparable()
```

Returns true if the kernel is separable. NOTE: when separable, there will be two valid vectors

---

isHorizontallySymmetric

```
public boolean isHorizontallySymmetric()
```

Returns true if the kernel has horizontal (Y axis) symmetry.

---

isVerticallySymmetric

```
public boolean isVerticallySymmetric()
```

Returns true if the kernel has vertical (X axis) symmetry.

---

getLeftPadding

```
public int getLeftPadding()
```

Returns the number of pixels required to the left of the key element.

---

getRightPadding

```
public int getRightPadding()
```

Returns the number of pixels required to the right of the key element.

---

getTopPadding

```
public int getTopPadding()
```

Returns the number of pixels required above the key element.

---

getBottomPadding

```
public int getBottomPadding()
```

Returns the number of pixels required below the key element.

---

fAbs

```
private static final float fAbs(float a)
```

Computing the absolute value of a float type

---

getRotatedKernel

```
public KernelJAI getRotatedKernel()
```

Returns a 180 degree rotated version of the kernel. This is needed by most convolve operations to get the correct results. modification on 9/20: make it work for separable kernels. -jxz

Returns:

the rotated kernel.

javax.media.jai

Class LookupTableJAI

```
java.lang.Object
  |
  +--javax.media.jai.LookupTableJAI
```

Direct Known Subclasses:

ColorCube

---

public class **LookupTableJAI**
extends java.lang.Object
implements java.io.Serializable

A lookup table object associated with the "Lookup" operation. The "Lookup" operation is described in `javax.media.jai.operator.LookupDescriptor`.

This object represents a single- or multi-banded table of any JAI supported data types. A single- or multi-banded source image of integral data types is passed through the table and transformed into a single- or multi-banded destination image of both integral and float or double data types.

The table data may cover only a subrange of the legal range of the input data type. The subrange is selected by means of an offset parameter which is to be subtracted from the input value before indexing into the table array. When only a subranged table is used with a source image, it is up to the user to make certain that the source image does not have pixel values outside of the table range. Other wise, the result is undefined, with possible outcomes being an ArrayIndexOutOfBoundsException, segmentation fault, or random results.

The table data is saved by reference only.

See Also:

`LookupDescriptor`

Field Detail

data

`transient java.awt.image.DataBuffer` **`data`**

The table data.

Constructor Detail

LookupTableJAI

```
public LookupTableJAI(byte[] data)
```

Constructs a single-banded byte lookup table. The index offset is 0.

Parameters:

`data` - The single-banded byte data.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(byte[] data,
                      int offset)
```

Constructs a single-banded byte lookup table with an index offset.

Parameters:

`data` - The single-banded byte data.
`offset` - The offset.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(byte[][] data)
```

Constructs a multi-banded byte lookup table. The index offset for each band is 0.

Parameters:

`data` - The multi-banded byte data in [band][index] format.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(byte[][] data,
                      int offset)
```

Constructs a multi-banded byte lookup table where all bands have the same index offset.

Parameters:

`data` - The multi-banded byte data in [band][index] format.
`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(byte[][] data,
                      int[] offsets)
```

Constructs a multi-banded byte lookup table where each band has a different index offset.

Parameters:

`data` - The multi-banded byte data in [band][index] format.
`offsets` - The offsets for the bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(short[] data,
                      boolean isUShort)
```

Constructs a single-banded short or unsigned short lookup table. The index offset is 0.

Parameters:

`data` - The single-banded short data.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(short[] data,
                      int offset,
                      boolean isUShort)
```

Constructs a single-banded short or unsigned short lookup table with an index offset.

Parameters:

`data` - The single-banded short data.
`offset` - The offset.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(short[][] data,
                      boolean isUShort)
```

Constructs a multi-banded short or unsigned short lookup table. The index offset for each band is 0.

Parameters:

`data` - The multi-banded short data in [band][index] format.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(short[][] data,
                      int offset,
                      boolean isUShort)
```

Constructs a multi-banded short or unsigned short lookup table where all bands have the same index offset.

Parameters:
`data` - The multi-banded short data in [band][index] format.
`offset` - The common offset for all bands.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(short[][] data,
                      int[] offsets,
                      boolean isUShort)
```

Constructs a multi-banded short or unsigned short lookup table where each band has a different index offset.

Parameters:
`data` - The multi-banded short data in [band][index] format.
`offsets` - The offsets for the bands.
`isUShort` - True if data type is DataBuffer.TYPE_USHORT; false if data type is DataBuffer.TYPE_SHORT.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(int[] data)
```

Constructs a single-banded int lookup table. The index offset is 0.

Parameters:
`data` - The single-banded int data.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(int[] data,
                      int offset)
```

Constructs a single-banded int lookup table with an index offset.

Parameters:
`data` - The single-banded int data.
`offset` - The offset.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(int[][] data)
```

Constructs a multi-banded int lookup table. The index offset for each band is 0.

Parameters:
`data` - The multi-banded int data in [band][index] format.

Throws:
NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(int[][] data,
                      int offset)
```

Constructs a multi-banded int lookup table where all bands have the same index offset.

Parameters:

`data` - The multi-banded int data in [band][index] format.

`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(int[][] data,
                      int[] offsets)
```

Constructs a multi-banded int lookup table where each band has a different index offset.

Parameters:

`data` - The multi-banded int data in [band][index] format.

`offsets` - The offsets for the bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(float[] data)
```

Constructs a single-banded float lookup table. The index offset is 0.

Parameters:

`data` - The single-banded float data.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(float[] data,
                      int offset)
```

Constructs a single-banded float lookup table with an index offset.

Parameters:

`data` - The single-banded float data.

`offset` - The offset.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(float[][] data)
```

Constructs a multi-banded float lookup table. The index offset for each band is 0.

Parameters:

`data` - The multi-banded float data in [band][index] format.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(float[][] data,
                      int offset)
```

Constructs a multi-banded float lookup table where all bands have the same index offset.

Parameters:

`data` - The multi-banded float data in [band][index] format.

`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(float[][] data,
                      int[] offsets)
```

Constructs a multi-banded float lookup table where each band has a different index offset.

Parameters:

`data` - The multi-banded float data in [band][index] format.

`offsets` - The offsets for the bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(double[] data)
```

Constructs a single-banded double lookup table. The index offset is 0.

Parameters:

`data` - The single-banded double data.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(double[] data,
                      int offset)
```

Constructs a single-banded double lookup table with an index offset.

Parameters:

`data` - The single-banded double data.

`offset` - The offset.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(double[][] data)
```

Constructs a multi-banded double lookup table. The index offset for each band is 0.

Parameters:

`data` - The multi-banded double data in [band][index] format.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(double[][] data,
                      int offset)
```

Constructs a multi-banded double lookup table where all bands have the same index offset.

Parameters:

`data` - The multi-banded double data in [band][index] format.

`offset` - The common offset for all bands.

Throws:

NullPointerException - if data is null.

LookupTableJAI

```
public LookupTableJAI(double[][] data,
                      int[] offsets)
```

Constructs a multi-banded double lookup table where each band has a different index offset.

Parameters:

`data` - The multi-banded double data in [band][index] format.

`offsets` - The offsets for the bands.

Throws:

NullPointerException - if data is null.

Method Detail

getData

`public java.awt.image.DataBuffer **getData**()`

Returns the table data as a DataBuffer.

---

getByteData

`public byte[][] **getByteData**()`

Returns the byte table data in array format, or null if the table’s data type is not byte.

---

getByteData

`public byte[] **getByteData**(int band)`

Returns the byte table data of a specific band in array format, or null if the table’s data type is not byte.

---

getShortData

`public short[][] **getShortData**()`

Returns the short table data in array format, or null if the table’s data type is not short. This includes both signed and unsigned short table data.

---

getShortData

`public short[] **getShortData**(int band)`

Returns the short table data of a specific band in array format, or null if the table’s data type is not short.

---

getIntData

`public int[][] **getIntData**()`

Returns the integer table data in array format, or null if the table’s data type is not int.

---

getIntData

`public int[] **getIntData**(int band)`

Returns the integer table data of a specific band in array format, or null if table’s data type is not int.

---

getFloatData

`public float[][] **getFloatData**()`

Returns the float table data in array format, or null if the table’s data type is not float.

---

getFloatData

`public float[] **getFloatData**(int band)`

Returns the float table data of a specific band in array format, or null if table’s data type is not float.

---

getDoubleData

`public double[][] **getDoubleData**()`

Returns the double table data in array format, or null if the table’s data type is not double.

---

getDoubleData

`public double[] **getDoubleData**(int band)`

Returns the double table data of a specific band in array format, or null if table’s data type is not double.

---

getOffsets

`public int[] getOffsets()`

Returns the index offsets of entry 0 for all bands.

---

getOffset

`public int getOffset()`

Returns the index offset of entry 0 for the default band.

---

getOffset

`public int getOffset(int band)`

Returns the index offset of entry 0 for a specific band.

---

getNumBands

`public int getNumBands()`

Returns the number of bands of the table.

---

getNumEntries

`public int getNumEntries()`

Returns the number of entries per band of the table.

---

getDataType

`public int getDataType()`

Returns the data type of the table data.

---

getDestNumBands

`public int getDestNumBands(int srcNumBands)`

Returns the number of bands of the destination image, based on the number of bands of the source image and lookup table.

Parameters:

`srcNumBands` - The number of bands of the source image.

Returns:

the number of bands in destination image.

---

getDestSampleModel

`public java.awt.image.SampleModel getDestSampleModel(java.awt.image.SampleModel srcSampleModel)`

Returns a `SampleModel` suitable for holding the output of a lookup operation on the source data described by a given SampleModel with this table. The width and height of the destination SampleModel are the same as that of the source. This method will return null if the source SampleModel has a non-integral data type.

Parameters:

`srcSampleModel` - The SampleModel of the source image.

Returns:

sampleModel suitable for the destination image.

Throws:

NullPointerException - if srcSampleModel is null.

---

getDestSampleModel

```
public java.awt.image.SampleModel getDestSampleModel(java.awt.image.SampleModel srcSampleModel,
                                                     int width,
                                                     int height)
```

Returns a `SampleModel` suitable for holding the output of a lookup operation on the source data described by a given SampleModel with this table. This method will return null if the source SampleModel has a non-integral data type.

Parameters:

`srcSampleModel` - The SampleModel of the source image.

`width` - The width of the destination SampleModel.

`height` - The height of the destination SampleModel.
Returns:
sampleModel suitable for the destination image.
Throws:
NullPointerException - if srcSampleModel is null.

---

isIntegralDataType

```
public boolean isIntegralDataType(java.awt.image.SampleModel sampleModel)
```

Validates data type. Returns true if it's one of the integral data types; false otherwise.
Throws:
NullPointerException - if sampleModel is null.

---

isIntegralDataType

```
public boolean isIntegralDataType(int dataType)
```

Returns `true` if the specified data type is an integral data type, such as byte, ushort, short, or int.

---

lookup

```
public int lookup(int band,
                  int value)
```

Performs lookup on a given value belonging to a given source band, and returns the result as an int.
Parameters:
`band` - The source band the value is from.
`value` - The source value to be placed through the lookup table.

---

lookupFloat

```
public float lookupFloat(int band,
                         int value)
```

Performs lookup on a given value belonging to a given source band, and returns the result as a float.
Parameters:
`band` - The source band the value is from.
`value` - The source value to be placed through the lookup table.

---

lookupDouble

```
public double lookupDouble(int band,
                           int value)
```

Performs lookup on a given value belonging to a given source band, and returns the result as a double.
Parameters:
`band` - The source band the value is from.
`value` - The source value to be placed through the lookup table.

---

lookup

```
public java.awt.image.WritableRaster lookup(java.awt.image.WritableRaster src)
```

Performs table lookup in place on a given WritableRaster. The The lookup operation must preserve the data type and SampleModel of the source. A reference to the supplied WritableRaster will be returned.
Throws:
NullPointerException - if the source is null.
java.lang.IllegalArgumentException - if the source's SampleModel is not of integral type.
java.lang.IllegalArgumentException - if the lookup operation would result in a change in the data type or number of bands of the Raster.

---

lookup

```
public java.awt.image.WritableRaster lookup(java.awt.image.Raster src,
                                            java.awt.image.WritableRaster dst,
                                            java.awt.Rectangle rect)
```

Performs table lookup on a source Raster, writing the result into a supplied WritableRaster. The destination must have a data type and SampleModel appropriate to the results of the lookup operation. The table lookup operation is performed within a specified rectangle.

The `dst` argument may be null, in which case a new WritableRaster is created using the appropriate SampleModel.

The rectangle of interest may be null, in which case the operation will be performed on the intersection of the source and destination bounding rectangles.

Parameters:

`src` - A Raster containing the source pixel data.
`dst` - The WritableRaster to be computed, or null. If supplied, its data type and number of bands must be suitable for the source and lookup table.
`rect` - The rectangle within the tile to be computed. If rect is null, the intersection of the source and destination bounds will be used. Otherwise, it will be clipped to the intersection of the source and destination bounds.

Returns:

A reference to the supplied WritableRaster, or to a new WritableRaster if the supplied one was null.

Throws:

java.lang.IllegalArgumentException - if the source is null.
java.lang.IllegalArgumentException - if the source's SampleModel is not of integral type.
java.lang.IllegalArgumentException - if the destination's data type or number of bands differ from those returned by getDataType() and getDestNumBands().

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    byte[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    byte[][] dstData,
                    int[] tblOffsets,
                    byte[][] tblData)
```

---

lookupU

```
private void lookupU(int srcLineStride,
                     int srcPixelStride,
                     int[] srcBandOffsets,
                     short[][] srcData,
                     int width,
                     int height,
                     int bands,
                     int dstLineStride,
                     int dstPixelStride,
                     int[] dstBandOffsets,
                     byte[][] dstData,
                     int[] tblOffsets,
                     byte[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    short[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    byte[][] dstData,
                    int[] tblOffsets,
                    byte[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    int[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    byte[][] dstData,
                    int[] tblOffsets,
                    byte[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    byte[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    short[][] dstData,
                    int[] tblOffsets,
                    short[][] tblData)
```

lookupU

```
private void lookupU(int srcLineStride,
                     int srcPixelStride,
                     int[] srcBandOffsets,
                     short[][] srcData,
                     int width,
                     int height,
                     int bands,
                     int dstLineStride,
                     int dstPixelStride,
                     int[] dstBandOffsets,
                     short[][] dstData,
                     int[] tblOffsets,
                     short[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    short[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    short[][] dstData,
                    int[] tblOffsets,
                    short[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    int[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    short[][] dstData,
                    int[] tblOffsets,
                    short[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    byte[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    int[][] dstData,
                    int[] tblOffsets,
                    int[][] tblData)
```

---

lookupU

```
private void lookupU(int srcLineStride,
                     int srcPixelStride,
                     int[] srcBandOffsets,
                     short[][] srcData,
                     int width,
                     int height,
                     int bands,
                     int dstLineStride,
                     int dstPixelStride,
                     int[] dstBandOffsets,
                     int[][] dstData,
                     int[] tblOffsets,
                     int[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    short[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    int[][] dstData,
                    int[] tblOffsets,
                    int[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    int[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    int[][] dstData,
                    int[] tblOffsets,
                    int[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    byte[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    float[][] dstData,
                    int[] tblOffsets,
                    float[][] tblData)
```

lookupU

```
private void lookupU(int srcLineStride,
                     int srcPixelStride,
                     int[] srcBandOffsets,
                     short[][] srcData,
                     int width,
                     int height,
                     int bands,
                     int dstLineStride,
                     int dstPixelStride,
                     int[] dstBandOffsets,
                     float[][] dstData,
                     int[] tblOffsets,
                     float[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    short[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    float[][] dstData,
                    int[] tblOffsets,
                    float[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    int[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    float[][] dstData,
                    int[] tblOffsets,
                    float[][] tblData)
```

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    byte[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    double[][] dstData,
                    int[] tblOffsets,
                    double[][] tblData)
```

---

lookupU

```
private void lookupU(int srcLineStride,
                     int srcPixelStride,
                     int[] srcBandOffsets,
                     short[][] srcData,
                     int width,
                     int height,
                     int bands,
                     int dstLineStride,
                     int dstPixelStride,
                     int[] dstBandOffsets,
                     double[][] dstData,
                     int[] tblOffsets,
                     double[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    short[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    double[][] dstData,
                    int[] tblOffsets,
                    double[][] tblData)
```

---

lookup

```
private void lookup(int srcLineStride,
                    int srcPixelStride,
                    int[] srcBandOffsets,
                    int[][] srcData,
                    int width,
                    int height,
                    int bands,
                    int dstLineStride,
                    int dstPixelStride,
                    int[] dstBandOffsets,
                    double[][] dstData,
                    int[] tblOffsets,
                    double[][] tblData)
```

---

findNearestEntry

```
public int findNearestEntry(float[] pixel)
```

Determine which entry in the `LookupTableJAI` is closest in Euclidean distance to the argument pixel.

Parameters:

`pixel` - the pixel the closest entry to which is to be found.

Returns:

the index of the closest entry. If the data array of the lookup table is in the format data[numBands][numEntries], then the value *v* for band *b* of the closest entry is

```
v = data[b][index - lookup.getOffset()]
```

where *index* is the returned value of this method.

Throws:

NullPointerException - if pixel is null.

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the `LookupTableJAI`.

Parameters:

`out` - The `ObjectOutputStream`.

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the `LookupTableJAI`.

Parameters:

`in` - The `ObjectInputStream`.

javax.media.jai
Class MultiResolutionRenderableImage

```
java.lang.Object
  |
  +--javax.media.jai.MultiResolutionRenderableImage
```

public class **MultiResolutionRenderableImage**
extends java.lang.Object
implements java.awt.image.renderable.RenderableImage, java.io.Serializable

A RenderableImage that produces renderings based on a set of supplied RenderedImages at various resolutions.

Field Detail

renderedSource

```
protected transient java.awt.image.RenderedImage[] renderedSource
```

An array of RenderedImage sources.

numSources

```
private int numSources
```

aspect

```
protected float aspect
```

The aspect ratio, derived from the highest-resolution source.

minX

```
protected float minX
```

The min X coordinate in Renderable coordinates.

minY

```
protected float minY
```

The min Y coordinate in Renderable coordinates.

width

```
protected float width
```

The width in Renderable coordinates.

height

```
protected float height
```

The height in Renderable coordinates.

Constructor Detail

MultiResolutionRenderableImage

```
public MultiResolutionRenderableImage(java.util.Vector renderedSources,
                                      float minX,
                                      float minY,
                                      float height)
```

Constructs a MultiResolutionRenderableImage with given dimensions from a Vector of progressively lower resolution versions of a RenderedImage.

Parameters:

`renderedSources` - a Vector of RenderedImages.
`minX` - the minimum X coordinate of the Renderable, as a float.
`minY` - the minimum Y coordinate of the Renderable, as a float.
`height` - the height of the Renderable, as a float.

Throws:

java.lang.IllegalArgumentException - if the supplied height is non-positive.

Method Detail

getSources

`public java.util.Vector **getSources**()`

Returns an empty Vector, indicating that this RenderableImage has no Renderable sources.

Specified by:

getSources in interface java.awt.image.renderable.RenderableImage

Returns:

an empty Vector.

---

getProperty

`public java.lang.Object **getProperty**(java.lang.String name)`

Gets a property from the property set of this image. If the property name is not recognized, java.awt.Image.UndefinedProperty will be returned. The default implementation returns `java.awt.Image.UndefinedProperty`.

Specified by:

getProperty in interface java.awt.image.renderable.RenderableImage

Parameters:

`name` - the name of the property to get, as a String.

Returns:

a reference to the property Object, or the value java.awt.Image.UndefinedProperty.

---

getPropertyNames

`public java.lang.String[] **getPropertyNames**()`

Returns a list of the properties recognized by this image. If no properties are recognized by this image, null will be returned. The default implementation returns `null`, i.e., no property names are recognized.

Specified by:

getPropertyNames in interface java.awt.image.renderable.RenderableImage

Returns:

an array of Strings representing valid property names.

---

getWidth

`public float **getWidth**()`

Returns the floating-point width of the RenderableImage.

Specified by:

getWidth in interface java.awt.image.renderable.RenderableImage

---

getHeight

`public float **getHeight**()`

Returns the floating-point height of the RenderableImage.

Specified by:

getHeight in interface java.awt.image.renderable.RenderableImage

---

getMinX

`public float **getMinX**()`

Returns the floating-point min X coordinate of the RenderableImage.

Specified by:

getMinX in interface java.awt.image.renderable.RenderableImage

---

getMaxX

```
public float getMaxX()
```

Returns the floating-point max X coordinate of the RenderableImage.

---

getMinY

```
public float getMinY()
```

Returns the floating-point min Y coordinate of the RenderableImage.
Specified by:
getMinY in interface java.awt.image.renderable.RenderableImage

---

getMaxY

```
public float getMaxY()
```

Returns the floating-point max Y coordinate of the RenderableImage.

---

isDynamic

```
public boolean isDynamic()
```

Returns false since successive renderings (that is, calls to createRendering() or createScaledRendering()) with the same arguments will never produce different results.
Specified by:
isDynamic in interface java.awt.image.renderable.RenderableImage

---

createScaledRendering

```
public java.awt.image.RenderedImage createScaledRendering(int width,
                                                          int height,
                                                          java.awt.RenderingHints hints)
```

Returns a rendering with a given width, height, and rendering hints.

If a JAI rendering hint named `JAI.KEY_INTERPOLATION` is provided, its corresponding `Interpolation` object is used as an argument to the JAI operator used to scale the image. If no such hint is present, an instance of `InterpolationNearest` is used.
Specified by:
createScaledRendering in interface java.awt.image.renderable.RenderableImage
Parameters:
`width` - the width of the rendering in pixels.
`height` - the height of the rendering in pixels.
`hints` - a Hashtable of rendering hints.
Throws:
java.lang.IllegalArgumentException - if width or height are non-positive.

---

createDefaultRendering

```
public java.awt.image.RenderedImage createDefaultRendering()
```

Returns the full resolution source RenderedImage with no rendering hints.
Specified by:
createDefaultRendering in interface java.awt.image.renderable.RenderableImage

---

createRendering

```
public java.awt.image.RenderedImage createRendering(java.awt.image.renderable.RenderContext renderContext)
```

Returns a rendering based on a RenderContext.

If a JAI rendering hint named `JAI.KEY_INTERPOLATION` is provided, its corresponding `Interpolation` object is used as an argument to the JAI operator used to transform the image. If no such hint is present, an instance of `InterpolationNearest` is used.

The `RenderContext` may contain a `Shape` that represents the area-of-interest (aoi). If the aoi is specifed, it is still legal to return an image that's larger than this aoi. Therefore, by default, the aoi, if specified, is ignored at the rendering.
Specified by:
createRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`renderContext` - a RenderContext describing the transform rendering hints.
Throws:
NullPointerException - if renderContext is null.

---

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the MultiResolutionRenderableImage.
Parameters:
`out` - The stream provided by the VM to which to write the object.

---

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the MultiResolutionRenderableImage.
Parameters:
`in` - The stream provided by the VM from which to read the object.

javax.media.jai
Class NoParameterDefault

```
java.lang.Object
   |
   +--javax.media.jai.NoParameterDefault
```

class **NoParameterDefault**
extends java.lang.Object

A class that signifies that a parameter has no default value.

Constructor Detail

NoParameterDefault

`NoParameterDefault()`

javax.media.jai
Class NullOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.PointOpImage
                    |
                    +--javax.media.jai.NullOpImage
```

public class **NullOpImage**
extends PointOpImage

A trivial `OpImage` subclass that simply transmits its source unchanged. This may be useful when an interface requires an `OpImage` but another sort of `RenderedImage` (such as a `BufferedImage` or `TiledImage`) is to be used. Additionally, `NullOpImage` is able to make use of JAI's tile caching mechanisms.

Methods that get or set properties are implemented to forward the requests to the source image; no independent property information is stored in the `NullOpImage` itself.

Field Detail

computeType

```
protected int computeType
```

Constructor Detail

NullOpImage

```
public NullOpImage(java.awt.image.RenderedImage source,
                   TileCache cache,
                   int computeType,
                   ImageLayout layout)
```

Constructs a `NullOpImage`. The image bounds are copied from the source image. The tile grid layout, `SampleModel`, and `ColorModel` may be overridden by an `ImageLayout` parameter. The image bounds (min X and Y, width, and height) are always taken from the source image.

The superclass constructor will be passed a new `ImageLayout` object with all of its fields filled in.

Parameters:

`source` - A `RenderedImage`.
`cache` - a TileCache object to store tiles from this OpImage, or null. If null, a default cache will be used.
`computeType` - A tag indicating whether the source is OpImage.OP_COMPUTE_BOUND, OpImage.OP_IO_BOUND or OpImage.OP_NETWORK_BOUND. This information is used as a hint to optimize OpImage computation.
`layout` - An `ImageLayout` describing the layout parameters that will override the corresponding parameters of the source image layout. The image bounds parameters are ignored.

Throws:

java.lang.IllegalArgumentException - if combining the source bounds with the layout parameter results in negative output width or height.

Method Detail

layoutHelper

```
private static ImageLayout layoutHelper(java.awt.image.RenderedImage source,
                                        ImageLayout il)
```

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                          int tileY)
```

Returns a tile for reading.
Parameters:
`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.
Returns:
The tile as a `Raster`.
Overrides:
computeTile in class PointOpImage

---

computesUniqueTiles

```
public boolean computesUniqueTiles()
```

Returns false as NullOpImage can return via computeTile() tile that are internally cached.
Overrides:
computesUniqueTiles in class OpImage

---

getProperties

```
protected java.util.Hashtable getProperties()
```

Returns the properties from the source image.
Overrides:
getProperties in class PlanarImage

---

setProperties

```
protected void setProperties(java.util.Hashtable properties)
```

Set the properties `Hashtable` of the source image to the supplied `Hashtable`.
Overrides:
setProperties in class PlanarImage

---

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the property names from the source image or `null` if no property names are recognized.
Overrides:
getPropertyNames in class PlanarImage

---

getPropertyNames

```
public java.lang.String[] getPropertyNames(java.lang.String prefix)
```

Returns the property names with the supplied prefix from the source image or `null` if no property names are recognized.
Overrides:
getPropertyNames in class PlanarImage

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Retrieves a property from the source image by name or `java.awt.Image.UndefinedProperty` if the property with the specified name is not defined.
Overrides:
getProperty in class PlanarImage

---

setProperty

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
```

Sets a property on the source image by name.
Overrides:
setProperty in class PlanarImage

---

getOperationComputeType

```
public int getOperationComputeType()
```

Returns one of OP_COMPUTE_BOUND, OP_IO_BOUND, or OP_NETWORK_BOUND to indicate how the operation is likely to spend its time. The answer does not affect the output of the operation, but may allow a scheduler to parallelize the computation of multiple operations more effectively. The default implementation returns OP_COMPUTE_BOUND.
Overrides:
getOperationComputeType in class OpImage

javax.media.jai

Class OpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
```

Direct Known Subclasses:

AreaOpImage, PointOpImage, SourcelessOpImage, StatisticsOpImage, UntiledOpImage, WarpOpImage

---

public abstract class **OpImage**
extends PlanarImage

The parent class for all imaging operations. OpImage centralizes a number of common functions, including connecting sources and sinks during construction of OpImage chains, and tile cache management.

Most significantly, OpImage defines `getTile()` to make calls to the `computeRect()` routine of the subclass, performing cobbling if necessary. Two variants of the `computeRect method exist. The first is used when the OpImage constructor is called with its cobbleSources parameter set to true; it receives cobbled source data in the form of an array of Rasters, one per source.`

`The second computeRect variant is called if cobbleSources has been set to false; it receives an array of PlanarImages and is responsible for performing its own source accesses. This variant may be useful if iterators are to be used for the underlying implementation.`

`Every OpImage subclass must supply overrided versions of at least one of these methods, and specify which one is to be called via the cobbleSources constructor argument. If the designated variant has not been overridden, the default implementation will throw a RuntimeException.`

`Many of these functions are overridden in OpImage's direct subclasses, such as PointOpImage and AreaOpImage. These subclasses also implement the abstract methods mapSourceRect() and mapDestRest(), which describe the relationship between areas in the source and destination images.`

See Also:

`PlanarImage, AreaOpImage, PointOpImage, SourcelessOpImage, WarpOpImage`

Field Detail

OP_COMPUTE_BOUND

`public static final int` **`OP_COMPUTE_BOUND`**

A constant indicating that an operation is likely to spend its time mainly performing computation.

OP_IO_BOUND

`public static final int` **`OP_IO_BOUND`**

A constant indicating that an operation is likely to spend its time mainly performing local I/O.

OP_NETWORK_BOUND

`public static final int` **`OP_NETWORK_BOUND`**

A constant indicating that an operation is likely to spend its time mainly performing network I/O.

---

extenders

`protected BorderExtender[]` **`extenders`**

An array of `BorderExtender`s, one per source, or null. If extenders is non-null, there must be a non-null entry for each source.

---

cobbleSources

```
protected boolean cobbleSources
```

Set to true if computeRect needs contiguous sources.

---

formatTags

```
private RasterFormatTag[] formatTags
```

The default RasterAccessor format tags.

---

cache

```
protected transient TileCache cache
```

A reference to a centralized TileCache object.

Constructor Detail

OpImage

```
public OpImage(java.util.Vector sources,
               BorderExtender[] extenders,
               TileCache cache,
               ImageLayout layout,
               boolean cobbleSources)
```

Constructs an `OpImage`, given a `Vector` of sources.

This constructor makes a copy of the source `Vector`, wrapping non-JAI sources (using `PlanarImage.wrapRenderedImage()`). Each source is informed that this image is now one of its sinks.

The structure of the output image is determined using the following algorithm.

First, if no source images are present, the min X, min Y, width, height, and `SampleModel` fields are set using the `layout` parameter. If `layout` is `null`, or one or more of those fields are not set, an `IllegalArgumentException` will be thrown. The tile grid layout fields are copied from the layout, if present. If not, the tile width and height are set to default values and the tile grid is set to start at the image min X and Y.

If one or more source images are present, the `layout` parameter is combined with the layout of the first source using `PlanarImage.setImageParameters()` to provide initial values for the layout fields other than the image bounds. The bounding rectangles of all sources are intersected, and any of minX, minY, width, and height that were not supplied in the `layout` parameter are set according to the intersected bounds.

For example, if the intersection of the source bounding rectangles extends from (50, 60) and has width=100 and height=200, the minX field of `layout` is set to 80, and the height field is set to 90, the output image will begin at (75, 60) and have width=70 (since getMaxX() on the intersected rectangle is 50 + 100 = 150 and the layout min X is 80), and height=90 (taken directly from the layout).

If the resulting output width or height is negative, an `IllegalArgumentException` will be thrown.

If no `SampleModel` was explicitly supplied using the `layout` parameter, one is automatically constructed. The output `SampleModel` will be interleaved, will have a data type with sufficient range to include all values in the range of any of the sources, and will have a number of bands that is the minimum of that of any of the sources. For the purposes of this computation, a source with an `IndexColorModel` is considered to have as many bands as the number of components in the `ColorModel`, so a single-banded source with a `ColorModel` outputting RGB components will be treated as having three bands.

It is possible to supply a `null` `SampleModel` explicitly using the `layout` parameter. In this case, the calling constructor must set the `sampleModel` instance variable manually.

(The `RasterAccessor` class will automatically detect the case of an indexed source and component destination, and perform expansion of the source pixels. If a means other than `RasterAccessor` is used for pixel access, for example iterators, source pixel expansion is the responsibility of the operation implementor.)

Note that the choice of the output data type is based only on the ranges of the source image data types. For example, mixed `TYPE_BYTE` and `TYPE_SHORT` sources will result in `TYPE_SHORT` output. Sources with `TYPE_SHORT` and `TYPE_USHORT` data types will result in an output of type `TYPE_INT`. However, the nature of the operation is not considered so an operation that performs data type conversion must supply its own `SampleModel`.

If a `SampleModel` was explictly supplied, and its width and height match the tile width and height of the image, it is used as-is. If not, and the `SampleModel` is non-`null`, `createCompatibleSampleModel()` is called to produce the output `SampleModel`.

If no `ColorModel` was explicitly supplied using the `layout` parameter, one is automatically constructed by calling `PlanarImage.createColorModel()` using the `SampleModel` derived from the previous step. Note that this may result in a `null` `ColorModel` if the `SampleModel` is `null` or there is no standard `ColorModel` available for the `SampleModel`.

This standard process may be altered in two ways. First, and preferably, the `layout` parameter will contain those values needed to produce the proper output. Second, the subclass constructor may alter the values of any fields it needs to after calling its superclass constructor. However, once the subclass constructor exits, these fields should not be altered further in order to guarantee a consistent state for the `OpImage`.

If the subclass calls `getFormatTags()` in order to obtain a value for use with the `RasterAccessor` class, it should ensure that the `sampleModel` and `colorModel` fields have their final values prior to making the call, since this method caches its result.

Parameters:

`sources` - a `Vector` of sources, or `null`.
`extenders` - an array of `BorderExtender` objects, one per source.
`cache` - a `TileCache` object to store tiles from this `OpImage`, or `null`. If `null`, a default cache will be used.
`layout` - an `ImageLayout`, or `null`.
`cobbleSources` - a `boolean` indicating whether `computeRect` expects contiguous sources.

Throws:

java.lang.IllegalArgumentException - if no source is supplied and `layout` is `null` or does not contain valid values for min X, min Y, width, height, and `SampleModel`.
java.lang.IllegalArgumentException - if combining the intersected source bounds with the layout parameter results in negative output width or height.

OpImage

```
public OpImage(java.awt.image.RenderedImage source,
               BorderExtender extender,
               TileCache cache,
               ImageLayout layout,
               boolean cobbleSources)
```

Constructs an OpImage, given a single source image.

Parameters:

`source` - a `RenderedImage` source.
`extender` - a `BorderExtender`, or null.
`cache` - a `TileCache` object to store tiles from this `OpImage`, or `null. If null, a default cache will be used.`
`layout - an ImageLayout, or null.`
`cobbleSources - a boolean indicating whether computeRect expects contiguous sources.`

OpImage

```
public OpImage(java.awt.image.RenderedImage source0,
               java.awt.image.RenderedImage source1,
               BorderExtender extender0,
               BorderExtender extender1,
               TileCache cache,
               ImageLayout layout,
               boolean cobbleSources)
```

`Constructs an OpImage, given two source images.`

`Parameters:`

`source0 - a RenderedImage source.`
`source1 - a RenderedImage source.`
`extender0 - a BorderExtender for source 0, or null.`
`extender1 - a BorderExtender for source 1, or null.`
`cache - a TileCache object to store tiles from this OpImage, or null. If null, a default cache will be used.`
`layout - an ImageLayout, or null.`
`cobbleSources - a boolean indicating whether computeRect expects contiguous sources.`

Method Detail

vectorize

```
static java.util.Vector vectorize(java.awt.image.RenderedImage source)
```

A utility method used by constructors to store sources in a Vector.

Parameters:

`source` - The source image.

Returns:
A `Vector` containing the source.

---

vectorize

```
static java.util.Vector vectorize(java.awt.image.RenderedImage source1,
                                  java.awt.image.RenderedImage source2)
```

A utility method used by constructors to store sources in a Vector.
Parameters:
`source1` - The first source image.
`source2` - The second source image.
Returns:
A `Vector` sontaining the source.

---

vectorize

```
static java.util.Vector vectorize(java.awt.image.RenderedImage source1,
                                  java.awt.image.RenderedImage source2,
                                  java.awt.image.RenderedImage source3)
```

A utility method used by constructors to store sources in a Vector.
Parameters:
`source1` - The first source image.
`source2` - The second source image.
`source2` - The third source image.
Returns:
A `Vector` sontaining the source.

---

mergeTypes

```
private static int mergeTypes(int type0,
                              int type1)
```

Returns a type (one of the enumerated constants from DataBuffer) that has sufficent range to contain values from either of two given types. This corresponds to an upwards move in the type lattice.

Note that the merge of SHORT and USHORT is INT, so it is not correct to simply use the larger of the types.

---

initializeNoSource

```
private void initializeNoSource(ImageLayout layout)
```

---

initialize

```
private void initialize(BorderExtender[] extenders,
                        ImageLayout layout,
                        boolean cobbleSources)
```

---

setTileCache

```
public void setTileCache(TileCache cache)
```

Sets the tile cache of this image. If `null`, no caching will be performed. Any previously set cache will be informed that it may release this image’s tiles.
Parameters:
`cache` - a `TileCache` object, or `null` if no caching is desired.

---

getTileFromCache

```
protected java.awt.image.Raster getTileFromCache(int tileX,
                                                 int tileY)
```

Gets a tile from the cache by location.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:
the tile as a `Raster`.

---

addTileToCache

```
protected void addTileToCache(int tileX,
                              int tileY,
                              java.awt.image.Raster tile)
```

Adds a tile at a given location to the cache.

Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
`tile` - the tile as a `Raster`.

---

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Gets a tile for reading.

Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Overrides:
getTile in class PlanarImage

---

computesUniqueTiles

```
public boolean computesUniqueTiles()
```

Returns true if the OpImage returns a unique Raster object every time computeTile() is called. OpImages that internally cache Rasters and return them via computeTile() should return false for this method.

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

The internal counterpart of `getTile()`. The `getTile()` method may perform optimizations such as compute scheduling and interaction with a `TileCache`. This method, on the other hand, is responsible only for computing a particular tile, without regard to its eventual disposition.

The default implementation of the method simply cobbles all of the necessary source data, if `cobbleSources` was set to be `true` at construction time, and calls the appropriate variant of `computeRect`.

More efficient, specialized implementations are provided by subclasses such as `AreaOpImage`, `PointOpImage`, `SourcelessOpImage`, and `WarpOpImage`.

Note that this method should generally only be called by an implementation of `TileScheduler`. Normal users should generally call `getTile` which automatically takes advantage of caching and scheduling to reuse results and increase performance.

---

finalize

```
protected void finalize()
                 throws java.lang.Throwable
```

Uncache all tiles when this image is garbage collected.

Overrides:
finalize in class PlanarImage

---

computeRect

```
protected void computeRect(java.awt.image.Raster[] sources,
                           java.awt.image.WritableRaster dest,
                           java.awt.Rectangle destRect)
```

Computes a rectangle of output, given `Raster` sources. This method should be overridden by `OpImage` subclasses that make use of cobbled sources, as determined by the setting of the `cobbleSources` constructor argument to this class.

The source `Raster`s are guaranteed to include at least the area specified by `mapDestRect(destRect)`. Only the specified destination region should be written.

Since the subclasses of `OpImage` may choose between the cobbling and non-cobbling versions of `computeRect`, it is not possible to leave this method abstract in `OpImage`. Instead, a default implementation is provided that throws a `RuntimeException`.

Parameters:
: `sources` - an array of source `Raster`s, one per source image.
: `dest` - a `WritableRaster` to be filled in.
: `destRect` - the `Rectangle` within the destination to be written.

Throws:
: java.lang.RuntimeException - if a subclass sets `cobbleSources` to `true` but does not supply an implementation of this method.

---

computeRect

```
protected void computeRect(PlanarImage[] sources,
                           java.awt.image.WritableRaster dest,
                           java.awt.Rectangle destRect)
```

Computes a rectangle of output, given `PlanarImage` sources. This method should be overridden by `OpImage` subclasses that do not require cobbled sources; typically they will instantiate iterators to perform source access, but they may access sources directly (via the `SampleModel`/`DataBuffer` interfaces) if they wish.

Since the subclasses of `OpImage` may choose between the cobbling and non-cobbling versions of `computeRect`, it is not possible to leave this method abstract in `OpImage`. Instead, a default implementation is provided that throws a `RuntimeException`.

Parameters:
: `sources` - an array of `PlanarImage` sources.
: `dest` - a `WritableRaster` to be filled in.
: `destRect` - the `Rectangle` within the destination to be written.

Throws:
: java.lang.RuntimeException - if a subclass sets `cobbleSources` to `false` but does not supply an implementation of this method.

---

getOperationComputeType

```
public int getOperationComputeType()
```

Returns one of `OP_COMPUTE_BOUND`, `OP_IO_BOUND`, or `OP_NETWORK_BOUND` to indicate how the operation is likely to spend its time. The answer does not affect the output of the operation, but may allow a scheduler to parallelize the computation of multiple operations more effectively. The default implementation returns `OP_COMPUTE_BOUND`.

---

mapSourceRect

```
public abstract java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                                 int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:
: `sourceRect` - the `Rectangle` in source coordinates.
: `sourceIndex` - the index of the source image.

Returns:
: a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:
: java.lang.IllegalArgumentException - if the source index is negative or greater than that of the last source.
: NullPointerException - if `sourceRect` is `null`.

---

mapDestRect

```
public abstract java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                               int sourceIndex)
```

Returns a conservative estimate of the region of a specified source that is required in order to compute the pixels of a given destination rectangle.

Parameters:
: `destRect` - the `Rectangle` in destination coordinates.
: `sourceIndex` - the index of the source image.

Returns:
a `Rectangle` indicating the required source region.
Throws:
java.lang.IllegalArgumentException - if the source index is negative or greater than that of the last source.
NullPointerException - if `destRect` is `null`.

---

getTileDependencies

```
public java.awt.Point[] getTileDependencies(int tileX,
                                            int tileY,
                                            int sourceIndex)
```

Returns a list of indices of the tiles of a given source image that may be required in order to compute a given tile. Ideally, only tiles that will be requested by means of calls to the source's `getTile()` method should be reported. The default implementation uses `mapDestRect()` to obtain a conservative estimate.

If no dependencies exist because the image has no sources, `null` is returned.

This method may be used by optimized implementations of JAI in order to predict future work and create an optimized schedule for performing it.

A given `OpImage` may mix calls to `getTile()` with calls to other methods such as `getData()` and `copyData()` in order to avoid requesting entire tiles where only a small portion is needed. In such a case, this method may be overridden to provide a more accurate estimate of the set of `getTile()` calls that will actually be performed.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
`sourceIndex` - the index of the source image.
Returns:
an array of `Point`s indicating the source tile dependencies.

---

getTiles

```
public java.awt.image.Raster[] getTiles(java.awt.Point[] tileIndices)
```

Computes the tiles indicated by the given tile indices. This call is preferable to a series of `getTile()` calls because certain implementations can make optimizations based on the knowledge that multiple tiles are being asked for at once.
Parameters:
`tileIndices` - An array of `Point`s representing tile indices.
Returns:
An array of `Raster`s containing the tiles corresponding to the given tile indices.
Overrides:
getTiles in class PlanarImage

---

prefetchTiles

```
public void prefetchTiles(java.awt.Point[] tileIndices)
```

Hints that the given tiles might be needed in the near future. Some implementations wmay spawn one or more threads to compute the tiles, while others may ignore the hint.
Parameters:
`tileIndices` - A list of tile indices indicating which tiles to prefetch.
Overrides:
prefetchTiles in class PlanarImage

---

hasExtender

```
public boolean hasExtender(int sourceIndex)
```

Indicates whether the source with the given index has a `BorderExtender`. If the source index is out of bounds for the source vector of this `OpImage` then an `ArrayIndexOutOfBoundsException` may be thrown.
Parameters:
`sourceIndex` - The index of the source in question.
Returns:
`true` if the indicated source has an extender.

---

getExpandedNumBands

```
public static int getExpandedNumBands(java.awt.image.SampleModel sampleModel,
                                      java.awt.image.ColorModel colorModel)
```

Returns the effective number of bands of an image with a given `SampleModel` and `ColorModel`. Normally, this is given by `sampleModel.getNumBands()`, but for images with an `IndexColorModel` the effective number of bands is given by `colorModel.getNumComponents()`, since a single physical sample represents multiple color components.

---

getAppropriateDataType

```
private static int getAppropriateDataType(java.awt.image.SampleModel sampleModel)
```

---

getFormatTags

```
protected RasterFormatTag[] getFormatTags()
```

Returns the image's format tags to be used with a `RasterAccessor`.

This method will compute and cache the tags the first time it is called on a particular image. The image's `SampleModel` and `ColorModel` must be set to their final values before calling this method.

javax.media.jai
Interface OperationDescriptor

All Known Implementing Classes:
OperationDescriptorImpl

---

public abstract interface **OperationDescriptor**

This interface provides a comprehensive description of a specific image operation. All information regarding the operation, such as its name, version, input, and property, should be listed. Any conditions placed on the operation, such as its input format and legal parameter range, should also be included, and the methods to enforce these conditions should be implemented. A set of `PropertyGenerator`s may be specified to be used as a basis for the operation's property management.

Each family of the image operation in JAI must have a descriptor that implements this interface. The following basic resource data must be provided:

- A global operation name that is visible to all and is the same in all `Locale`s.
- A localized operation name that may be used as a synonym for the global operation name.
- The name of the vendor defining this operation.
- A brief description of this operation.
- An URL where additional documentation on this operation may be found.
- The version of this operation.

Additional information must be provided when appropriate. Only then can this operation be added to an `OperationRegistry`. Furthermore, it is recommended that a detailed description of the operation's functionality be documented in the class comment.

There are two image modes in JAI: the "rendered" mode and the "renderable" mode. An operation supporting the rendered mode takes `RenderedImage`s as its sources, can only be used in a rendered operation chain, and produces a `RenderedImage`. An operation supporting the renderable mode takes `RenderableImage`s as its sources, can only be used in a renderable operation chain, and produces a `RenderableImage`. Therefore, the class types of the sources and the destination of an operation are different between the two modes, but the parameters must be the same for both modes.

Those operations that support the rendered mode must specify this feature using the `isRenderedSupported()` method and implement those methods that supply the additional information for the rendered mode. Those operations that support the renderable mode must specify this feature using the `isRenderableSupported()` method and implement those methods that supply the additional information for the renderable mode.

See Also:
`JAI`

Field Detail

NO_PARAMETER_DEFAULT

`public static final java.lang.Object` **`NO_PARAMETER_DEFAULT`**

An `Object` that signifies that a parameter has no default value.

Method Detail

getResources

`public java.lang.String[][]` **`getResources`**`(java.util.Locale locale)`

Returns the resource data for this operation in the specified `Locale`. It must contain `String` data for the following tags:

- "GlobalName" - A global operation name that is visible to all and is the same in all `Locale`s.
- "LocalName" - A localized operation name that may be used as a synonym for the "GlobalName".
- "Vendor" - The name of the vendor defining this operation.
- "Description" - A brief description of this operation.
- "DocURL" - An URL where additional documentation on this operation may be found.
- "Version" - A free-form version indicator of this operation.

In addition, it may contain `String` data for the following tags when appropriate:

- "arg0Desc", "arg1Desc", ... - Description of the input parameters.
- "hint0Desc", hint1Desc", ... - Description of the rendering hints.

Parameters:
`locale` - The `Locale` for which the information should be localized. It may be different from the default `Locale`.

Returns:

A two-dimensional array of `String`s containing the mandatory and optional resource tags and their corresponding resource data.

---

getResourceBundle

`public java.util.ResourceBundle **getResourceBundle**(java.util.Locale locale)`

Returns the resource data for this operation in the specified `Locale` in a `ResourceBundle`. The resource data values are taken from the `getResources()` method which must be implemented by each operation descriptor.

Parameters:

`locale` - The `Locale` for which the information should be localized. It may be different from the default `Locale`.

Returns:

A `ResourceBundle` containing the mandatory and optional resource information.

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerator`s implementing the property inheritance for this operation. They may be used as a basis for the operation's property management.

Returns:

An array of `PropertyGenerator`s, or `null` if this operation does not have any of its own `PropertyGenerator`s.

---

getName

`public java.lang.String **getName**()`

Returns the name of this operation; this is the same as the `GlobalName` value in the resources.

Returns:

A `String` representing the operation's global name.

---

getNumSources

`public int **getNumSources**()`

Returns the number of sources required by this operation.

---

isRenderedSupported

`public boolean **isRenderedSupported**()`

Returns `true` if this operation supports the rendered image mode. That is, it may be performed on `RenderedImage` sources in a rendered operation chain, and produces a rendered result. The `JAI.create()` method should be used to instantiate the operation.

---

isImmediate

`public boolean **isImmediate**()`

Returns `true` if the operation should be rendered immediately during the call to `JAI.create()`; that is, the operation is placed in immediate mode. If `true`, and the rendering fails, `null` will be returned from `JAI.create()`. If `false`, `JAI.create()` will return an instance of the `RenderedOp` that may be asked to render itself at a later time; this rendering may fail silently at that time. This method applies to the rendered mode only.

Operations that rely on an external resource, such as a source file, or that produce externally-visible side effects, such as writing to an output file, should return `true` from this method. Operations that rely only on their sources and parameters usually wish to return `false` in order to defer rendering as long as possible.

---

getSourceClasses

`public java.lang.Class[] **getSourceClasses**()`

Returns an array of `Class`es that describe the types of sources required by this operation in the rendered image mode. If this operation has no source, this method returns `null`.

---

getDestClass

```
public java.lang.Class getDestClass()
```

Returns a `Class` that describes the type of destination this operation produces in the rendered image mode. Currently JAI supports two destination class types: `java.awt.image.RenderedImage.class` and `java.util.Collection.class`.

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Returns `true` if this operation is capable of handling the input rendered source(s) and/or parameter(s) specified in the `ParameterBlock`, or `false` otherwise, in which case an explanatory message may be appended to the `StringBuffer`.

This method is the standard place where input arguments are validated against this operation's specification for the rendered mode. It is called by `JAI.create()` as a part of its validation process. Thus it is strongly recommended that the application programs use the `JAI.create()` methods to instantiate all the rendered operations.

This method sets all the undefined parameters in the `ParameterBlock` to their default values, if the default values are specified.

Parameters:

`args` - Input arguments, including source(s) and/or parameter(s).
`msg` - A string that may contain error messages.

---

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` if this operation supports the renderable image mode. That is, it may be performed on `RenderableImage` sources in a renderable operation chain, and produces a renderable result. The `JAI.createRenderable()` method should be used to instantiate the operation.

If this method returns `true`, all the additional methods that supply the renderable mode information must be implemented.

---

getRenderableSourceClasses

```
public java.lang.Class[] getRenderableSourceClasses()
```

Returns an array of `Class`es that describe the types of sources required by this operation in the renderable image mode. If this operation does not support the renderable mode, or if it has no source, this method returns `null`.

---

getRenderableDestClass

```
public java.lang.Class getRenderableDestClass()
```

Returns a `Class` that describes the type of destination this operation produces in the renderable image mode. Currently JAI supports two destination class types: `java.awt.image.renderable.RenderableImage.class` and `java.util.Collection.class`.

---

validateRenderableArguments

```
public boolean validateRenderableArguments(java.awt.image.renderable.ParameterBlock args,
                                           java.lang.StringBuffer msg)
```

Returns `true` if this operation is capable of handling the input renderable source(s) and/or parameter(s) specified in the `ParameterBlock`, or `false` otherwise, in which case an explanatory message may be appended to the `StringBuffer`.

This method is the standard place where input arguments are validated against this operation's specification for the renderable mode. It is called by `JAI.createRenderable()` as a part of its validation process. Thus it is strongly recommended that the application programs use the `JAI.createRenderable()` method to instantiate all the renderable operations.

This method sets all the undefined parameters in the `ParameterBlock` to their default values, if the default values are specified.

If this operation does not support the renderable mode, this method returns `false` regardless of the input arguments

Parameters:

`args` - Input arguments, including source(s) and/or parameter(s).
`msg` - A string that may contain error messages.

getNumParameters

`public int getNumParameters()`

Returns the number of parameters (not including the sources) required by this operation.

getParamClasses

`public java.lang.Class[] getParamClasses()`

Returns an array of `Class`es that describe the types of parameters required by this operation. If this operation has no parameter, this method returns `null`.

getParamNames

`public java.lang.String[] getParamNames()`

Returns an array of `String`s that are the localized parameter names of this operation. If this operation has no parameter, this method returns `null`.

getParamDefaults

`public java.lang.Object[] getParamDefaults()`

Returns an array of `Object`s that define the default values of the parameters for this operation. Default values may be `null`. When instantiating the operation, the default values may be used for those parameters whose values are not supplied. The `NO_PARAMETER_DEFAULT` static `Object` indicates that a parameter has no default value. If this operation has no parameter, this method returns `null`.

getParamDefaultValue

`public java.lang.Object getParamDefaultValue(int index)`

Returns the default value of a specified parameter. The default value may be `null`. If a parameter has no default value, this method returns `NO_PARAMETER_DEFAULT`.

Parameters:

`index` - The index of the parameter whose default value is queried.

Throws:

NullPointerException - if this operation has no parameter.
ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

getParamMinValue

`public java.lang.Number getParamMinValue(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation. If the specified parameter is non-numeric, this method returns `null`.

The return value should be of the class type appropriate for the parameter’s type, that is, `Byte` for a `byte` parameter, `Integer` for an `int` parameter, and so forth.

Parameters:

`index` - The index of the numeric parameter whose minimum value is queried.

Returns:

A `Number` representing the minimum legal value of the queried parameter, or `null`.

Throws:

NullPointerException - if this operation has no parameter.
ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

getParamMaxValue

`public java.lang.Number getParamMaxValue(int index)`

Returns the maximum legal value of a specified numeric parameter for this operation. If the specified parameter is non-numeric, this method returns `null`.

The return value should be of the class type appropriate for the parameter’s type, that is, `Byte` for a `byte` parameter, `Integer` for an `int` parameter, and so forth.

Parameters:

`index` - The index of the numeric parameter whose maximum value is queried.

Returns:

A `Number` representing the maximum legal value of the queried parameter, or `null`.

Throws:

NullPointerException - if this operation has no parameter.

ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

javax.media.jai

Class OperationDescriptorImpl

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
```

Direct Known Subclasses:

AbsoluteDescriptor, AddCollectionDescriptor, AddConstDescriptor, AddConstToCollectionDescriptor, AddDescriptor, AffineDescriptor, AndConstDescriptor, AndDescriptor, AWTImageDescriptor, BandCombineDescriptor, BandSelectDescriptor, BMPDescriptor, BorderDescriptor, BoxFilterDescriptor, ClampDescriptor, ColorConvertDescriptor, CompositeDescriptor, ConjugateDescriptor, ConstantDescriptor, ConvolveDescriptor, CropDescriptor, DCTDescriptor, DFTDescriptor, DivideByConstDescriptor, DivideComplexDescriptor, DivideDescriptor, DivideIntoConstDescriptor, EncodeDescriptor, ErrorDiffusionDescriptor, ExpDescriptor, ExtremaDescriptor, FileLoadDescriptor, FileStoreDescriptor, FormatDescriptor, FPXDescriptor, GIFDescriptor, GradientMagnitudeDescriptor, HistogramDescriptor, IDCTDescriptor, IDFTDescriptor, IIPDescriptor, IIPResolutionDescriptor, ImageFunctionDescriptor, InvertDescriptor, JPEGDescriptor, LogDescriptor, LookupDescriptor, MagnitudeDescriptor, MagnitudeSquaredDescriptor, MatchCDFDescriptor, MaxDescriptor, MeanDescriptor, MedianFilterDescriptor, MinDescriptor, MultiplyComplexDescriptor, MultiplyConstDescriptor, MultiplyDescriptor, NotDescriptor, OrConstDescriptor, OrderedDitherDescriptor, OrDescriptor, OverlayDescriptor, PatternDescriptor, PeriodicShiftDescriptor, PhaseDescriptor, PiecewiseDescriptor, PNGDescriptor, PNMDescriptor, PolarToComplexDescriptor, RenderableDescriptor, RescaleDescriptor, RotateDescriptor, ScaleDescriptor, ShearDescriptor, StreamDescriptor, SubtractConstDescriptor, SubtractDescriptor, SubtractFromConstDescriptor, ThresholdDescriptor, TIFFDescriptor, TranslateDescriptor, TransposeDescriptor, URLDescriptor, WarpDescriptor, XorConstDescriptor, XorDescriptor

---

public abstract class **OperationDescriptorImpl**
extends java.lang.Object
implements OperationDescriptor

This class provides a concrete implementation of the `OperationDescriptor` interface, and is suitable for subclassing.

See Also:

`OperationDescriptor`

Field Detail

resources

`protected java.lang.String[][] **resources**`

The resource tags and their corresponding data, stored as an two-dimensional `String` array.

---

sourceClasses

`protected java.lang.Class[] **sourceClasses**`

An array of `Class`es that describe the types of sources required by this operation in the rendered mode.

---

renderableSourceClasses

`protected java.lang.Class[] **renderableSourceClasses**`

An array of `Class`es that describe the types of sources required by this operation in the renderable mode. The length of this array must be the same as the length of the `sourceClasses` array.

---

paramClasses

`protected java.lang.Class[] **paramClasses**`

An array of `Class`es that describe the types of parameters required by this operation.

---

paramNames

`protected java.lang.String[] **paramNames**`

An array of `String`s that are the localized parameter names of this operation. The names must be listed in the same order corresponding to the parameter `Classes`es.

paramDefaults

```
protected java.lang.Object[] paramDefaults
```

An array of `Object`s that define the default values of the parameters of this operation. The values must be listed in the same order corresponding to the parameter `Class`es. The default value may be `null`. The `OperationDescriptor.NO_PARAMETER_DEFAULT` static `Object` indicates that a parameter has no default value.

---

name

```
private java.lang.String name
```

The global name of this operation.

Constructor Detail

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               java.lang.Class[] sourceClasses,
                               java.lang.Class[] renderableSourceClasses,
                               java.lang.Class[] paramClasses,
                               java.lang.String[] paramNames,
                               java.lang.Object[] paramDefaults)
```

Constructor.

Parameters:

`resources` - The resource tags and their corresponding data.
`sourceClasses` - The source types required by this operation in the rendered mode. It may be `null` if this operation does not support the rendered mode, or if it has no sources.
`renderableSourceClasses` - The source types required by this operation in the renderable mode. It may be `null` if this operation does not support the renderable mode, or if it has no sources.
`paramClasses` - The parameter types required by this operation. It may be `null` if this operation has no parameters.
`paramNames` - The localized parameter names. It may be `null` if this operation has no parameters.
`paramDefaults` - The parameter default values. It may be `null` if this operation has no parameters, or none of the parameters has a default value.

Throws:

NullPointerException - if `resources` is `null`.
NullPointerException - if this operation supports the rendered mode, and it has sources, and `sourceClasses` is `null`.
NullPointerException - if this operation supports the renderable mode, and it has sources, and `renderableSourceClasses` is `null`.
java.lang.IllegalArgumentException - if `sourceClasses` and `renderableSourceClasses` (if both are not `null`) do not have the same number of elements.
NullPointerException - if this operation has parameters and `paramClasses` or `paramNames` is `null`.
java.lang.IllegalArgumentException - if this operation has parameters and `paramClasses`, `paramNames`, and `paramDefaults` (if all are not `null`) do not all have the same number of elements.

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               java.lang.Class[] sourceClasses)
```

Constructor for operations that supports only the rendered mode and requires no parameters.

Parameters:

`resources` - The resource tags and their corresponding data.
`sourceClasses` - The source types required by this operation in the rendered mode. It may be `null` if this operation has no sources.

Throws:

NullPointerException - if `resources` is `null`.

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               java.lang.Class[] sourceClasses,
                               java.lang.Class[] renderableSourceClasses)
```

Constructor for operations that supports either the rendered or the renderable or both modes and requires no parameters.

Parameters:

`resources` - The resource tags and their corresponding data.
`sourceClasses` - The source types required by this operation in the rendered mode. It may be `null` if this operation does not support the rendered mode, or if it has no sources.
`renderableSourceClasses` - The source types required by this operation in the renderable mode. It may be `null` if this operation does not support the renderable mode, or if it has no sources.

Throws:

NullPointerException - if `resources` is `null`.
NullPointerException - if this operation supports the rendered mode, and it has sources, and `sourceClasses` is `null`.
NullPointerException - if this operation supports the renderable mode, and it has sources, and `renderableSourceClasses` is `null`.
java.lang.IllegalArgumentException - if `sourceClasses` and `renderableSourceClasses` (if both are not `null`) do not have the same number of elements.

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               java.lang.Class[] paramClasses,
                               java.lang.String[] paramNames,
                               java.lang.Object[] paramDefaults)
```

Constructor for operations that supports either the rendered or the renderable or both modes and requires no sources.

Throws:

NullPointerException - if `resources` is `null`.
NullPointerException - if this operation has parameters and `paramClasses` or `paramNames` is `null`.
java.lang.IllegalArgumentException - if this operation has parameters and `paramClasses`, `paramNames`, and `paramDefaults` (if not `null`) do not all have the same number of elements.

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               int numSources,
                               java.lang.Class[] paramClasses,
                               java.lang.String[] paramNames,
                               java.lang.Object[] paramDefaults)
```

Constructor for operations that supports either the rendered or the renderable or both modes. The class type for all the source(s) of the rendered mode (if supported) is set to `java.awt.image.RenderedImage.class`. The class type for all the source(s) of the renderable mode (if supported) is set to `java.awt.image.renderable.RenderableImage`.

Parameters:

`resources` - The resource tags and their corresponding data.
`numSources` - The number of sources required by this operation. It should not be negative. A negative value indicates this operation has no sources.
`paramClasses` - The parameter types required by this operation. It may be `null` if this operation has no parameters.
`paramNames` - The localized parameter names. It may be `null` if this operation has no parameters.
`paramDefaults` - The parameter default values. It may be `null` if this operation has no parameters, or none of the parameters has a default value.

Throws:

NullPointerException - if `resources` is `null`.
NullPointerException - if this operation has parameters and `paramClasses` or `paramNames` is `null`.
java.lang.IllegalArgumentException - if this operation has parameters and `paramClasses`, `paramNames`, and `paramDefaults` (if not `null`) do not all have the same number of elements.

OperationDescriptorImpl

```
public OperationDescriptorImpl(java.lang.String[][] resources,
                               int numSources)
```

Constructor for operations that support the rendered mode and possibly the renderable mode and require no parameters. The class type for all the source(s) of the rendered mode is set to `java.awt.image.RenderedImage.class`. The class type for all the source(s) of the renderable mode (if supported) is set to `java.awt.image.renderable.RenderableImage`.

Parameters:

`resources` - The resource tags and their corresponding data.
`numSources` - The number of sources required by this operation. It should not be negative. A negative value indicates this operation has no sources.

Throws:
NullPointerException - if `resources` is `null`.

Method Detail

getResources

`public java.lang.String[][] **getResources**(java.util.Locale locale)`

Returns the resource data for this operation. It must contain `String` data for the following tags: "GlobalName", "LocalName", "Vendor", "Description", "DocURL", and "Version". Additional resources should be supplied when appropriate.

The default implementation simply returns a reference to the local "resources" variable, which should be supplied by each subclass by way of the superclass constructor. It also ignores the `Locale` argument, and always returns the `String`s in the default `Locale`.

Specified by:
getResources in interface OperationDescriptor

Parameters:
`locale` - The `Locale` in which to localize the resource data.

---

getResourceBundle

`public java.util.ResourceBundle **getResourceBundle**(java.util.Locale locale)`

Returns the resource data for this operation in a `ResourceBundle`. The resource data are taken from the `getResources()` method.

The default implementation ignores the `Locale` argument, and always returns the resources in the default `Locale`.

Specified by:
getResourceBundle in interface OperationDescriptor

Parameters:
`locale` - The `Locale` in which to localize the resource data.

Returns:
A `ResourceBundle` containing mandatory and optional resource information.

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerator`s implementing the property inheritance for this operation. The default implementation returns `null`, indicating that source properties are simply copied. Subclasses should override this method if they wish to produce inherited properties.

Specified by:
getPropertyGenerators in interface OperationDescriptor

---

getName

`public java.lang.String **getName**()`

Returns the name of this operation; this is the same as the `GlobalName` value in the resources and is visible to all.

Specified by:
getName in interface OperationDescriptor

Returns:
A `String` representing the operation’s global name.

Throws:
MissingResourceException - if the `GlobalName` resource value is not supplied in the `resources`.

---

getNumSources

`public int **getNumSources**()`

Returns the number of sources required by this operation.

Specified by:
getNumSources in interface OperationDescriptor

---

isRenderedSupported

`public boolean **isRenderedSupported**()`

Returns `true` if this operation supports the rendered mode. The default implementation in this class returns `true`.

Specified by:

isRenderedSupported in interface OperationDescriptor

---

isImmediate

`public boolean **isImmediate**()`

Returns `true` if the operation should be rendered immediately during the call to `JAI.create()`; that is, the operation is placed in immediate mode.

The default implementation in this class returns `false` so that deferred execution is invoked. Operations that wish to be placed in the immediate mode must override this implementation.

Specified by:

isImmediate in interface OperationDescriptor

---

getSourceClasses

`public java.lang.Class[] **getSourceClasses**()`

Returns the source class types of this operation for the rendered mode. If this operation has no sources, or if it does not support the rendered mode, this method returns `null`.

Specified by:

getSourceClasses in interface OperationDescriptor

---

getDestClass

`public java.lang.Class **getDestClass**()`

Returns the destination class type of this operation for the rendered mode. The default implementation in this class returns `java.awt.image.RenderedImage.class` if this operation supports the rendered mode, or `null` otherwise.

Specified by:

getDestClass in interface OperationDescriptor

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Returns `true` if this operation supports the rendered mode, and is capable of handling the input arguments for the rendered mode. The default implementation validates both the source(s) and the parameter(s).

Additional validations should be added by each individual operation based on its specification.

Specified by:

validateArguments in interface OperationDescriptor

Throws:

NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` if this operation supports the renderable mode. The default implementation in this class returns `false`. Operations that support the renderable mode must override this implementation.

Specified by:

isRenderableSupported in interface OperationDescriptor

---

getRenderableSourceClasses

`public java.lang.Class[] **getRenderableSourceClasses**()`

Returns the source class types of this operation for the renderable mode. If this operation has no sources, or if it does not support the renderable mode, this method returns `null`.

Specified by:

getRenderableSourceClasses in interface OperationDescriptor

---

getRenderableDestClass

```
public java.lang.Class getRenderableDestClass()
```

Returns the destination class type of this operation for the renderable mode. The default implementation in this class returns `java.awt.image.renderable.RenderableImage.class` if this operation supports the renderable mode, or `null` otherwise.

Specified by:

getRenderableDestClass in interface OperationDescriptor

---

validateRenderableArguments

```
public boolean validateRenderableArguments(java.awt.image.renderable.ParameterBlock args,
                                           java.lang.StringBuffer msg)
```

Returns `true` if this operation supports the renderable mode, and is capable of handling the input arguments for the renderable mode. The default implementation validates both the source(s) and the parameter(s).

If this operation does not support the renderable mode, this method returns `false` regardless of the input arguments.

Additional validations should be added by each individual operation based on its specification.

Specified by:

validateRenderableArguments in interface OperationDescriptor

Throws:

NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

getNumParameters

```
public int getNumParameters()
```

Returns the number of parameters (not including sources) required by this operation.

Specified by:

getNumParameters in interface OperationDescriptor

---

getParamClasses

```
public java.lang.Class[] getParamClasses()
```

Returns the parameter class types of this operation. If this operation has no parameters, this method returns `null`.

Specified by:

getParamClasses in interface OperationDescriptor

---

getParamNames

```
public java.lang.String[] getParamNames()
```

Returns the localized parameter names of this operation. If this operation has no parameters, this method returns `null`.

Specified by:

getParamNames in interface OperationDescriptor

---

getParamDefaults

```
public java.lang.Object[] getParamDefaults()
```

Returns the default values of the parameters for this operation. If this operation has no parameters, this method returns `null`. If a parameter does not have a default value, the constant `OperationDescriptor.NO_PARAMETER_DEFAULT` should be used. The `validateArguments()` and `validateRenderableArguments` method will return `false` if an input parameter without a default value is supplied as `null`, or if an unspecified tailing parameter does not have a default value.

Specified by:

getParamDefaults in interface OperationDescriptor

---

getParamDefaultValue

```
public java.lang.Object getParamDefaultValue(int index)
```

Returns the default value of specified parameter. The default value may be `null`. If a parameter has no default value, this method returns `OperationDescriptor.NO_PARAMETER_DEFAULT`.

Specified by:
getParamDefaultValue in interface OperationDescriptor
Parameters:
`index` - The index of the parameter whose default value is queried.
Throws:
NullPointerException - if this operation has no parameters.
ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

---

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation. If the specified parameter is non-numeric, this method returns `null`.

The return value should be of the class type appropriate for the parameter’s type, that is, `Byte` for a `byte` parameter, `Integer` for an `int` parameter, and so forth.

The default implementation returns the minimum value in the parameter data type’s full range.

Specified by:
getParamMinValue in interface OperationDescriptor
Parameters:
`index` - The index of the parameter to be queried.
Returns:
A `Number` representing the minimum legal value, or `null` if the specified parameter is not numeric.
Throws:
NullPointerException - if this operation has no parameters.
ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

---

getParamMaxValue

```
public java.lang.Number getParamMaxValue(int index)
```

Returns the maximum legal value of a specified numeric parameter for this operation. If the specified parameter is non-numeric, this method returns `null`.

The return value should be of the class type appropriate for the parameter’s type, that is, `Byte` for a `byte` parameter, `Integer` for an `int` parameter, and so forth.

The default implementation returns the maximum value in the parameter data type’s full range.

Specified by:
getParamMaxValue in interface OperationDescriptor
Parameters:
`index` - The index of the parameter to be queried.
Returns:
A `Number` representing the maximum legal value, or `null` if the specified parameter is not numeric.
Throws:
NullPointerException - if this operation has no parameters.
ArrayIndexOutOfBoundsException - if there is no parameter corresponding to the specified `index`.

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Returns `true` if this operation supports the rendered mode, and is capable of handling the input source(s) for the rendered mode. The default implementation validates the number of sources, the class type of each source, and null source. Subclasses should override this implementation if their requirement on the sources are different from the default.

Throws:
NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

validateRenderableSources

```
protected boolean validateRenderableSources(java.awt.image.renderable.ParameterBlock args,
                                            java.lang.StringBuffer msg)
```

Returns `true` if this operation supports the renderable mode, and is capable of handling the input source(s) for the renderable mode. The default implementation validates the number of sources, the class type of each source, and null source. Subclasses should override this implementation if their requirement on the sources are different from the default.

Throws:
NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

validateSources

```
private boolean validateSources(java.lang.Class[] sources,
                                java.awt.image.renderable.ParameterBlock args,
                                java.lang.StringBuffer msg)
```

Validates sources in the `ParameterBlock` against the sources of the specification.

Throws:
NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Returns `true` if this operation is capable of handling the input parameters. The default implementation validates the number of parameters, the class type of each parameter, and null parameter. For those non-null numeric parameters, it also checks to see if the parameter value is within the minimum and maximum parameter range. Subclasses should override this implementation if their requirement on the parameter objects are different from the default.

JAI allows unspecified tailing parameters if these parameters have default values. This method automatically sets these unspecified parameters to their default values. However, if a parameter, who has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the `ParameterBlock`; else this method returns `false`.

Throws:
NullPointerException - if `args` is `null`.
NullPointerException - if `msg` is `null` and the validation fails.

---

getMinNumParameters

```
private int getMinNumParameters()
```

Returns the minimum number of parameters must be supplied in the `ParameterBlock`.

---

createFormatter

```
private java.text.MessageFormat createFormatter(java.lang.String msg)
```

Creates a `MessageFormat` object and set the `Locale` to default.

javax.media.jai

Class OperationGraph

```
java.lang.Object
  |
  +--javax.media.jai.OperationGraph
```

class **OperationGraph**
extends java.lang.Object

OperationGraph manages a list of products belonging to a particular operation descriptor. The operations have pairwise preferences between them. The getOrderedOperationList method performs a topological sort. The topological sort follows the algorithm described in Horowitz and Sahni, *Fundamentals of Data Structures* (1976), p. 315.

Several minor changes are made to their implementation. First, nodes are represented as objects, not as integers. The count (in-degree) field is not used to link zero in-degree objects, but instead a separate zeroLink field is used. The neighbor lists are stored as Vectors, not linked lists, and enumerations are used to iterate over them.

This class is used by the implementation of the OperationRegistry class and is not intended to be part of the API.

Field Detail

RIFoperations

`protected java.util.Vector` **`RIFoperations`**

A Vector of RIF implementations.

CIFoperations

`protected java.util.Vector` **`CIFoperations`**

A Vector of CIF implementations.

orderedRIFOperations

`protected java.util.Vector` **`orderedRIFOperations`**

The cached list of ordered operations for RIF/CIF

orderedCIFOperations

`protected java.util.Vector` **`orderedCIFOperations`**

isRIFChanged

`protected boolean` **`isRIFChanged`**

isCIFChanged

`protected boolean` **`isCIFChanged`**

lock

`com.sun.media.jai.util.ReaderWriterLock` **`lock`**

Constructor Detail

OperationGraph

`public` **`OperationGraph`**`()`

Constructs an OperationGraph.

Method Detail

addRIF

```
public void addRIF(java.awt.image.renderable.RenderedImageFactory rif)
```

Adds a RIF to an OperationGraph. A new PartialOrderNode is constructed to hold the RIF and its graph adjacency information.

---

addCIF

```
public void addCIF(CollectionImageFactory cif)
```

Adds a CIF to an OperationGraph. A new PartialOrderNode is constructed to hold the CIF and its graph adjacency information.

---

removeRIF

```
public void removeRIF(java.awt.image.renderable.RenderedImageFactory rif)
```

Removes a RIF from an OperationGraph.

---

removeCIF

```
public void removeCIF(CollectionImageFactory cif)
```

Removes a CIF from an OperationGraph.

---

lookupRIF

```
public PartialOrderNode lookupRIF(java.awt.image.renderable.RenderedImageFactory op)
```

Locates a RIF within the vector of PartialOrderNodes. Equality is by object reference. NOTE: CHANGING access from private to public

---

lookupCIF

```
public PartialOrderNode lookupCIF(CollectionImageFactory op)
```

Locates a CIF within the vector of PartialOrderNodes. Equality is by object reference. NOTE: CHANGING access from private to public

---

setRIFPreference

```
public void setRIFPreference(java.awt.image.renderable.RenderedImageFactory preferredOp,
                             java.awt.image.renderable.RenderedImageFactory otherOp)
```

Sets a preference between two RIFs.

---

unsetRIFPreference

```
public void unsetRIFPreference(java.awt.image.renderable.RenderedImageFactory preferredOp,
                               java.awt.image.renderable.RenderedImageFactory otherOp)
```

Removes a preference between two RIFs.

---

setCIFPreference

```
public void setCIFPreference(CollectionImageFactory preferredOp,
                             CollectionImageFactory otherOp)
```

Sets a preference between two CIFs.

---

unsetCIFPreference

```
public void unsetCIFPreference(CollectionImageFactory preferredOp,
                               CollectionImageFactory otherOp)
```

Removes a preference between two CIFs.

---

getOrderedOperationList

`public java.util.Vector **getOrderedOperationList**(java.lang.String imageFactory)`

Returns an ordered list of the specified imageFactory

---

orderList

`private java.util.Vector **orderList**(java.util.Vector operations)`

Performs a topological sort on the set of image factories.

javax.media.jai

Class OperationRegistry

```
java.lang.Object
  |
  +--javax.media.jai.OperationRegistry
```

public class **OperationRegistry**
extends java.lang.Object
implements java.io.Externalizable

A class implementing the translation of operation names into instances of RenderedImageFactory, ContextualRenderedImageFactory and CollectionImageFactory.

The OperationRegistry class maps an operation name into the particular kind of ImageFactory requested, capable of implementing the operation, given a specific set of sources and parameters. The mapping is constructed in several stages:

One or more OperationDescriptors are registered by calling registerOperationDescriptor(). Once an OperationDescriptor has been registered, it may be obtained by name by calling getOperationDescriptor(). It is not possible to register more than one OperationDescriptor under the same name.

A set of RenderedImageFactory objects are registered using the registerRIF method. Each RIF is registered with a specific operation name, and furthermore is given a product name. Similar methods exist for registering a CIF.

A single CRIF is registered under a specific operation name using the registerCRIF method. If multiple CRIFs are registered under the same operation name, the one registered last will be the one honored. Since only a single CRIF is registered under an operation name, no ordering of CRIFs is possible. Thus product preferences do not have any effect on the selection of a CRIF, and preferences amongst CRIFs cannot be set.

The ordering of RIFs is determined by the order of the products attached to an OperationDescriptor, and the order of the RIFs within each product. The orders are established by setting pairwise preferences, resulting in a partial order which is then sorted topologically. The results of creating a cycle are undefined.

The ordering of RIFs within a product is intended to allow vendors to create complex "fallback" chains. An example would be installing a RIF that implements separable convolution ahead of a RIF that implements a more general algorithm.

The ordering of CIFs is managed in a manner identical to the RIFs.

Vendors are encouraged to use unique product names (by means of the Java programming language convention of reversed internet addresses) in order to maximize the likelihood of clean installation. See *The Java Programming Language*, §10.1 for a discussion of this convention in the context of package naming.

Users will, for the most part, only wish to set ordering preferences on the product level, since the RIF/CIF orderings will be complex. However, it is possible for a knowledgable user to insert a RIF/CIF into an existing product for tuning purposes.

The registry handles all names (except class names) in a case-insensitive manner.

The OperationRegistry also has the responsibility of associating a set of PropertyGenerators with each OperationDescriptor. This set will be coalesced into a PropertySource suitable for use by a RenderedOp by the getPropertySource() method. If several PropertyGenerators associated with a particular OperationDescriptor generate the same property, only the last one to be registered will have any effect.

Field Detail

opDescsName

`java.util.Hashtable` **`opDescsName`**

A Hashtable of all the OperationDescriptors, hashed by the operation name of the OperationDescriptors.

products

`java.util.Hashtable` **`products`**

A Hashtable of all the products, hashed by the operation name of the OperationDescriptor to which they belong.

rifs

`java.util.Hashtable` **`rifs`**

A Hashtable of all the RIFs, hashed by a filename that uniquely identifies each registered RIF.

rifsByName

`java.util.Hashtable` **`rifsByName`**

A Hashtable of all the unique RIF filenames, hashed by the RIF they represent.

---

rifcount

`int` **`rifcount`**

A count to give a number to each registered RIF.

---

cifs

`java.util.Hashtable` **`cifs`**

Same as above three structures, but for CIFs.

---

cifsByName

`java.util.Hashtable` **`cifsByName`**

---

cifcount

`int` **`cifcount`**

---

crifs

`java.util.Hashtable` **`crifs`**

Hashtable of all the crifs, hashed by the operationName to which they belong.

---

productPrefs

`java.util.Hashtable` **`productPrefs`**

A Hashtable of all the product preferences, hashed by the operation name descriptor that the products belong to.

---

rifPrefs

`java.util.Hashtable` **`rifPrefs`**

A Hashtable of all the RIF preferences, hashed by the operation name that the RIF belongs to.

---

cifPrefs

`java.util.Hashtable` **`cifPrefs`**

A Hashtable of all the CIF preferences, hashed by the operation name that the CIF belongs to.

---

properties

`protected java.util.Hashtable` **`properties`**

---

suppressed

`protected java.util.Hashtable` **`suppressed`**

---

sourceForProp

`protected java.util.Hashtable` **`sourceForProp`**

---

propNames

```
protected java.util.Hashtable propNames
```

---

formatter

```
private static java.text.MessageFormat formatter
```

Required to I18N compound messages.

---

lock

```
private com.sun.media.jai.util.ReaderWriterLock lock
```

The ReaderWriter Lock for this class.

Constructor Detail

OperationRegistry

```
public OperationRegistry()
```

Default Constructor.

Method Detail

initializeRegistry

```
static OperationRegistry initializeRegistry()
```

Initializes the default registry, creating it if necessary.

Returns:

the default OperationRegistry

---

readInitFile

```
private static RegistryInitData readInitFile(java.io.Reader reader)
                                      throws java.io.IOException
```

Reads the registry initialization file and stores the information read into memory data structures.

Parameters:

`reader` - A Reader stream used to read initialization data from.

Returns:

In memory initialization data.

---

loadDescriptors

```
private void loadDescriptors(RegistryInitData rid)
```

A method for registry initialization.

Parameters:

`rid` - The in-memory initialization data.

---

registerOperationDescriptorNoLock

```
private void registerOperationDescriptorNoLock(OperationDescriptor odesc,
                                               java.lang.String operationName)
```

---

registerRIFNoLock

```
private void registerRIFNoLock(java.lang.String operationName,
                               java.lang.String productName,
                               java.awt.image.renderable.RenderedImageFactory RIF)
```

---

registerCRIFNoLock

```
private void registerCRIFNoLock(java.lang.String operationName,
                                java.awt.image.renderable.ContextualRenderedImageFactory CRIF)
```

---

registerCIFNoLock

```
private void registerCIFNoLock(java.lang.String operationName,
                               java.lang.String productName,
                               CollectionImageFactory CIF)
```

---

setProductPreferenceNoLock

```
private void setProductPreferenceNoLock(java.lang.String operationName,
                                        java.lang.String preferredProductName,
                                        java.lang.String otherProductName)
```

---

setRIFPreferenceNoLock

```
private void setRIFPreferenceNoLock(java.lang.String operationName,
                                    java.lang.String productName,
                                    java.awt.image.renderable.RenderedImageFactory preferredRIF,
                                    java.awt.image.renderable.RenderedImageFactory otherRIF)
```

---

setCIFPreferenceNoLock

```
private void setCIFPreferenceNoLock(java.lang.String operationName,
                                    java.lang.String productName,
                                    CollectionImageFactory preferredCIF,
                                    CollectionImageFactory otherCIF)
```

---

toString

```
public java.lang.String toString()
```

Returns a String representation of the registry.

Returns:
the string representation of this OperationRegistry.

Overrides:
toString in class java.lang.Object

---

writeToStream

```
public void writeToStream(java.io.OutputStream out)
                   throws java.io.IOException
```

Writes out the contents of the OperationRegistry to a stream.

Parameters:
`out` - The OutputStream to which the OperationRegistry state is written.

Throws:
NullPointerException - if out is null.

---

initializeFromStream

```
public void initializeFromStream(java.io.InputStream in)
                          throws java.io.IOException
```

Loads the contents of the OperationRegistry from an InputStream.

Parameters:
`in` - The InputStream from which to read the data.

Throws:
NullPointerException - if in is null.

---

readExternal

```
public void readExternal(java.io.ObjectInput in)
                  throws java.io.IOException,
                         java.lang.ClassNotFoundException
```

Restores the contents of the registry from an ObjectInput which was previously written using the `writeExternal` method.

Specified by:

readExternal in interface java.io.Externalizable

Parameters:

`in` - An ObjectInput from which to read the data.

Throws:

NullPointerException - if in is null.

---

writeExternal

```
public void writeExternal(java.io.ObjectOutput out)
                   throws java.io.IOException
```

Saves the contents of the registry in the format described for the `writeToStream` method.

Specified by:

writeExternal in interface java.io.Externalizable

Parameters:

`out` - An ObjectOutput to which to write the data.

Throws:

NullPointerException - if out is null.

---

registerOperationDescriptor

```
public void registerOperationDescriptor(OperationDescriptor odesc,
                                        java.lang.String operationName)
```

Registers an OperationDescriptor with the registry. Each operation must have an OperationDescriptor before registerRIF() may be called to add RIFs to the operation.

An OperationDescriptor cannot be registered under an operation name under which another OperationDescriptor was registered previously. If such an attempt is made, an error message will be printed.

Parameters:

`odesc` - an OperationDescriptor containing information about the operation.
`operationName` - the operation name as a String.

Throws:

NullPointerException - if odesc is null.
NullPointerException - if operationName is null.

---

unregisterOperationDescriptor

```
public void unregisterOperationDescriptor(java.lang.String operationName)
```

Unregisters an OperationDescriptor from the registry. An error message will be printed if an attempt is made to unregister an OperationDescriptor that was not previously registered.

Parameters:

`operationName` - the operation name as a String.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if any of the PropertyGenerators associated with the OperationDescriptor to be removed is null.

---

getOperationDescriptor

```
public OperationDescriptor getOperationDescriptor(java.lang.String operationName)
```

Returns the OperationDescriptor that is currently registered under the given name, or null if none exists.

Parameters:

`operationName` - the String to be queried.

Returns:

an OperationDescriptor.

Throws:

NullPointerException - if operationName is null.

getOperationDescriptors

```
public java.util.Vector getOperationDescriptors()
```

Returns a Vector of all currently registered OperationDescriptors.

Returns:

a Vector of OperationDescriptors.

getOperationNames

```
public java.lang.String[] getOperationNames()
```

Returns a list of names under which all the OperationDescriptors in the registry are registered.

Returns:

a list of currently existing operation names.

registerRIF

```
public void registerRIF(java.lang.String operationName,
                        java.lang.String productName,
                        java.awt.image.renderable.RenderedImageFactory RIF)
```

Registers a RIF with a particular product and operation. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.
`productName` - the product name, as a String.
`RIF` - the RenderedImageFactory to be registered.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if RIF is null.

unregisterRIF

```
public void unregisterRIF(java.lang.String operationName,
                          java.lang.String productName,
                          java.awt.image.renderable.RenderedImageFactory RIF)
```

Unregisters a RIF from a particular product and operation. An error message will be printed out if the operation was not registered previously or if the product has not been registered under the operation.

Parameters:

`operationName` - the operation name as a String.
`productName` - the product name, as a String.
`RIF` - the RenderedImageFactory to be unregistered.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if RIF is null.

registerCRIF

```
public void registerCRIF(java.lang.String operationName,
                         java.awt.image.renderable.ContextualRenderedImageFactory CRIF)
```

Registers a CRIF under a particular operation. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.
`CRIF` - the ContextualRenderedImageFactory to be registered.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if CRIF is null.

unregisterCRIF

```
public void unregisterCRIF(java.lang.String operationName,
                           java.awt.image.renderable.ContextualRenderedImageFactory CRIF)
```

Unregisters a CRIF from a particular operation. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.

`CRIF` - the ContextualRenderedImageFactory to be unregistered.

Throws:

NullPointerException - if operationName is null.

NullPointerException - if CRIF is null.

---

registerCIF

```
public void registerCIF(java.lang.String operationName,
                        java.lang.String productName,
                        CollectionImageFactory CIF)
```

Registers a CIF with a particular product and operation. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.

`productName` - the product name, as a String.

`CIF` - the CollectionImageFactory to be registered.

Throws:

NullPointerException - if operationName is null.

NullPointerException - if productName is null.

NullPointerException - if CIF is null.

---

unregisterCIF

```
public void unregisterCIF(java.lang.String operationName,
                          java.lang.String productName,
                          CollectionImageFactory CIF)
```

Unregisters a CIF from a particular product and operation. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation previously.

Parameters:

`operationName` - the operation name as a String.

`productName` - the product name, as a String.

`CIF` - the CollectionImageFactory to be unregistered.

Throws:

NullPointerException - if operationName is null.

NullPointerException - if productName is null.

NullPointerException - if CIF is null.

---

setProductPreference

```
public void setProductPreference(java.lang.String operationName,
                                 java.lang.String preferredProductName,
                                 java.lang.String otherProductName)
```

Sets a preference between two products registered under a common OperationDescriptor. An error will be printed out if the operation was not registered previously and no preference will be set. Any attempt to set a preference between a product and itself will be ignored.

Parameters:

`operationName` - the operation name as a String.

`preferredProductName` - the product to be preferred.

`otherProductName` - the other product.

Throws:

NullPointerException - if operationName is null.

NullPointerException - if preferredProductName is null.

NullPointerException - if otherProductName is null.

---

unsetProductPreference

```
public void unsetProductPreference(java.lang.String operationName,
                                   java.lang.String preferredProductName,
                                   java.lang.String otherProductName)
```

Removes a preference between two products registered under a common OperationDescriptor. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.
`preferredProductName` - the product formerly preferred.
`otherProductName` - the other product.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if preferredProductName is null.
NullPointerException - if otherProductName is null.

---

clearProductPreferences

```
public void clearProductPreferences(java.lang.String operationName)
```

Removes all preferences between products registered under a common OperationDescriptor. An error message will be printed out if the operation was not registered previously.

Parameters:

`operationName` - the operation name as a String.

Throws:

NullPointerException - if operationName is null.

---

getProductPreferences

```
public java.lang.String[][] getProductPreferences(java.lang.String operationName)
```

Returns a list of the pairwise product preferences under a particular OperationDescriptor. If no product preferences have been set, returns null.

Parameters:

`operationName` - the operation name as a String.

Returns:

an array of 2-element arrays of Strings.

Throws:

NullPointerException - if operationName is null.

---

getOrderedProductList

```
public java.util.Vector getOrderedProductList(java.lang.String operationName)
```

Returns a list of the products registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set. Cycles will be broken in an arbitrary manner. Returns null if no OperationDescriptor has been registered under this operationName, or if no products exist for this operation.

Parameters:

`operationName` - the operation name as a String.

Returns:

a Vector of Strings representing product names.

Throws:

NullPointerException - if operationName is null.

---

setRIFPreference

```
public void setRIFPreference(java.lang.String operationName,
                             java.lang.String productName,
                             java.awt.image.renderable.RenderedImageFactory preferredRIF,
                             java.awt.image.renderable.RenderedImageFactory otherRIF)
```

Sets a preference between two RIFs within the same product. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation or if either of the supplied RIFs is null. Any attempt to set a preference between a RIF and itself will be ignored.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.
`preferredRIF` - the preferred RenderedImageFactory.
`otherRIF` - the other RenderedImageFactory.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if preferredRIF is null.
NullPointerException - if otherRIF is null.

---

setCIFPreference

```
public void setCIFPreference(java.lang.String operationName,
                             java.lang.String productName,
                             CollectionImageFactory preferredCIF,
                             CollectionImageFactory otherCIF)
```

Sets a preference between two CIFs within the same product. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation or if either of the two supplied CIF's is null. Any attempt to set a preference between a CIF and itself will be ignored.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.
`preferredCIF` - the preferred CollectionRenderedImageFactory.
`otherCIF` - the other CollectionRenderedImageFactory.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if preferredCIF is null.
NullPointerException - if otherCIF is null.

---

unsetRIFPreference

```
public void unsetRIFPreference(java.lang.String operationName,
                               java.lang.String productName,
                               java.awt.image.renderable.RenderedImageFactory preferredRIF,
                               java.awt.image.renderable.RenderedImageFactory otherRIF)
```

Removes a preference between two RIFs within the same product. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation or if either of the two supplied RIF's is null.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.
`preferredRIF` - the formerly preferred RenderedImageFactory.
`otherRIF` - the other RenderedImageFactory.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if preferredRIF is null.
NullPointerException - if otherRIF is null.

---

unsetCIFPreference

```
public void unsetCIFPreference(java.lang.String operationName,
                               java.lang.String productName,
                               CollectionImageFactory preferredCIF,
                               CollectionImageFactory otherCIF)
```

Removes a preference between two CIFs within the same product. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation or if either of the two supplied CIF's is null.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.
`preferredCIF` - the formerly preferred CollectionImageFactory.
`otherCIF` - the other CollectionImageFactory.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.
NullPointerException - if preferredCIF is null.
NullPointerException - if otherCIF is null.

---

clearRIFPreferences

```
public void clearRIFPreferences(java.lang.String operationName,
                                java.lang.String productName)
```

Removes all preferences between RIFs within a product registered under a particular OperationDescriptor. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.

---

clearCIFPreferences

```
public void clearCIFPreferences(java.lang.String operationName,
                                java.lang.String productName)
```

Removes all preferences between CIFs within a product registered under a particular OperationDescriptor. An error message will be printed out if the operation was not registered previously or if the product was not registered under the operation.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.

---

clearOperationPreferences

```
public void clearOperationPreferences(java.lang.String operationName,
                                      java.lang.String productName)
```

Removes all RIF and CIF preferences within a product registered under a particular OperationDescriptor.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.

---

getOrderedRIFList

```
public java.util.Vector getOrderedRIFList(java.lang.String operationName,
                                          java.lang.String productName)
```

Returns a list of the RIFs of a product registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set. Cycles will be broken in an arbitrary manner. Returns null, if the product does not exist under this operationName.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.

Returns:

a Vector of RIFs.

Throws:

NullPointerException - if operationName is null.
NullPointerException - if productName is null.

---

getOrderedCIFList

```
public java.util.Vector getOrderedCIFList(java.lang.String operationName,
                                          java.lang.String productName)
```

Returns a list of the CIFs of a product registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set. Cycles will be broken in an arbitrary manner. Returns null, if the product does not exist under this operationName.

Parameters:

`operationName` - the operation name as a String.
`productName` - the name of the product.

Returns:
a Vector of CIFs.
Throws:
NullPointerException - if operationName is null.
NullPointerException - if productName is null.

---

create

```
public PlanarImage create(java.lang.String operationName,
                          java.awt.image.renderable.ParameterBlock paramBlock,
                          java.awt.RenderingHints renderHints)
```

Constructs a PlanarImage (usually a RenderedOp) representing the results of applying a given operation to a particular ParameterBlock and rendering hints. The registry is used to determine the RIF to be used to instantiate the operation.

If none of the RIFs registered with this OperationRegistry returns a non-null value, null is returned. Exceptions thrown by the RIFs will be caught by this method and will not be propagated.

Parameters:
`operationName` - the operation name as a String.
`paramBlock` - the operation’s ParameterBlock.
`renderHints` - a RenderingHints object containing rendering hints.
Throws:
NullPointerException - if operationName is null.

---

createRenderable

```
public java.awt.image.renderable.ContextualRenderedImageFactory createRenderable(java.lang.String operationName,
                                                                                 java.awt.image.renderable.ParameterBlock paramBlock)
```

Constructs the CRIF to be used to instantiate the operation. Returns null, if no CRIF is registered with the given operation name.

Parameters:
`operationName` - the operation name as a String.
`paramBlock` - the operation’s ParameterBlock.
Throws:
NullPointerException - if operationName is null.

---

createCollection

```
public CollectionImage createCollection(java.lang.String operationName,
                                        java.awt.image.renderable.ParameterBlock args,
                                        java.awt.RenderingHints hints)
```

Constructs a CollectionImage (usually a CollectionOp) representing the results of applying a given operation to a particular ParameterBlock and rendering hints. The registry is used to determine the CIF to be used to instantiate the operation.

If none of the CIFs registered with this OperationRegistry returns a non-null value, null is returned. Exceptions thrown by the CIFs will be caught by this method and will not be propagated.

Parameters:
`operationName` - The operation name as a String.
`args` - The operation’s input parameters.
`hints` - A RenderingHints object containing rendering hints.
Throws:
NullPointerException - if operationName is null.

---

clearPropertyState

```
public void clearPropertyState()
```

Removes all property associated information from this OperationRegistry.

---

addPropertyGenerator

```
public void addPropertyGenerator(java.lang.String operationName,
                                 PropertyGenerator generator)
```

Adds a PropertyGenerator to the registry, associating it with a particular OperationDescriptor.

Parameters:
`operationName` - the operation name as a String.
`generator` - the PropertyGenerator to be added.

Throws:
NullPointerException - if operationName is null.
NullPointerException - if generator is null.

---

hashNames

```
private void hashNames(java.lang.String operationName)
```

---

removePropertyGenerator

```
public void removePropertyGenerator(java.lang.String operationName,
                                    PropertyGenerator generator)
```

Removes a PropertyGenerator from its association with a particular OperationDescriptor in the registry. If the generator was not associated with the operation, nothing happens.

Parameters:
`operationName` - the operation name as a String.
`generator` - the PropertyGenerator to be removed.

Throws:
NullPointerException - if operationName is null.
NullPointerException - if generator is null.

---

suppressProperty

```
public void suppressProperty(java.lang.String operationName,
                             java.lang.String propertyName)
```

Forces a particular property to be suppressed by nodes performing a particular operation. By default, properties are passed through operations unchanged.

Parameters:
`operationName` - the operation name as a String.
`propertyName` - the name of the property to be suppressed.

Throws:
NullPointerException - if operationName is null.
NullPointerException - if propertyName is null.

---

suppressAllProperties

```
public void suppressAllProperties(java.lang.String operationName)
```

Forces all properties to be suppressed by nodes performing a particular operation. By default, properties are passed through operations unchanged.

Parameters:
`operationName` - the operation name as a String.

Throws:
NullPointerException - if operationName is null.

---

copyPropertyFromSource

```
public void copyPropertyFromSource(java.lang.String operationName,
                                   java.lang.String propertyName,
                                   int sourceIndex)
```

Forces a property to be copied from the specified source image by `RenderedOp` nodes performing a particular operation. By default, a property is copied from the first source node that emits it. The result of specifying an invalid source is undefined.

Parameters:
`operationName` - the operation name as a String.
`propertyName` - the name of the property to be copied.
`sourceIndex` - the index of the source to copy the property from.

Throws:
NullPointerException - if operationName is null.
NullPointerException - if propertyName is null.

---

getGeneratedPropertyNames

`public java.lang.String[] getGeneratedPropertyNames(java.lang.String operationName)`

Returns a list of the properties generated by nodes implementing the operation associated with a particular Operation Name. Returns null if no properties are generated.

Parameters:

`operationName` - the operation name as a String.

Returns:

an array of Strings.

Throws:

NullPointerException - if operationName is null.

---

getPropertySource

`public PropertySource getPropertySource(RenderedOp op)`

Constructs and returns a PropertySource suitable for use by a given RenderedOp. The PropertySource includes properties copied from prior nodes as well as those generated at the node itself. Additionally, property suppression is taken into account. The actual implementation of getPropertySource() may make use of deferred execution and caching.

Parameters:

`op` - the RenderedOp requesting its PropertySource.

Throws:

NullPointerException - if op is null.

---

getPropertySource

`public PropertySource getPropertySource(RenderableOp op)`

Constructs and returns a PropertySource suitable for use by a given RenderableOp. The PropertySource includes properties copied from prior nodes as well as those generated at the node itself. Additionally, property suppression is taken into account. The actual implementation of getPropertySource() may make use of deferred execution and caching.

Parameters:

`op` - the RenderableOp requesting its PropertySource.

javax.media.jai
Class ParameterBlockJAI

```
java.lang.Object
  |
  +--java.awt.image.renderable.ParameterBlock
        |
        +--javax.media.jai.ParameterBlockJAI
```

public class **ParameterBlockJAI**
extends java.awt.image.renderable.ParameterBlock

A convenience subclass of ParameterBlock that allows the use of default parameter values and getting/setting parameters by name. A ParameterBlockJAI is constructed using either an OperationDescriptor, or an operation name that will be looked up in the appropriate (rendered or renderable) default OperationRegistry.

Once constructed, a ParameterBlockJAI appears to have no sources. It contains all the parameters required by its OperationDescriptor, each having its default value as given by the OperationDescriptor. Such a ParameterBlockJAI may not yet be usable, its sources (if any) are not set, and some or all of its parameters may have inapproriate values. The addSource methods of ParameterBlock may be used to set the source values, and the set(value, index) methods may be used to insert new parameter values. The add() methods should not be used since the parameter list is already long enough to hold all of the parameters required by the OperationDescriptor.

Additionally, ParameterBlockJAI offers set(value, name) methods that take a parameter name; the index of the parameter is determined from the OperationDescriptor and the corresponding parameter is set. As in ParameterBlock, all parameters are stored internally as subclasses of Object and all get/set methods that take or return values of base types are simply conveniences that transform values between the base types and their corresponding Number subclasses.

Field Detail

odesc

`private OperationDescriptor odesc`

The OperationDescriptor associated with this ParameterBlockJAI.

paramClasses

`private java.lang.Class[] paramClasses`

The Class types of the parameters.

indexTable

`private java.util.Hashtable indexTable`

A Hashtable mapping parameter names to their index.

Constructor Detail

ParameterBlockJAI

`public ParameterBlockJAI(OperationDescriptor odesc)`

Constructs a ParameterBlockJAI for use with an operation described by a particular OperationDescriptor. The default values of the parameters are filled in.

ParameterBlockJAI

`public ParameterBlockJAI(java.lang.String name)`

Constructs a ParameterBlockJAI for a particular operation by name. The `OperationRegistry` associated with the default instance of the `JAI` class is used to locate the `OperationDescriptor` associated with the operation name.
Parameters:
`name` - a String giving the name of the operation.

Method Detail

indexOf

```
public int indexOf(java.lang.String paramName)
```

Returns the index of a named parameter within the list of parameters, starting with 0.

Parameters:

`paramName` - a String containing the parameter name.

---

getOperationDescriptor

```
public OperationDescriptor getOperationDescriptor()
```

Returns the OperationDescriptor associated with this ParameterBlockJAI.

---

set

```
public java.awt.image.renderable.ParameterBlock set(byte b,
                                                    java.lang.String paramName)
```

Sets a named parameter to a byte value.

Parameters:

`paramName` - a String naming a parameter.
`b` - a byte value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(char c,
                                                    java.lang.String paramName)
```

Sets a named parameter to a char value.

Parameters:

`paramName` - a String naming a parameter.
`c` - a char value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(short s,
                                                    java.lang.String paramName)
```

Sets a named parameter to a short value.

Parameters:

`paramName` - a String naming a parameter.
`s` - a short value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(int i,
                                                    java.lang.String paramName)
```

Sets a named parameter to an int value.

Parameters:

`paramName` - a String naming a parameter.
`i` - an int value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(long l,
                                                    java.lang.String paramName)
```

Sets a named parameter to a long value.

Parameters:

`paramName` - a String naming a parameter.
`l` - a long value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(float f,
                                                    java.lang.String paramName)
```

Sets a named parameter to a float value.

Parameters:

`paramName` - a String naming a parameter.
`f` - a float value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(double d,
                                                    java.lang.String paramName)
```

Sets a named parameter to a double value.

Parameters:

`paramName` - a String naming a parameter.
`d` - a double value for the parameter.

---

set

```
public java.awt.image.renderable.ParameterBlock set(java.lang.Object obj,
                                                    java.lang.String paramName)
```

Sets a named parameter to an Object value.

Parameters:

`paramName` - a String naming a parameter.
`obj` - an Object value for the parameter.

---

getObjectParameter

```
public java.lang.Object getObjectParameter(java.lang.String paramName)
```

Gets a named parameter as an Object. Parameters belonging to a base type, such as int, will be returned as a member of the corresponding Number subclass, such as Integer.

---

getByteParameter

```
public byte getByteParameter(java.lang.String paramName)
```

A convenience method to return a parameter as a byte. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

---

getCharParameter

```
public char getCharParameter(java.lang.String paramName)
```

A convenience method to return a parameter as a char. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

---

getIntParameter

```
public int getIntParameter(java.lang.String paramName)
```

A convenience method to return a parameter as an int. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

---

getLongParameter

```
public long getLongParameter(java.lang.String paramName)
```

A convenience method to return a parameter as a long. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

getFloatParameter

```
public float getFloatParameter(java.lang.String paramName)
```

A convenience method to return a parameter as a float. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

getDoubleParameter

```
public double getDoubleParameter(java.lang.String paramName)
```

A convenience method to return a parameter as a double. An exception will be thrown if the parameter is of a different type.

Parameters:

`paramName` - the name of the parameter to be returned.

javax.media.jai

Class PartialOrderNode

```
java.lang.Object
  |
  +--javax.media.jai.PartialOrderNode
```

class **PartialOrderNode**
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable

A node in a directed graph of operations. Each node maintains three pieces of information, in addition to an arbitrary Object containing user data associated with the node, in order to allow topological sorting to be performed in linear time.

First, the in-degree (number of other nodes pointing to this node) is stored as an int. Nodes with in-degree equal to 0 are "free" and may appear first in a topological sort.

Second, a reference called zeroLink to another PartialOrderNode is kept in order to allow construction of a linked list of nodes with zero in-degree.

Third, a Vector of neighboring nodes is maintained (in no particular order). These are the nodes which are pointed to by the current node.

This class is used by the implementation of the OperationRegistry class and is not intended to be part of the API.

Field Detail

name

`protected java.lang.String name`

The name of the object associated with this node.

nodeData

`protected java.lang.Object nodeData`

The data associated with this node.

inDegree

`protected int inDegree`

The in-degree of the node.

copyInDegree

`protected int copyInDegree`

Copy of the inDegree of the node.

zeroLink

`protected PartialOrderNode zeroLink`

A link to another node with 0 in-degree, or null.

neighbors

`java.util.Vector neighbors`

A Vector of neighboring nodes.

Constructor Detail

PartialOrderNode

```
public PartialOrderNode(java.lang.Object nodeData,
                        java.lang.String name)
```

Constructs an PartialOrderNode with given associated data.

Parameters:

`nodeData` - an Object to associate with this node.

Method Detail

getData

```
public java.lang.Object getData()
```

Returns the Object represented by this node.

---

getName

```
public java.lang.String getName()
```

Returns the name of the Object represented by this node.

---

getInDegree

```
public int getInDegree()
```

Returns the in-degree of this node.

---

getCopyInDegree

```
public int getCopyInDegree()
```

Returns the copy in-degree of this node.

---

setCopyInDegree

```
public void setCopyInDegree(int copyInDegree)
```

Sets the copy in-degree of this node.

---

getZeroLink

```
public PartialOrderNode getZeroLink()
```

Returns the next zero in-degree node in the linked list.

---

setZeroLink

```
public void setZeroLink(PartialOrderNode poNode)
```

Sets the next zero in-degree node in the linked list.

---

getNeighbors

```
public java.util.Enumeration getNeighbors()
```

Returns the neighbors of this node as an enumeration.

---

addEdge

```
public void addEdge(PartialOrderNode poNode)
```

Adds a directed edge to the graph. The neighbors list of this node is updated and the in-degree of the other node is incremented.

---

removeEdge

```
public void removeEdge(PartialOrderNode poNode)
```

Removes a directed edge from the graph. The neighbors list of this node is updated and the in-degree of the other node is decremented.

---

incrementInDegree

```
public void incrementInDegree()
```

Increments the in-degree of a node.

---

incrementCopyInDegree

```
public void incrementCopyInDegree()
```

Increments the copy-in-degree of a node.

---

decrementInDegree

```
public void decrementInDegree()
```

Decrements the in-degree of a node.

---

decrementCopyInDegree

```
public void decrementCopyInDegree()
```

Decrements the copy in-degree of a node.

javax.media.jai

Class PerspectiveTransform

```
java.lang.Object
  |
  +--javax.media.jai.PerspectiveTransform
```

public final class **PerspectiveTransform**
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable

A 2D perspective (or projective) transform, used by various OpImages.

A perspective transformation is capable of mapping an arbitrary quadrilateral into another arbitrary quadrilateral, while preserving the straightness of lines. Unlike an affine transformation, the parallelism of lines in the source is not necessarily preserved in the output.

Such a coordinate transformation can be represented by a 3x3 matrix which transforms homogenous source coordinates `(x, y, 1)` into destination coordinates `(x′, y′, w)`. To convert back into non-homogenous coordinates (X, Y), `x′` and `y′` are divided by `w`.

```
        [ x′]   [  m00  m01  m02  ] [ x ]   [ m00x + m01y + m02 ]
        [ y′] = [  m10  m11  m12  ] [ y ] = [ m10x + m11y + m12 ]
        [ w ]   [  m20  m21  m22  ] [ 1 ]   [ m20x + m21y + m22 ]

          x′ = (m00x + m01y + m02)/(m20x + m21y + m22)
          y′ = (m10x + m11y + m12)/(m20x + m21y + m22)

        X = x′ / w
        Y = y′ / w
```

Field Detail

PERSPECTIVE_DIVIDE_EPSILON

`private static final double` **`PERSPECTIVE_DIVIDE_EPSILON`**

m00

`double` **`m00`**

An element of the transform matrix.

m01

`double` **`m01`**

An element of the transform matrix.

m02

`double` **`m02`**

An element of the transform matrix.

m10

`double` **`m10`**

An element of the transform matrix.

m11

`double` **`m11`**

An element of the transform matrix.

m12

`double m12`

An element of the transform matrix.

---

m20

`double m20`

An element of the transform matrix.

---

m21

`double m21`

An element of the transform matrix.

---

m22

`double m22`

An element of the transform matrix.

Constructor Detail

PerspectiveTransform

`public PerspectiveTransform()`

Constructs an identity PerspectiveTransform.

PerspectiveTransform

```
public PerspectiveTransform(float m00,
                            float m01,
                            float m02,
                            float m10,
                            float m11,
                            float m12,
                            float m20,
                            float m21,
                            float m22)
```

Constructs a new PerspectiveTransform from 9 floats.

PerspectiveTransform

```
public PerspectiveTransform(double m00,
                            double m01,
                            double m02,
                            double m10,
                            double m11,
                            double m12,
                            double m20,
                            double m21,
                            double m22)
```

Constructs a new PerspectiveTransform from 9 doubles.

PerspectiveTransform

`public PerspectiveTransform(float[] flatmatrix)`

Constructs a new PerspectiveTransform from a one-dimensional array of 9 floats, in row-major order. The values in the array are assumed to be { m00 m01 m02 m10 m11 m12 m20 m21 m22 }.

Throws:

NullPointerException - if flatmatrix is null

ArrayBoundsException - if flatmatrix is too small

PerspectiveTransform

```
public PerspectiveTransform(float[][] matrix)
```

Constructs a new PerspectiveTransform from a two-dimensional array of floats.

Throws:

NullPointerException - if matrix is null
ArrayBoundsException - if matrix is too small

PerspectiveTransform

```
public PerspectiveTransform(double[] flatmatrix)
```

Constructs a new PerspectiveTransform from a one-dimensional array of 9 doubles, in row-major order. The values in the array are assumed to be { m00 m01 m02 m10 m11 m12 m20 m21 m22 }.

Throws:

NullPointerException - if flatmatrix is null
ArrayBoundsException - if flatmatrix is too small

PerspectiveTransform

```
public PerspectiveTransform(double[][] matrix)
```

Constructs a new PerspectiveTransform from a two-dimensional array of doubles.

Throws:

NullPointerException - if matrix is null
ArrayBoundsException - if matrix is too small

PerspectiveTransform

```
public PerspectiveTransform(java.awt.geom.AffineTransform transform)
```

Constructs a new PerspectiveTransform with the same effect as an existing AffineTransform.

Throws:

NullPointerException - if transform is null

Method Detail

makeAdjoint

```
private final void makeAdjoint()
```

Replaces the matrix with its adjoint.

---

normalize

```
private final void normalize()
```

Scales the matrix elements so m22 is equal to 1.0. m22 must not be equal to 0.

---

getSquareToQuad

```
private static final void getSquareToQuad(double x0,
                                          double y0,
                                          double x1,
                                          double y1,
                                          double x2,
                                          double y2,
                                          double x3,
                                          double y3,
                                          PerspectiveTransform tx)
```

---

getSquareToQuad

```
public static PerspectiveTransform getSquareToQuad(double x0,
                                                   double y0,
                                                   double x1,
                                                   double y1,
                                                   double x2,
                                                   double y2,
                                                   double x3,
                                                   double y3)
```

Creates a PerspectiveTransform that maps the unit square onto an arbitrary quadrilateral.

```
(0, 0) -> (x0, y0)
(1, 0) -> (x1, y1)
(1, 1) -> (x2, y2)
(0, 1) -> (x3, y3)
```

---

getSquareToQuad

```
public static PerspectiveTransform getSquareToQuad(float x0,
                                                    float y0,
                                                    float x1,
                                                    float y1,
                                                    float x2,
                                                    float y2,
                                                    float x3,
                                                    float y3)
```

Creates a PerspectiveTransform that maps the unit square onto an arbitrary quadrilateral.

```
(0, 0) -> (x0, y0)
(1, 0) -> (x1, y1)
(1, 1) -> (x2, y2)
(0, 1) -> (x3, y3)
```

---

getQuadToSquare

```
public static PerspectiveTransform getQuadToSquare(double x0,
                                                    double y0,
                                                    double x1,
                                                    double y1,
                                                    double x2,
                                                    double y2,
                                                    double x3,
                                                    double y3)
```

Creates a PerspectiveTransform that maps an arbitrary quadrilateral onto the unit square.

```
(x0, y0) -> (0, 0)
(x1, y1) -> (1, 0)
(x2, y2) -> (1, 1)
(x3, y3) -> (0, 1)
```

---

getQuadToSquare

```
public static PerspectiveTransform getQuadToSquare(float x0,
                                                    float y0,
                                                    float x1,
                                                    float y1,
                                                    float x2,
                                                    float y2,
                                                    float x3,
                                                    float y3)
```

Creates a PerspectiveTransform that maps an arbitrary quadrilateral onto the unit square.

```
(x0, y0) -> (0, 0)
(x1, y1) -> (1, 0)
(x2, y2) -> (1, 1)
(x3, y3) -> (0, 1)
```

---

getQuadToQuad

```
public static PerspectiveTransform getQuadToQuad(double x0,
                                                  double y0,
                                                  double x1,
                                                  double y1,
                                                  double x2,
                                                  double y2,
                                                  double x3,
                                                  double y3,
                                                  double x0p,
                                                  double y0p,
                                                  double x1p,
                                                  double y1p,
                                                  double x2p,
                                                  double y2p,
                                                  double x3p,
                                                  double y3p)
```

Creates a PerspectiveTransform that maps an arbitrary quadrilateral onto another arbitrary quadrilateral.

```
 (x0, y0) -> (x0p, y0p)
 (x1, y1) -> (x1p, y1p)
 (x2, y2) -> (x2p, y2p)
 (x3, y3) -> (x3p, y3p)
```

---

getQuadToQuad

```
public static PerspectiveTransform getQuadToQuad(float x0,
                                                  float y0,
                                                  float x1,
                                                  float y1,
                                                  float x2,
                                                  float y2,
                                                  float x3,
                                                  float y3,
                                                  float x0p,
                                                  float y0p,
                                                  float x1p,
                                                  float y1p,
                                                  float x2p,
                                                  float y2p,
                                                  float x3p,
                                                  float y3p)
```

Creates a PerspectiveTransform that maps an arbitrary quadrilateral onto another arbitrary quadrilateral.

```
 (x0, y0) -> (x0p, y0p)
 (x1, y1) -> (x1p, y1p)
 (x2, y2) -> (x2p, y2p)
 (x3, y3) -> (x3p, y3p)
```

---

getDeterminant

```
public double getDeterminant()
```

Returns the determinant of the matrix representation of the transform.

---

getMatrix

```
public double[] getMatrix(double[] flatmatrix)
```

Retrieves the 9 specifiable values in the 3x3 affine transformation matrix into an array of double precision values. The values are stored into the array as { m00 m01 m02 m10 m11 m12 m20 m21 m22 }.

Parameters:

`flatmatrix` - The double array used to store the returned values. The length of the array is assumed to be at least 9.

Throws:

ArrayBoundsException - if flatmatrix is too small

---

getMatrix

```
public double[][] getMatrix(double[][] matrix)
```

Retrieves the 9 specifiable values in the 3x3 affine transformation matrix into a 2-dimensional array of double precision values. The values are stored into the 2-dimensional array using the row index as the first subscript and the column index as the second.

Parameters:

`matrix` - The 2-dimensional double array to store the returned values. The array is assumed to be at least 3x3.

Throws:

ArrayBoundsException - if matrix is too small

---

translate

```
public void translate(double tx,
                      double ty)
```

Concatenates this transform with a translation transformation. This is equivalent to calling concatenate(T), where T is an PerspectiveTransform represented by the following matrix:

```
[   1    0    tx  ]
[   0    1    ty  ]
[   0    0    1   ]
```

---

rotate

```
public void rotate(double theta)
```

Concatenates this transform with a rotation transformation. This is equivalent to calling concatenate(R), where R is an PerspectiveTransform represented by the following matrix:

```
[   cos(theta)    -sin(theta)    0   ]
[   sin(theta)     cos(theta)    0   ]
[       0              0         1   ]
```

Rotating with a positive angle theta rotates points on the positive X axis toward the positive Y axis.

Parameters:

`theta` - The angle of rotation in radians.

---

rotate

```
public void rotate(double theta,
                   double x,
                   double y)
```

Concatenates this transform with a translated rotation transformation. This is equivalent to the following sequence of calls:

```
translate(x, y);
rotate(theta);
translate(-x, -y);
```

Rotating with a positive angle theta rotates points on the positive X axis toward the positive Y axis.

Parameters:

`theta` - The angle of rotation in radians.
`x` - The X coordinate of the origin of the rotation
`y` - The Y coordinate of the origin of the rotation

---

scale

```
public void scale(double sx,
                  double sy)
```

Concatenates this transform with a scaling transformation. This is equivalent to calling concatenate(S), where S is an PerspectiveTransform represented by the following matrix:

```
[   sx   0    0   ]
[   0    sy   0   ]
[   0    0    1   ]
```

Parameters:

`sx` - The X axis scale factor.
`sy` - The Y axis scale factor.

---

shear

```
public void shear(double shx,
                  double shy)
```

Concatenates this transform with a shearing transformation. This is equivalent to calling concatenate(SH), where SH is an PerspectiveTransform represented by the following matrix:

```
[   1   shx   0   ]
[  shy   1    0   ]
[   0    0    1   ]
```

Parameters:

`shx` - The factor by which coordinates are shifted towards the positive X axis direction according to their Y coordinate.
`shy` - The factor by which coordinates are shifted towards the positive Y axis direction according to their X coordinate.

setToIdentity

```
public void setToIdentity()
```

Resets this transform to the Identity transform.

setToTranslation

```
public void setToTranslation(double tx,
                             double ty)
```

Sets this transform to a translation transformation. The matrix representing this transform becomes:

```
[   1    0    tx  ]
[   0    1    ty  ]
[   0    0    1   ]
```

Parameters:

`tx` - The distance by which coordinates are translated in the X axis direction
`ty` - The distance by which coordinates are translated in the Y axis direction

setToRotation

```
public void setToRotation(double theta)
```

Sets this transform to a rotation transformation. The matrix representing this transform becomes:

```
[   cos(theta)    -sin(theta)    0   ]
[   sin(theta)     cos(theta)    0   ]
[       0              0         1   ]
```

Rotating with a positive angle theta rotates points on the positive X axis toward the positive Y axis.

Parameters:

`theta` - The angle of rotation in radians.

setToRotation

```
public void setToRotation(double theta,
                          double x,
                          double y)
```

Sets this transform to a rotation transformation about a specified point (x, y). This is equivalent to the following sequence of calls:

```
setToTranslate(x, y);
rotate(theta);
translate(-x, -y);
```

Rotating with a positive angle theta rotates points on the positive X axis toward the positive Y axis.

Parameters:

`theta` - The angle of rotation in radians.
`x` - The X coordinate of the origin of the rotation
`y` - The Y coordinate of the origin of the rotation

setToScale

```
public void setToScale(double sx,
                       double sy)
```

Sets this transform to a scale transformation with scale factors sx and sy. The matrix representing this transform becomes:

```
[   sx   0    0   ]
[   0    sy   0   ]
[   0    0    1   ]
```

Parameters:

`sx` - The X axis scale factor.
`sy` - The Y axis scale factor.

setToShear

```
public void setToShear(double shx,
                       double shy)
```

Sets this transform to a shearing transformation with shear factors sx and sy. The matrix representing this transform becomes:

```
[   1  shx    0   ]
[ shy    1    0   ]
[   0    0    1   ]
```

Parameters:

`shx` - The factor by which coordinates are shifted towards the positive X axis direction according to their Y coordinate.
`shy` - The factor by which coordinates are shifted towards the positive Y axis direction according to their X coordinate.

---

setTransform

```
public void setTransform(java.awt.geom.AffineTransform Tx)
```

Sets this transform to a given AffineTransform.

Throws:

NullPointerException - if Tx is null

---

setTransform

```
public void setTransform(PerspectiveTransform Tx)
```

Sets this transform to a given PerspectiveTransform.

Throws:

NullPointerException - if Tx is null

---

setTransform

```
public void setTransform(float m00,
                         float m10,
                         float m20,
                         float m01,
                         float m11,
                         float m21,
                         float m02,
                         float m12,
                         float m22)
```

Sets this transform to a given PerspectiveTransform, expressed by the elements of its matrix.

---

concatenate

```
public void concatenate(java.awt.geom.AffineTransform Tx)
```

Post-concatenates a given AffineTransform to this transform.

Throws:

NullPointerException - if Tx is null

---

concatenate

```
public void concatenate(PerspectiveTransform Tx)
```

Post-concatenates a given PerspectiveTransform to this transform.

Throws:

NullPointerException - if Tx is null

---

preConcatenate

```
public void preConcatenate(java.awt.geom.AffineTransform Tx)
```

Pre-concatenates a given AffineTransform to this transform.

Throws:

NullPointerException - if Tx is null

---

preConcatenate

```
public void preConcatenate(PerspectiveTransform Tx)
```

Pre-concatenates a given PerspectiveTransform to this transform.

Throws:

NullPointerException - if Tx is null

createInverse

```
public PerspectiveTransform createInverse()
                                   throws java.awt.geom.NoninvertibleTransformException
```

Returns a new PerpectiveTransform that is the inverse of the current transform.

Throws:

java.awt.geom.NoninvertibleTransformException - if transform cannot be inverted

createAdjoint

```
public PerspectiveTransform createAdjoint()
```

Returns a new PerpectiveTransform that is the adjoint, of the current transform. The adjoint is defined as the matrix of cofactors, which in turn are the determinants of the submatrices defined by removing the row and column of each element from the original matrix in turn.

The adjoint is a scalar multiple of the inverse matrix. Because points to be transformed are converted into homogeneous coordinates, where scalar factors are irrelevant, the adjoint may be used in place of the true inverse. Since it is unnecessary to normalize the adjoint, it is both faster to compute and more numerically stable than the true inverse.

transform

```
public java.awt.geom.Point2D transform(java.awt.geom.Point2D ptSrc,
                                       java.awt.geom.Point2D ptDst)
```

Transforms the specified ptSrc and stores the result in ptDst. If ptDst is null, a new Point2D object will be allocated before storing. In either case, ptDst containing the transformed point is returned for convenience. Note that ptSrc and ptDst can the same. In this case, the input point will be overwritten with the transformed point.

Parameters:

`ptSrc` - The array containing the source point objects.
`ptDst` - The array where the transform point objects are returned.

Throws:

NullPointerException - if ptSrc is null

transform

```
public void transform(java.awt.geom.Point2D[] ptSrc,
                      int srcOff,
                      java.awt.geom.Point2D[] ptDst,
                      int dstOff,
                      int numPts)
```

Transforms an array of point objects by this transform.

Parameters:

`ptSrc` - The array containing the source point objects.
`ptDst` - The array where the transform point objects are returned.
`srcOff` - The offset to the first point object to be transformed in the source array.
`dstOff` - The offset to the location where the first transformed point object is stored in the destination array.
`numPts` - The number of point objects to be transformed.

Throws:

NullPointerException - if ptSrc is null
ArrayBoundsException - if ptSrc is too small

transform

```
public void transform(float[] srcPts,
                      int srcOff,
                      float[] dstPts,
                      int dstOff,
                      int numPts)
```

Transforms an array of floating point coordinates by this transform.

Parameters:

`srcPts` - The array containing the source point coordinates. Each point is stored as a pair of x,y coordinates.
`srcOff` - The offset to the first point to be transformed in the source array.
`dstPts` - The array where the transformed point coordinates are returned. Each point is stored as a pair of x,y coordinates.
`dstOff` - The offset to the location where the first transformed point is stored in the destination array.
`numPts` - The number of points to be transformed.

Throws:
NullPointerException - if srcPts is null
ArrayBoundsException - if srcPts is too small

---

transform

```
public void transform(double[] srcPts,
                      int srcOff,
                      double[] dstPts,
                      int dstOff,
                      int numPts)
```

Transforms an array of double precision coordinates by this transform.

Parameters:
`srcPts` - The array containing the source point coordinates. Each point is stored as a pair of x,y coordinates.
`dstPts` - The array where the transformed point coordinates are returned. Each point is stored as a pair of x,y coordinates.
`srcOff` - The offset to the first point to be transformed in the source array.
`dstOff` - The offset to the location where the first transformed point is stored in the destination array.
`numPts` - The number of point objects to be transformed.

Throws:
NullPointerException - if srcPts is null
ArrayBoundsException - if srcPts is too small

---

transform

```
public void transform(float[] srcPts,
                      int srcOff,
                      double[] dstPts,
                      int dstOff,
                      int numPts)
```

Transforms an array of floating point coordinates by this transform, storing the results into an array of doubles.

Parameters:
`srcPts` - The array containing the source point coordinates. Each point is stored as a pair of x,y coordinates.
`srcOff` - The offset to the first point to be transformed in the source array.
`dstPts` - The array where the transformed point coordinates are returned. Each point is stored as a pair of x,y coordinates.
`dstOff` - The offset to the location where the first transformed point is stored in the destination array.
`numPts` - The number of points to be transformed.

Throws:
NullPointerException - if srcPts is null
ArrayBoundsException - if srcPts is too small

---

transform

```
public void transform(double[] srcPts,
                      int srcOff,
                      float[] dstPts,
                      int dstOff,
                      int numPts)
```

Transforms an array of double precision coordinates by this transform, storing the results into an array of floats.

Parameters:
`srcPts` - The array containing the source point coordinates. Each point is stored as a pair of x,y coordinates.
`dstPts` - The array where the transformed point coordinates are returned. Each point is stored as a pair of x,y coordinates.
`srcOff` - The offset to the first point to be transformed in the source array.
`dstOff` - The offset to the location where the first transformed point is stored in the destination array.
`numPts` - The number of point objects to be transformed.

Throws:
NullPointerException - if srcPts is null
ArrayBoundsException - if srcPts is too small

---

inverseTransform

```
public java.awt.geom.Point2D inverseTransform(java.awt.geom.Point2D ptSrc,
                                              java.awt.geom.Point2D ptDst)
                                       throws java.awt.geom.NoninvertibleTransformException
```

Inverse transforms the specified ptSrc and stores the result in ptDst. If ptDst is null, a new Point2D object will be allocated before storing. In either case, ptDst containing the transformed point is returned for convenience. Note that ptSrc and ptDst can the same. In this case, the input point will be overwritten with the transformed point.

Parameters:

`ptSrc` - The point to be inverse transformed.
`ptDst` - The resulting transformed point.

Throws:

java.awt.geom.NoninvertibleTransformException - if the matrix cannot be inverted.
NullPointerException - if ptSrc is null

---

inverseTransform

```
public void inverseTransform(double[] srcPts,
                             int srcOff,
                             double[] dstPts,
                             int dstOff,
                             int numPts)
                      throws java.awt.geom.NoninvertibleTransformException
```

Inverse transforms an array of double precision coordinates by this transform.

Parameters:

`srcPts` - The array containing the source point coordinates. Each point is stored as a pair of x,y coordinates.
`dstPts` - The array where the transformed point coordinates are returned. Each point is stored as a pair of x,y coordinates.
`srcOff` - The offset to the first point to be transformed in the source array.
`dstOff` - The offset to the location where the first transformed point is stored in the destination array.
`numPts` - The number of point objects to be transformed.

Throws:

java.awt.geom.NoninvertibleTransformException - if the matrix cannot be inverted.
NullPointerException - if srcPts is null
ArrayBoundsException - if srcPts is too small
java.awt.geom.NoninvertibleTransformException - transform cannot be inverted

---

toString

```
public java.lang.String toString()
```

Returns a String that represents the value of this Object.

Overrides:

toString in class java.lang.Object

---

isIdentity

```
public boolean isIdentity()
```

Returns the boolean true value if this PerspectiveTransform is an identity transform. Returns false otherwise.

---

clone

```
public java.lang.Object clone()
```

Returns a copy of this PerspectiveTransform object.

Overrides:

clone in class java.lang.Object

---

equals

```
public boolean equals(java.lang.Object obj)
```

Tests if this PerspectiveTransform equals a supplied one.

Parameters:

`obj` - The PerspectiveTransform to be compared to this one.

Throws:

NullPointerException - if the supplied object is null

Overrides:

equals in class java.lang.Object

javax.media.jai

Class PlanarImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
```

Direct Known Subclasses:

OpImage, RemoteImage, RenderedImageAdapter, RenderedOp, Snapshot, SnapshotImage, SnapshotProxy, TiledImage

---

public abstract class **PlanarImage**
extends java.lang.Object
implements ImageJAI, java.awt.image.RenderedImage

The fundamental base class representing two-dimensional images.

The `PlanarImage` class provides a home for the functionality common to the JAI classes that implement the `RenderedImage` interface, including TiledImage and `OpImage`. These subclasses manipulate the instance variables they inherit from `PlanarImage`, such as the image size, origin, tile dimensions, and tile grid offsets, as well as lists containing the sources and sinks of the image. With these instance variables properly defined, most of the method calls mandated by `RenderedImage` are correctly (if not necessarily optimally) implemented at this level.

Subclasses are responsible for initializing all of the protected instance variables prior to allowing any calls to non-static methods. `PlanarImage` does not perform sanity checking on the state of its instance variables.

`PlanarImage` implements a `createSnapshot` method that produces a new, immutable image with a copy of the source image's current contents. In practice, this snapshot is only a virtual copy; it is managed by the `SnapshotImage` class in such a way as to minimize copying and memory footprint generally. Multiple calls to `createSnapshot` make use of a single `SnapshotImage` per `PlanarImage` in order to centralize version management. These mechanisms are transparent to the API user and are discussed here only for edification.

All non-JAI `RenderedImage` instances must be converted into `PlanarImage`s by means of the `RenderedImageAdapter` and `WritableRenderedImageAdapter` classes. The `wrapRenderedImage` method provides a convenient interface to both add a wrapper and take a snapshot if the image is writable. The standard `PlanarImage` constructor used by `OpImage`s performs this wrapping automatically. Images that already extend `PlanarImage` will be returned unchanged by `wrapRenderedImage`, that is, it is idempotent.

Going in the other direction, existing code that makes use of the `RenderedImage` interface will be able to use `PlanarImage`s directly, without any changes or recompilation. Therefore within JAI images are returned from methods as `PlanarImage`s, even though incoming `RenderedImages` are accepted as arguments directly.

The source and sink lists have the effect of creating a graph structure between a set of `PlanarImage`s. Note that the practice of making such bidirectional connections between images means that the garbage collector will not inform us when all user references to a node are lost, since there will still be internal references up until the point where the entire graph is detached from user space. A solution is available in the form of *Reference Objects*; see http://java.sun.com/products/jdk/1.2/docs/guide/refobs/ for more information. These classes include *weak references* that allow the GC to collect objects they reference, setting the reference to `null` in the process.

The reference problem requires us to be careful about how we define the *reachability* of DAG nodes. If we were to allow nodes to be reached by arbitrary graph traversal, we would be unable to garbage collect any subgraphs of an active graph at all since any node may be reached from any other. Instead, we define the set of reachable nodes as those that may be accessed directly from a reference in user code, or that are the source (not sink) of a reachable node. Reachable nodes are always accessible, whether they are reached by traversing upwards or downwards in the DAG.

A DAG may also contain nodes that are not reachable, that is, they require a downward traversal at some point. Say a node `A` is reachable, and a call to `A.getSinks()` yields a `Vector` containing a reference to a previously unreachable node `B`. The node `B` naturally becomes reachable by virtue of the new user reference pointing to it. However, if the user were to relinquish that reference, the node might be garbage collected, and a future call to `A.getSinks()` might no longer include `B` in its return value.

Because the set of sinks of a node is inherently unstable, only the `getSinks` method is provided for external access to the sink vector at a node. A hypothetical method such as `getSink` or `getNumSinks` would produce confusing results should a sink be garbage collected between that call and a subsequent call to `getSinks`.

The dimensions and tile grid layout of an image may be completely specified by the instance variables `minX`, `minY`, `width`, `height`, `tileWidth`, `tileHeight`, `tileGridXOffset`, and `tileGridYOffset`. The accessor methods returning these values simply return the values of the corresponding instance variable. All other accessor methods derive their values from these "primitive" accessor methods. This implies that a subclass may set its instance variables at construction time and implement none of the accessor methods, or else may provide implementations of the primitive accessors and simply inherit the others.

See Also:

`Reference`, `WeakReference`, `RenderedImage`, `OpImage`, `RenderedImageAdapter`, `SnapshotImage`, `TiledImage`

Field Detail

minX

`protected int` **`minX`**

The X coordinate of the image’s upper-left pixel.

minY

`protected int` **`minY`**

The Y coordinate of the image’s upper-left pixel.

width

`protected int` **`width`**

The image’s width in pixels.

height

`protected int` **`height`**

The image’s height in pixels.

tileWidth

`protected int` **`tileWidth`**

The width of a tile.

tileHeight

`protected int` **`tileHeight`**

The height of a tile.

tileGridXOffset

`protected int` **`tileGridXOffset`**

The X coordinate of the upper-left pixel of tile (0, 0).

tileGridYOffset

`protected int` **`tileGridYOffset`**

The Y coordinate of the upper-left pixel of tile (0, 0).

sampleModel

`protected java.awt.image.SampleModel` **`sampleModel`**

The image’s `SampleModel`.

colorModel

`protected java.awt.image.ColorModel` **`colorModel`**

The image’s `ColorModel`.

snapshot

`protected SnapshotImage` **`snapshot`**

A `SnapshotImage` that will centralize tile versioning for this image.

---

properties

`private java.util.Hashtable` **`properties`**

A `Hashtable` containing the image properties.

MIN_ARRAYCOPY_SIZE

`private static final int` **`MIN_ARRAYCOPY_SIZE`**

---

weakThis

`private java.lang.ref.WeakReference` **`weakThis`**

A `WeakReference` to this image.

---

sinks

`private java.util.AbstractList` **`sinks`**

A set of `WeakReference`s to the image's sinks.

---

defaultColorModels

`private static final java.awt.image.ColorModel[][]` **`defaultColorModels`**

---

source0

`protected PlanarImage` **`source0`**

The image's first source, stored separately for convenience. `source0` will be `null` for images that have no sources.

---

source1

`protected PlanarImage` **`source1`**

The image's second source, stored separately for convenience. `source0` will be `null` for images that have no or one source.

---

sources

`protected java.util.Vector` **`sources`**

The image's third and later sources, stored in a Vector.

---

disposed

`private boolean` **`disposed`**

Constructor Detail

PlanarImage

`public` **`PlanarImage`**`()`

The default constructor.

Method Detail

setImageParameters

```
protected void setImageParameters(ImageLayout layout,
                                  java.awt.image.RenderedImage im)
```

Sets the image bounds, tile grid layout, `SampleModel` and `ColorModel` to match those of another image, overriding the image's values with values from an `ImageLayout` object. This method should only be called during the image construction process.

The image min coordinates, width, height, tile grid offsets, tile width, tile height, `SampleModel`, and `ColorModel` are taken either from the image or from the layout.

Parameters:

`layout` - an ImageLayout that is used to selectively override the image's layout, `SampleModel`, and `ColorModel`. If `null`, all parameters will be taken from the image argument.

`im` - a `RenderedImage` used as the basis for the layout.

Throws:

java.lang.IllegalArgumentException - if `im` is `null`.

---

setImageParameters

```
protected void setImageParameters(java.awt.image.RenderedImage im)
```

Sets the image bounds, tile grid layout, `SampleModel` and `ColorModel` to match those of another image. This method should only be called during the image construction process.

Parameters:

`im` - a `RenderedImage` used as the basis for the layout.

Throws:

java.lang.IllegalArgumentException - if `im` is `null`.

---

wrapRenderedImage

```
public static PlanarImage wrapRenderedImage(java.awt.image.RenderedImage im)
```

Wraps an arbitrary `RenderedImage` to produce a `PlanarImage`. `PlanarImage` adds various properties to an image, such as source and sink vectors and the ability to produce snapshots, that are necessary for JAI.

If the image is already a `PlanarImage`, it is simply returned unchanged. Otherwise, the image is wrapped in a `RenderedImageAdapter` or `WritableRenderedImageAdapter` as appropriate.

Parameters:

`im` - a `RenderedImage` to be used as a source.

Returns:

a `PlanarImage` containing the source's pixel data.

Throws:

java.lang.IllegalArgumentException - if `im` is `null`.

---

createSnapshot

```
public PlanarImage createSnapshot()
```

Creates a snapshot, that is, a virtual copy of the image's current contents. If the image is not a `WritableRenderedImage`, it is returned unchanged. Otherwise, a `SnapshotImage` is created and the result of calling its `createSnapshot()` is returned.

Returns:

a `PlanarImage` with immutable contents.

---

getMinX

```
public int getMinX()
```

Returns the X coordinate of the leftmost column of the image.

Specified by:

getMinX in interface java.awt.image.RenderedImage

---

getMaxX

```
public int getMaxX()
```

Returns the X coordinate of the column immediately to the right of the rightmost column of the image. `getMaxX` is implemented directly in terms of the instance variables `minX` and `width`; therefore subclasses that override `getMinX()` or `getWidth()` must also override this method.

getMinY

`public int getMinY()`

Returns the Y coordinate of the uppermost row of the image.

Specified by:

getMinY in interface java.awt.image.RenderedImage

getMaxY

`public int getMaxY()`

Returns the Y coordinate of the row immediately below the bottom row of the image. `getMaxY` is implemented directly in terms of the instance variables `minY` and `height`; therefore subclasses that override `getMinY()` or `getHeight()` must also override this method.

getWidth

`public int getWidth()`

Returns the width of the image.

Specified by:

getWidth in interface java.awt.image.RenderedImage

getHeight

`public int getHeight()`

Returns the height of the image.

Specified by:

getHeight in interface java.awt.image.RenderedImage

getTileWidth

`public int getTileWidth()`

Returns the width of a tile.

Specified by:

getTileWidth in interface java.awt.image.RenderedImage

getTileHeight

`public int getTileHeight()`

Returns the height of a tile.

Specified by:

getTileHeight in interface java.awt.image.RenderedImage

getTileGridXOffset

`public int getTileGridXOffset()`

Returns the X coordinate of the upper-left pixel of tile (0, 0).

Specified by:

getTileGridXOffset in interface java.awt.image.RenderedImage

getTileGridYOffset

`public int getTileGridYOffset()`

Returns the Y coordinate of the upper-left pixel of tile (0, 0).

Specified by:

getTileGridYOffset in interface java.awt.image.RenderedImage

getMinTileX

`public int getMinTileX()`

Returns the horizontal index of the leftmost column of tiles. `getMinTileX` is implemented as `XToTileX(getMinX())` and so does not need to be implemented by subclasses.

Specified by:

getMinTileX in interface java.awt.image.RenderedImage

---

getMaxTileX

`public int getMaxTileX()`

Returns the horizontal index of the rightmost column of tiles. `getMaxTileX` is implemented as `XToTileX(getMaxX() - 1)` and so does not need to be implemented by subclasses.

---

getNumXTiles

`public int getNumXTiles()`

Returns the number of tiles along the tile grid in the horizontal direction. `getNumXTiles` is implemented as `getMaxTileX() - getMinTileX() + 1` and so does not need to be implemented by subclasses.

Specified by:

getNumXTiles in interface java.awt.image.RenderedImage

---

getMinTileY

`public int getMinTileY()`

Returns the vertical index of the uppermost row of tiles. `getMinTileY` is implemented as `YToTileY(getMinY())` and so does not need to be implemented by subclasses.

Specified by:

getMinTileY in interface java.awt.image.RenderedImage

---

getMaxTileY

`public int getMaxTileY()`

Returns the vertical index of the bottom row of tiles. `getMaxTileY` is implemented as `YToTileY(getMaxY() - 1)` and so does not need to be implemented by subclasses.

---

getNumYTiles

`public int getNumYTiles()`

Returns the number of tiles along the tile grid in the vertical direction. `getNumYTiles` is implemented as `getMaxTileY() - getMinTileY() + 1` and so does not need to be implemented by subclasses.

Specified by:

getNumYTiles in interface java.awt.image.RenderedImage

---

getSampleModel

`public java.awt.image.SampleModel getSampleModel()`

Returns the `SampleModel` of the image.

Specified by:

getSampleModel in interface java.awt.image.RenderedImage

---

getColorModel

`public java.awt.image.ColorModel getColorModel()`

Returns the `ColorModel` of the image.

Specified by:

getColorModel in interface java.awt.image.RenderedImage

```
static void ()
```

---

createColorModel

```
public static java.awt.image.ColorModel createColorModel(java.awt.image.SampleModel sm)
```

Creates a `ColorModel` that may be used with the specified `SampleModel`. If the specified `SampleModel` is `null`, this method will throw an `IllegalArgumentException`. If a suitable `ColorModel` cannot be found, this method will return `null`.

Suitable `ColorModel`s are guaranteed to exist for all instances of `ComponentSampleModel` with no more than 4 bands.

Additionally, a `DirectColorModel` instance will be created for instances of `SinglePixelPackedSampleModel` with no more than 4 bands.

For 1- and 3- banded `SampleModel`s, the returned `ColorModel` will be opaque. For 2- and 4-banded `SampleModel`s, the output will use alpha transparency.

This method is called from the `OpImage` constructor to supply a `ColorModel` for images where none has been specified via tha `ImageLayout` parameter.

This method is intended as a useful utility for the creation of simple `ColorModel`s for some common cases. In more complex situations, it may be necessary to instantiate appropriate `ColorModel`s directly.

Returns:
an instance of `ColorModel`, or `null`.

Throws:
java.lang.IllegalArgumentException - if `sm` is `null`.

---

getBounds

```
public java.awt.Rectangle getBounds()
```

Returns a `Rectangle` indicating the image bounds.

---

getSource

```
public PlanarImage getSource(int index)
```

Returns an entry from the list of sources. If there is no source corresponding to the specified index, this method will throw an `ArrayIndexOutOfBoundsException`.

Parameters:
`index` - The index of the desired source.

Returns:
A `PlanarImage` source.

Throws:
ArrayIndexOutOfBoundsException - if the index is negative or greater than the maximum source index.

---

setSource

```
void setSource(PlanarImage source,
               int index)
```

Helper for RenderedOp.setSource().

---

addSource

```
protected void addSource(PlanarImage source)
```

Adds a `PlanarImage` source to the list of sources.

Parameters:
`source` - A `PlanarImage` to be added as a source.

Throws:
java.lang.IllegalArgumentException - if `source` is `null`.

---

addSink

```
protected void addSink(PlanarImage sink)
```

Adds a `PlanarImage` sink to the list of sinks.

Parameters:
`sink` - A `PlanarImage` to be added as a sink.

Throws:
java.lang.IllegalArgumentException - if `sink` is `null`.

---

removeSource

`protected boolean **removeSource**(PlanarImage source)`

Removes a `PlanarImage` source from the list of sources.

Parameters:
`source` - A `PlanarImage` to be removed.

Returns:
`true` if the element was present, false otherwise.

Throws:
java.lang.IllegalArgumentException - if `source` is `null`.

---

removeSink

`protected boolean **removeSink**(PlanarImage sink)`

Removes a `PlanarImage` sink from the list of sinks.

Parameters:
`sink` - a `PlanarImage` to be removed.

Returns:
true if the element was present, `false` otherwise.

Throws:
java.lang.IllegalArgumentException - if `sink` is `null`.

---

getSources

`public java.util.Vector **getSources**()`

Returns this image’s source(s) in a `Vector`.

Specified by:
getSources in interface java.awt.image.RenderedImage

---

getNumSources

`public int **getNumSources**()`

Returns the number of `PlanarImage` sources.

---

getSinks

`public java.util.Vector **getSinks**()`

Returns a `Vector` containing the currently available `PlanarImage` sinks of this image (images for which this image is a source), or `null` if no sinks are present.

Sinks are stored using weak references. This means that the set of sinks may change between calls to `getSinks()` if the garbage collector happens to identify a sink as not otherwise reachable (reachability is discussed in the class comments for this class).

Since the pool of sinks may change as garbage collection occurs, `PlanarImage` does not implement either a `getSink(int index)` or a `getNumSinks()` method. Instead, the caller must call `getSinks()`, which returns a Vector of normal references. As long as the returned `Vector` is referenced from user code, the images it references are reachable and may be reliably accessed.

---

setSources

`protected void **setSources**(java.util.List sourceList)`

Set the list of sources from a given `List` of `PlanarImage`s.

Parameters:
`sourceList` - a `List` of `PlanarImage`s.

---

removeSources

`protected void **removeSources**()`

Clears the list of sources.

---

removeSinks

`protected void **removeSinks**()`

Clears the list of sinks.

---

getProperties

`protected java.util.Hashtable **getProperties**()`

Returns the internal `Hashtable` containing the image properties.

---

setProperties

`protected void **setProperties**(java.util.Hashtable properties)`

Sets the `Hashtable` containing the image properties to a given `Hashtable`. The `Hashtable` is incorporated by reference and must not be altered by other classes after this method is called.

---

getProperty

`public java.lang.Object **getProperty**(java.lang.String name)`

Gets a property from the property set of this image. If the property name is not recognized, `java.awt.Image.UndefinedProperty` will be returned.

Specified by:

getProperty in interface java.awt.image.RenderedImage

Parameters:

`name` - the name of the property to get, as a `String`.

Returns:

a reference to the property `Object`, or the value `java.awt.Image.UndefinedProperty`.

---

setProperty

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
```

Sets a property on a `PlanarImage`. Some `PlanarImage` subclasses may ignore attempts to set properties.

Parameters:

`name` - a `String` containing the property's name.
`value` - the property, as a general `Object`.

Throws:

java.lang.IllegalArgumentException - if `name` or `value` is `null`.

---

getPropertyNames

`public java.lang.String[] **getPropertyNames**()`

Returns a list of property names that are recognized by this image or `null` if none are recognized.

Specified by:

getPropertyNames in interface java.awt.image.RenderedImage

Returns:

an array of `String`s containing valid property names.

---

getPropertyNames

```
static java.lang.String[] getPropertyNames(java.lang.String[] propertyNames,
                                           java.lang.String prefix)
```

Utility method to search the full list of property names for matches.

---

getPropertyNames

```
public java.lang.String[] getPropertyNames(java.lang.String prefix)
```

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names match, `null` will be returned. The comparison is done in a case-independent manner.

The default implementation calls `getPropertyNames()` and searches the list of names for matches.

Returns:

an array of `String`s giving the valid property names.

Throws:

java.lang.IllegalArgumentException - if `prefix` is `null`.

XToTileX

```
public static int XToTileX(int x,
                           int tileGridXOffset,
                           int tileWidth)
```

Converts a pixel's X coordinate into a horizontal tile index relative to a given tile grid layout specified by its X offset and tile width.

If `tileWidth < 0`, the results of this method are undefined. If `tileWidth == 0`, an `ArithmeticException` will be thrown.

Throws:

ArithmeticException - if `tileWidth == 0`.

YToTileY

```
public static int YToTileY(int y,
                           int tileGridYOffset,
                           int tileHeight)
```

Converts a pixel's Y coordinate into a vertical tile index relative to a given tile grid layout specified by its Y offset and tile height.

If `tileHeight < 0`, the results of this method are undefined. If `tileHeight == 0`, an `ArithmeticException` will be thrown.

Throws:

ArithmeticException - if `tileHeight == 0`.

XToTileX

```
public int XToTileX(int x)
```

Converts a pixel's X coordinate into a horizontal tile index. No attempt is made to detect out-of-range coordinates.

Parameters:

`x` - the X coordinate of a pixel.

Returns:

the X index of the tile containing the pixel.

YToTileY

```
public int YToTileY(int y)
```

Converts a pixel's Y coordinate into a vertical tile index. No attempt is made to detect out-of-range coordinates.

Parameters:

`y` - the Y coordinate of a pixel.

Returns:

the Y index of the tile containing the pixel.

---

tileXToX

```
public static int tileXToX(int tx,
                           int tileGridXOffset,
                           int tileWidth)
```

Converts a horizontal tile index into the X coordinate of its upper left pixel relative to a given tile grid layout specified by its X offset and tile width.

---

tileYToY

```
public static int tileYToY(int ty,
                           int tileGridYOffset,
                           int tileHeight)
```

Converts a vertical tile index into the Y coordinate of its upper left pixel relative to a given tile grid layout specified by its Y offset and tile height.

---

tileXToX

```
public int tileXToX(int tx)
```

Converts a horizontal tile index into the X coordinate of its upper left pixel. No attempt is made to detect out-of-range indices.

Parameters:

`tx` - the horizontal index of a tile.

Returns:

the X coordinate of the tile’s upper left pixel.

---

tileYToY

```
public int tileYToY(int ty)
```

Converts a vertical tile index into the Y coordinate of its upper left pixel. No attempt is made to detect out-of-range indices.

Parameters:

`ty` - the vertical index of a tile.

Returns:

the Y coordinate of the tile’s upper left pixel.

---

getTileRect

```
public java.awt.Rectangle getTileRect(int tileX,
                                      int tileY)
```

Returns a `Rectangle` indicating the active area of a given tile. The `Rectangle` is defined as the intersection of the tile area and the image bounds. No attempt is made to detect out-of-range indices; tile indices lying completely outside of the image will thus return results with a width and height of 0.

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:

a `Rectangle`

---

getSplits

```
public void getSplits(IntegerSequence xSplits,
                      IntegerSequence ySplits,
                      java.awt.Rectangle rect)
```

Within a given rectangle, store the list of tile seams of both X and Y directions into the corresponding split sequence.

Parameters:

`xSplits` - An `IntegerSequence` to which the tile seams in the X direction are to be added.
`ySplits` - An `IntegerSequence` to which the tile seams in the Y direction are to be added.
`rect` - The rectangular region of interest.

Throws:

java.lang.IllegalArgumentException - if `xSplits` is `null`.
java.lang.IllegalArgumentException - if `ySplits` is `null`.
java.lang.IllegalArgumentException - if `rect` is `null`.

---

getData

```
public java.awt.image.Raster getData()
```

Returns the entire image in a single `Raster`. For images with multiple tiles this will require creating a new `Raster` and copying data from multiple tiles into it ("cobbling").

The returned `Raster` is semantically a copy. This means that subsequent updates to this image will not be reflected in the returned `Raster`. For non-writable (immutable) images, the returned value may be a reference to the image’s internal data. The returned `Raster` should be considered non-writable; any attempt to alter its pixel data (such as by casting it to a `WritableRaster` or obtaining and modifying its `DataBuffer`) may result in undefined behavior. The `copyData` method should be used if the returned `Raster` is to be modified.

For a very large image, more than `Integer.MAX_VALUE` entries would be required in the returned `Raster`'s underlying data array. Since the Java language does not permit such an array, an `IllegalArgumentException` will be thrown.

Specified by:
getData in interface java.awt.image.RenderedImage

Returns:
A `Raster` containing the entire image data.

Throws:
java.lang.IllegalArgumentException - if the size of the returned data is too large to be stored in a single `Raster`.

---

getData

`public java.awt.image.Raster` **`getData`**`(java.awt.Rectangle region)`

Returns a specified region of this image in a `Raster`.

The returned `Raster` is semantically a copy. This means that subsequent updates to this image will not be reflected in the returned `Raster`. For non-writable (immutable) images, the returned value may be a reference to the image's internal data. The returned `Raster` should be considered non-writable; any attempt to alter its pixel data (such as by casting it to a `WritableRaster` or obtaining and modifying its `DataBuffer`) may result in undefined behavior. The `copyData` method should be used if the returned `Raster` is to be modified.

The region of the image to be returned is specified by a `Rectangle`. This region may go beyond this image's boundary. If so, the pixels in the areas outside this image's boundary are left unset. Use `getExtendedData` if a specific extension policy is required.

The `region` parameter may also be `null`, in which case the entire image data is returned in the `Raster`.

If `region` is non-`null` but does not intersect the image bounds at all, an `IllegalArgumentException` will be thrown.

It is possible to request a region of an image that would require more than `Integer.MAX_VALUE` entries in the returned `Raster`'s underlying data array. Since the Java language does not permit such an array, an `IllegalArgumentException` will be thrown.

Specified by:
getData in interface java.awt.image.RenderedImage

Parameters:
`region` - The rectangular region of this image to be returned, or `null`.

Returns:
A `Raster` containing the specified image data.

Throws:
java.lang.IllegalArgumentException - if the region does not interset the image bounds.
java.lang.IllegalArgumentException - if the size of the returned data is too large to be stored in a single `Raster`.

---

copyData

`public java.awt.image.WritableRaster` **`copyData`**`()`

Copies the entire image into a single raster.

---

copyData

`public java.awt.image.WritableRaster` **`copyData`**`(java.awt.image.WritableRaster raster)`

Copies an arbitrary rectangular region of this image's pixel data into a caller-supplied `WritableRaster`. The region to be copied is defined as the boundary of the `WritableRaster`, which can be obtained by calling `WritableRaster.getBounds()`.

The supplied `WritableRaster` may have a region that is larger than this image's boundary, in which case only pixels in the part of the region that intersects with this image are copied. The areas outside of this image's boundary are left untouched.

The supplied `WritableRaster` may also be `null`, in which case the entire image is copied into a newly-created `WritableRaster` with a `SampleModel` that is compatible with that of this image.

Specified by:
copyData in interface java.awt.image.RenderedImage

Parameters:
`raster` - A `WritableRaster` to hold the copied pixel data of this image.

Returns:
A reference to the supplied `WritableRaster`, or to a new `WritableRaster` if the supplied one was `null`.

---

copyExtendedData

```
public void copyExtendedData(java.awt.image.WritableRaster dest,
                             BorderExtender extender)
```

Copies an arbitrary rectangular region of the `RenderedImage` into a caller-supplied `WritableRaster`. The portion of the supplied `WritableRaster` that lies outside of the bounds of the image is computed by calling the given `BorderExtender`. The supplied `WritableRaster` must have a `SampleModel` that is compatible with that of the image.

Parameters:

`dest` - a `WritableRaster` to hold the returned portion of the image.
`extender` - an instance of `BorderExtender`.

Throws:

java.lang.IllegalArgumentException - if `dest` or `extender` is `null`.

---

getExtendedData

```
public java.awt.image.Raster getExtendedData(java.awt.Rectangle region,
                                             BorderExtender extender)
```

Returns a copy of an arbitrary rectangular region of this image in a `Raster`. The portion of the rectangle of interest ouside the bounds of the image will be computed by calling the given `BorderExtender`. If the region falls entirely within the image, `extender` will not be used in any way. Thus it is possible to use a `null` value for `extender` when it is known that no actual extension will be required.

The returned `Raster` should be considered non-writable; any attempt to alter its pixel data (such as by casting it to a `WritableRaster` or obtaining and modifying its `DataBuffer`) may result in undefined behavior. The `copyExtendedData` method should be used if the returned `Raster` is to be modified.

Parameters:

`region` - the region of the image to be returned.
`extender` - an instance of `BorderExtender`, used only if the region exceeds the image bounds, or `null`.

Returns:

a `Raster` containing the extended data.

Throws:

NullPointerException - if the region exceeds the image bounds and `extender` is `null`.

---

getAsBufferedImage

```
public java.awt.image.BufferedImage getAsBufferedImage(java.awt.Rectangle rect,
                                                       java.awt.image.ColorModel colorModel)
```

Returns a copy of this image as a `BufferedImage`. A subarea of the image may be copied by supplying a `Rectangle` parameter; if it is set to `null`, the entire image is copied. The supplied Rectangle will be clipped to the image bounds. The image's `ColorModel` may be overridden by supplying a non-`null` second argument. The resulting `ColorModel` must be non-`null` and appropriate for the image's `SampleModel`.

The resulting `BufferedImage` will contain the full requested area, but will always have its upper-left corner translated (0, 0) as required by the `BufferedImage` interface.

Parameters:

`rect` - the `Rectangle` of the image to be copied, or `null` to indicate that the entire image is to be copied.
`colorModel` - a `ColorModel` used to override this image's `ColorModel`, or `null`. The caller is responsible for supplying a `ColorModel` that is compatible with the image's `SampleModel`.

Throws:

java.lang.IllegalArgumentException - if an incompatible `ColorModel` is supplied.

---

getAsBufferedImage

```
public java.awt.image.BufferedImage getAsBufferedImage()
```

Returns a copy of the entire image as a `BufferedImage`. The image's `ColorModel` must be non-`null`, and appropriate for the image's `SampleModel`.

See Also:

`BufferedImage`

---

getGraphics

```
public java.awt.Graphics getGraphics()
```

Returns a `Graphics` object that may be used to draw into this image. By default, an `IllegalAccessError` is thrown. Subclasses that support such drawing, such as `TiledImage`, may override this method to return a suitable `Graphics` object.

getTile

```
public abstract java.awt.image.Raster getTile(int tileX,
                                              int tileY)
```

Returns tile (tileX, tileY). Note that tileX and tileY are indices into the tile array, not pixel locations.

Subclasses must override this method to return a non-`null` value for all tile indices between `getMinTile{X,Y}` and `getMaxTile{X,Y}`, inclusive. Tile indices outside of this region should result in a return value of `null`.

Specified by:
getTile in interface java.awt.image.RenderedImage

Parameters:
`tileX` - the X index of the requested tile in the tile array.
`tileY` - the Y index of the requested tile in the tile array.

getTiles

```
public java.awt.image.Raster[] getTiles(java.awt.Point[] tileIndices)
```

Returns the `Raster`s indicated by the `tileIndices` array. This call allows certain `PlanarImage` subclasses such as `OpImage` to take advantage of the knowledge that multiple tiles are requested at once.

Parameters:
`tileIndices` - An array of Points representing tile indices.

Returns:
An array of `Raster` containing the tiles corresponding to the given tile indices.

prefetchTiles

```
public void prefetchTiles(java.awt.Point[] tileIndices)
```

Hints that the given tiles might be needed in the near future. Some implementations may spawn a thread or threads to compute the tiles while others may ignore the hint.

Parameters:
`tileIndices` - A list of tile indices indicating which tiles to prefetch.

dispose

```
public void dispose()
```

Provides a hint that an image will no longer be accessed from a reference in user space. The results are equivalent to those that occur when the program loses its last reference to this image, the garbage collector discovers this, and finalize is called. This can be used as a hint in situations where waiting for garbage collection would be overly conservative.

`PlanarImage` defines this method to remove the image being disposed from the list of sinks in all of its source images. Subclasses should call `super.dispose()` in their `dispose` methods, if any.

The results of referencing an image after a call to `dispose()` are undefined.

finalize

```
protected void finalize()
                 throws java.lang.Throwable
```

Performs cleanup prior to garbage collection.

Throws:
`Throwable` - if an error occurs in the garbage collector.

Overrides:
finalize in class java.lang.Object

printBounds

```
private void printBounds()
```

For debugging.

print_tile

```
private void print_tile(int i,
                        int j)
```

For debugging.

---

print

```
private void print()
```

For debugging.

---

cobbleByte

```
private void cobbleByte(java.awt.Rectangle bounds,
                        java.awt.image.Raster dstRaster)
```

---

cobbleShort

```
private void cobbleShort(java.awt.Rectangle bounds,
                         java.awt.image.Raster dstRaster)
```

---

cobbleUShort

```
private void cobbleUShort(java.awt.Rectangle bounds,
                          java.awt.image.Raster dstRaster)
```

---

cobbleInt

```
private void cobbleInt(java.awt.Rectangle bounds,
                       java.awt.image.Raster dstRaster)
```

---

cobbleFloat

```
private void cobbleFloat(java.awt.Rectangle bounds,
                         java.awt.image.Raster dstRaster)
```

---

cobbleDouble

```
private void cobbleDouble(java.awt.Rectangle bounds,
                          java.awt.image.Raster dstRaster)
```

javax.media.jai
Class PointOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.PointOpImage
```

Direct Known Subclasses:
NullOpImage

---

public abstract class **PointOpImage**
extends OpImage

An abstract base class for image operators that require only the (x, y) pixel from each source image in order to compute the destination pixel (x, y).

`PointOpImage` is intended as a convenient superclass for `OpImage>s that only need to look at each destination pixel’s corresponding source pixels. Some examples are lookup, contrast adjustment, pixel arithmetic, and color space conversion.`

See Also: <B>
OpImage

Field Detail

areFieldsInitialized
`private boolean areFieldsInitialized`

---

checkInPlaceOperation
`private boolean checkInPlaceOperation`

---

isInPlaceEnabled
`private boolean isInPlaceEnabled`

---

source0AsWritableRenderedImage
`private java.awt.image.WritableRenderedImage source0AsWritableRenderedImage`

---

source0AsOpImage
`private OpImage source0AsOpImage`

---

source0IsWritableRenderedImage
`private boolean source0IsWritableRenderedImage`

---

sameBounds
`private boolean sameBounds`

---

sameTileGrid
`private boolean sameTileGrid`

Constructor Detail

PointOpImage

```
public PointOpImage(java.util.Vector sources,
                    TileCache cache,
                    ImageLayout layout,
                    boolean cobbleSources)
```

Constructs a `PointOpImage` with a `Vector` of `RenderedImage`s as its sources.

The `layout` parameter is passed to the superclass constructor unchanged, where it is used along with the source image layouts to determine the output image layout in the standard way.

Parameters:

`sources` - The source images.
`cache` - A TileCache object to store tiles from this OpImage, or null. If `null`, a default cache will be used.
`layout` - The layout parameters of the destination image.
`cobbleSources` - `true` if computeRect() expects contiguous sources.

Throws:

java.lang.IllegalArgumentException - if combining the intersected source bounds with the layout parameter results in negative output width or height.

PointOpImage

```
public PointOpImage(java.awt.image.RenderedImage source,
                    TileCache cache,
                    ImageLayout layout,
                    boolean cobbleSources)
```

Constructs a `PointOpImage` with one source image. The image layout is computed as described in the constructor taking a `Vector` of sources.

Parameters:

`source` - The source image.
`cache` - A TileCache object to store tiles from this OpImage, or `null`. If `null`, a default cache will be used.
`layout` - The layout parameters of the destination image.
`cobbleSources` - Indicates whether `computeRect()` expects contiguous sources.

PointOpImage

```
public PointOpImage(java.awt.image.RenderedImage source0,
                    java.awt.image.RenderedImage source1,
                    TileCache cache,
                    ImageLayout layout,
                    boolean cobbleSources)
```

Constructs a `PointOpImage` with two source images. The image layout is computed as described in the constructor taking a `Vector` of sources.

Parameters:

`source0` - The first source image.
`source1` - The second source image.
`cache` - A TileCache object to store tiles from this OpImage, or `null`. If `null`, a default cache will be used.
`layout` - The layout parameters of the destination image.
`cobbleSources` - Indicates whether `computeRect()` expects contiguous sources.

PointOpImage

```
public PointOpImage(java.awt.image.RenderedImage source0,
                    java.awt.image.RenderedImage source1,
                    java.awt.image.RenderedImage source2,
                    TileCache cache,
                    ImageLayout layout,
                    boolean cobbleSources)
```

Constructs a `PointOpImage` with three source images. The image layout is computed as described in the constructor taking a `Vector` of sources.

Parameters:

`source0` - The first source image.
`source1` - The second source image.
`source2` - The third source image.
`cache` - A TileCache object to store tiles from this OpImage, or `null`. If `null`, a default cache will be used.
`layout` - The layout parameters of the destination image.
`cobbleSources` - Indicates whether `computeRect()` expects contiguous sources.

Method Detail

initializeFields

```
private void initializeFields()
```

---

hasCompatibleSampleModel

```
private boolean hasCompatibleSampleModel(PlanarImage src)
```

---

permitInPlaceOperation

```
protected void permitInPlaceOperation()
```

Causes a flag to be set to indicate that in-place operation should be permitted if the image bounds, tile grid offset, tile dimensions, and SampleModels of the source and destination images are compatible. This method should be invoked in the constructor of the implementation of a given operation only if that implementation is amenable to in-place computation. Invocation of this method is a necessary but not a sufficient condition for in-place computation actually to occur. If the system property "javax.media.jai.PointOpImage.InPlace" is equal to the string "false" in a case-insensitive fashion then in-place operation will not be permitted.

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. If source cobbling was requested at construction time, the source tile boundaries are overlayed onto the destination and `computeRect(Raster[], WritableRaster, Rectangle)` is called for each of the resulting regions. Otherwise, `computeRect(PlanarImage[], WritableRaster, Rectangle)` is called once to compute the entire active area of the tile.

The image bounds may be larger than the bounds of the source image. In this case, samples for which there are no no corresponding sources are set to zero.

Parameters:
`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Overrides:
computeTile in class OpImage

---

mapSourceRect

```
public final java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                              int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:
`sourceRect` - the `Rectangle` in source coordinates.
`sourceIndex` - the index of the source image.

Returns:
a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:
java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `sourceRect` is `null`.

Overrides:
mapSourceRect in class OpImage

---

mapDestRect

```
public final java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                            int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:
`destRect` - the `Rectangle` in source coordinates.
`sourceIndex` - the index of the source image.

Returns:
a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:
java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `destRect` is `null`.

Overrides:
mapDestRect in class OpImage

javax.media.jai
Class ProductOperationGraph

```
java.lang.Object
  |
  +--javax.media.jai.ProductOperationGraph
```

class **ProductOperationGraph**
extends java.lang.Object
implements java.io.Serializable

ProductOperationGraph manages a list of operations (image factories) belonging to a particular product. The operations have pairwise preferences between them. The getOrderedOperationList method performs a topological sort. The topological sort follows the algorithm described in Horowitz and Sahni, *Fundamentals of Data Structures* (1976), p. 315.

Several minor changes are made to their implementation. First, nodes are represented as objects, not as integers. The count (in-degree) field is not used to link zero in-degree objects, but instead a separate zeroLink field is used. The neighbor lists are stored as Vectors, not linked lists, and enumerations are used to iterate over them.

This class is used by the implementation of the OperationRegistry class and is not intended to be part of the API.

Field Detail

operations

`protected java.util.Vector **operations**`

A Vector of RIF implementations.

orderedProducts

`protected java.util.Vector **orderedProducts**`

A cached version of the ordered product list

isChanged

`protected boolean **isChanged**`

Signifies whether the cached copy is out of date.

lock

`com.sun.media.jai.util.ReaderWriterLock **lock**`

Constructor Detail

ProductOperationGraph

`public **ProductOperationGraph**()`

Constructs an ProductOperationGraph.

Method Detail

addProduct

`public void **addProduct**(java.lang.String productName)`

Adds a product to an ProductOperationGraph. A new PartialOrderNode is constructed to hold the product and its graph adjacency information.

lookupOp

```
public PartialOrderNode lookupOp(java.lang.String productName)
```

Locates a product from within the vector of PartialOrderNodes using the productName provided. NOTE: CHANGING access from private to public

---

setPreference

```
public void setPreference(java.lang.String preferredOp,
                          java.lang.String otherOp)
```

Sets a preference between two products.

---

unsetPreference

```
public void unsetPreference(java.lang.String preferredOp,
                            java.lang.String otherOp)
```

Removes a preference between two products.

---

getOrderedOperationList

```
public java.util.Vector getOrderedOperationList()
```

Performs a topological sort on the set of RIFs.

javax.media.jai

Interface PropertyGenerator

All Known Implementing Classes:

PropertyGeneratorFromSource, CopyPropertyGenerator, PolarToComplexPropertyGenerator, ImageFunctionPropertyGenerator, WarpPropertyGenerator, MagnitudePropertyGenerator, PhasePropertyGenerator, MultiplyComplexPropertyGenerator, TransposePropertyGenerator, TranslatePropertyGenerator, DFTPropertyGenerator, ShearPropertyGenerator, AffinePropertyGenerator, RotatePropertyGenerator, IDFTPropertyGenerator, MagnitudeSquaredPropertyGenerator, ConjugatePropertyGenerator, DivideComplexPropertyGenerator, ScalePropertyGenerator

public abstract interface **PropertyGenerator**
extends java.io.Serializable

An interface through which properties may be computed dynamically with respect to an environment of pre-existing properties. In the interest of simplicity and consistency, a PropertyGenerator is required to be a pure function; that is, if called multiple times with the same environment it must produce identical results.

The OperationRegistry class allows PropertyGenerators to be associated with a particular operation type, and will automatically insert them into imaging chains as needed.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns an array of Strings naming properties emitted by this property generator.

Returns:

an array of Strings that may be passed as parameter names to the getProperty() method.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Computes the value of a property relative to an environment of pre-existing properties emitted by the sources of a RenderedOp, and the parameters of that operation.

The operation name, sources, and ParameterBlock of the RenderedOp being processed may be obtained by means of the op.getOperationName, op.getSources(), and op.getParameterBlock() methods. It is legal to call getProperty() on the operation’s sources.

Parameters:

`name` - the name of the property, as a String.
`op` - the RenderedOp representing the operation.

Returns:

the value of the property, as an Object.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Computes the value of a property relative to an environment of pre-existing properties emitted by the sources of a RenderableOp, and the parameters of that operation.

The operation sources and ParameterBlock of the RenderableOp being processed may be obtained by means of the op.getSources() and op.getParameterBlock() methods. It is legal to call getProperty() on the operation’s sources.

Parameters:

`name` - the name of the property, as a String.
`op` - the RenderableOp representing the operation.

Returns:

the value of the property, as an Object.

javax.media.jai

Class PropertyGeneratorFromSource

```
java.lang.Object
  |
  +--javax.media.jai.PropertyGeneratorFromSource
```

class **PropertyGeneratorFromSource**
extends java.lang.Object
implements PropertyGenerator

A class that implements the PropertyGenerator interface. This class is used when a property is to be calculated from a particular source.

Field Detail

sourceIndex

```
int sourceIndex
```

propertyName

```
java.lang.String propertyName
```

Constructor Detail

PropertyGeneratorFromSource

```
PropertyGeneratorFromSource(int sourceIndex,
                            java.lang.String propertyName)
```

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Specified by:
getProperty in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Specified by:
getProperty in interface PropertyGenerator

javax.media.jai

Interface PropertySource

All Known Subinterfaces:
ImageJAI

All Known Implementing Classes:
PropertySourceImpl, RenderableImageAdapter, RenderableOp

---

public abstract interface **PropertySource**

An interface encapsulating the set of operations involved in identifying and reading properties.

The interface consists of the getProperty() and getPropertyNames() methods familiar from the RenderedImage and RenderableImage interfaces.

PropertySource is implemented by PlanarImage. Since all RenderedImages used with JAI are "wrapped" by a RenderedImageAdapter, all JAI images may be assumed to implement PropertySource.

See Also:
`RenderedImage`, `RenderableImage`

Method Detail

getPropertyNames

`public java.lang.String[] getPropertyNames()`

Returns an array of `String`s recognized as names by this property source. If no properties are available, `null` will be returned.

Returns:
an array of `String`s giving the valid property names.

---

getPropertyNames

`public java.lang.String[] getPropertyNames(java.lang.String prefix)`

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names match, `null` will be returned. The comparison is done in a case-independent manner.

Returns:
an array of `String`s giving the valid property names.

---

getProperty

`public java.lang.Object getProperty(java.lang.String name)`

Returns the value of a property. If the property name is not recognized, `java.awt.Image.UndefinedProperty` will be returned.

Parameters:
`name` - the name of the property, as a `String`.

Returns:
the value of the property, as an `Object`, or the value `java.awt.Image.UndefinedProperty`.

javax.media.jai

Class PropertySourceImpl

```
java.lang.Object
  |
  +--javax.media.jai.PropertySourceImpl
```

---

class **PropertySourceImpl**
extends java.lang.Object
implements PropertySource

A class that implements the PropertySource interface.

Field Detail

pg

`java.util.Vector pg`

---

sources

`java.util.Vector sources`

---

suppNames

`java.util.Vector suppNames`

---

sourceForProp

`java.util.Hashtable sourceForProp`

---

isRendered

`boolean isRendered`

---

op

`java.lang.Object op`

---

propNames

`private java.util.Hashtable propNames`

Constructor Detail

PropertySourceImpl

```
public PropertySourceImpl(java.util.Vector sources,
                          java.util.Vector generators,
                          java.util.Vector suppressed,
                          java.util.Hashtable sourceForProp,
                          RenderedOp op)
```

PropertySourceImpl

```
public PropertySourceImpl(java.util.Vector sources,
                          java.util.Vector generators,
                          java.util.Vector suppressed,
                          java.util.Hashtable sourceForProp,
                          RenderableOp op)
```

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns an array of Strings recognized as names by this property source.

Specified by:
getPropertyNames in interface PropertySource

Returns:
an array of Strings giving the valid property names.

---

getPropertyNames

```
public java.lang.String[] getPropertyNames(java.lang.String prefix)
```

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names match, `null` will be returned. The comparison is done in a case-independent manner.

The default implementation calls `getPropertyNames()` and searches the list of names for matches.

Specified by:
getPropertyNames in interface PropertySource

Returns:
an array of `String`s giving the valid property names.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Returns the value of a property.

Specified by:
getProperty in interface PropertySource

Parameters:
`name` - the name of the property, as a String.

Returns:
the value of the property, as an Object.

---

copyPropertyFromSource

```
public void copyPropertyFromSource(java.lang.String propertyName,
                                   int sourceIndex)
```

---

suppressProperty

```
public void suppressProperty(java.lang.String propertyName)
```

---

addPropertyGenerator

```
public void addPropertyGenerator(PropertyGenerator generator)
```

---

removePropertyGenerator

```
public void removePropertyGenerator(PropertyGenerator generator)
```

---

removeSuppressedProps

```
private void removeSuppressedProps(PropertyGenerator generator)
```

---

hashNames

```
private void hashNames()
```

javax.media.jai
Class ROI

```
java.lang.Object
  |
  +--javax.media.jai.ROI
```

Direct Known Subclasses:
ROIShape

public class **ROI**
extends java.lang.Object
implements java.io.Serializable

The parent class for representations of a region of interest of an image. This class represents region information in image form, and can thus be used as a fallback where a `Shape` representation is unavailable. Where possible, subclasses such as ROIShape are used since they provide a more compact means of storage for large regions.

The getAsShape() method may be called optimistically on any instance of ROI; however, it may return null to indicate that a `Shape` representation of the ROI is not available. In this case, getAsImage() should be called as a fallback.

Inclusion and exclusion of pixels is defined by a threshold value. Pixel values greater than or equal to the threshold indicate inclusion.

Field Detail

iter

`private transient RandomIter iter`

A RandomIter used to grab pixels from the ROI.

theImage

`transient PlanarImage theImage`

The `PlanarImage` representation of the ROI.

threshold

`int threshold`

The inclusion/exclusion threshold of the ROI.

Constructor Detail

ROI

`protected ROI()`

The default constructor.

ROI

`public ROI(java.awt.image.RenderedImage im)`

Constructs an ROI from a RenderedImage. The inclusion threshold is taken to be halfway between the minimum and maximum sample values specified by the image’s SampleModel.

Parameters:
`im` - A single-banded RenderedImage.

Throws:
java.lang.IllegalArgumentException - if im is null.

ROI

```
public ROI(java.awt.image.RenderedImage im,
           int threshold)
```

Constructs an ROI from a RenderedImage. The inclusion threshold is specified explicitly.

Parameters:

`im` - A single-banded RenderedImage.

`threshold` - The desired inclusion threshold.

Throws:

java.lang.IllegalArgumentException - if im is null.

Method Detail

mergeRunLengthList

```
protected static java.util.LinkedList mergeRunLengthList(java.util.LinkedList rectList)
```

Merge a `LinkedList` of `Rectangle`s representing run lengths of pixels in the ROI into a minimal list wherein vertically abutting `Rectangle`s are merged. The operation is effected in place.

Parameters:

`rectList` - The list of run length `Rectangle`s.

Returns:

The merged list.

Throws:

NullPointerException - if rectList is null.

---

mergeImages

```
private static PlanarImage mergeImages(PlanarImage im1,
                                       PlanarImage im2,
                                       PlanarImage overlap)
```

---

getIter

```
private RandomIter getIter()
```

Get the iterator, construct it if need be.

---

getThreshold

```
public int getThreshold()
```

Returns the inclusion/exclusion threshold value.

---

setThreshold

```
public void setThreshold(int threshold)
```

Sets the inclusion/exclusion threshold value.

---

getBounds

```
public java.awt.Rectangle getBounds()
```

Returns the bounds of the ROI as a `Rectangle`.

---

getBounds2D

```
public java.awt.geom.Rectangle2D getBounds2D()
```

Returns the bounds of the ROI as a `Rectangle2D`.

---

contains

```
public boolean contains(java.awt.Point p)
```

Returns `true` if the ROI contains a given Point.

Parameters:

`p` - A Point identifying the pixel to be queried.

Returns:
`true` if the pixel lies within the ROI.
Throws:
NullPointerException - if p is null.

---

contains

```
public boolean contains(java.awt.geom.Point2D p)
```

Returns `true` if the ROI contains a given Point2D.
Parameters:
`p` - A Point2D identifying the pixel to be queried.
Returns:
`true` if the pixel lies within the ROI.
Throws:
NullPointerException - if p is null.

---

contains

```
public boolean contains(int x,
                        int y)
```

Returns `true` if the ROI contains the point (x, y).
Parameters:
`x` - An int specifying the X coordinate of the pixel to be queried.
`y` - An int specifying the Y coordinate of the pixel to be queried.
Returns:
`true` if the pixel lies within the ROI.

---

contains

```
public boolean contains(double x,
                        double y)
```

Returns `true` if the ROI contain the point (x, y).
Parameters:
`x` - A double specifying the X coordinate of the pixel to be queried.
`y` - A double specifying the Y coordinate of the pixel to be queried.
Returns:
`true` if the pixel lies within the ROI.

---

contains

```
public boolean contains(java.awt.Rectangle rect)
```

Returns `true` if a given `Rectangle` is entirely included within the ROI.
Parameters:
`rect` - A `Rectangle` specifying the region to be tested for inclusion.
Returns:
`true` if the rectangle is entirely contained within the ROI.
Throws:
NullPointerException - if rect is null.

---

contains

```
public boolean contains(java.awt.geom.Rectangle2D rect)
```

Returns `true` if a given `Rectangle2D` is entirely included within the ROI.
Parameters:
`rect` - A `Rectangle2D` specifying the region to be tested for inclusion.
Returns:
`true` if the rectangle is entirely contained within the ROI.
Throws:
NullPointerException - if rect is null.

---

contains

```
public boolean contains(int x,
                        int y,
                        int w,
                        int h)
```

Returns `true` if a given rectangle (x, y, w, h) is entirely included within the ROI.

Parameters:

`x` - The int X coordinate of the upper left corner of the region.
`y` - The int Y coordinate of the upper left corner of the region.
`w` - The int width of the region.
`h` - The int height of the region.

Returns:

`true` if the rectangle is entirely contained within the ROI.

---

contains

```
public boolean contains(double x,
                        double y,
                        double w,
                        double h)
```

Returns `true` if a given rectangle (x, y, w, h) is entirely included within the ROI.

Parameters:

`x` - The double X coordinate of the upper left corner of the region.
`y` - The double Y coordinate of the upper left corner of the region.
`w` - The double width of the region.
`h` - The double height of the region.

Returns:

`true` if the rectangle is entirely contained within the ROI.

---

intersects

```
public boolean intersects(java.awt.Rectangle rect)
```

Returns `true` if a given `Rectangle` intersects the ROI.

Parameters:

`rect` - A `Rectangle` specifying the region to be tested for inclusion.

Returns:

`true` if the rectangle intersects the ROI.

Throws:

NullPointerException - if rect is null.

---

intersects

```
public boolean intersects(java.awt.geom.Rectangle2D r)
```

Returns `true` if a given `Rectangle2D` intersects the ROI.

Parameters:

`r` - A `Rectangle2D` specifying the region to be tested for inclusion.

Returns:

`true` if the rectangle intersects the ROI.

Throws:

NullPointerException - if r is null.

---

intersects

```
public boolean intersects(int x,
                          int y,
                          int w,
                          int h)
```

Returns `true` if a given rectangular region intersects the ROI.

Parameters:

`x` - The int X coordinate of the upper left corner of the region.
`y` - The int Y coordinate of the upper left corner of the region.
`w` - The int width of the region.
`h` - The int height of the region.

Returns:
`true` if the rectangle intersects the ROI.

---

intersects

```
public boolean intersects(double x,
                          double y,
                          double w,
                          double h)
```

Returns `true` if a given rectangular region intersects the ROI.

Parameters:
`x` - The double X coordinate of the upper left corner of the region.
`y` - The double Y coordinate of the upper left corner of the region.
`w` - The double width of the region.
`h` - The double height of the region.

Returns:
`true` if the rectangle intersects the ROI.

---

add

```
public ROI add(ROI roi)
```

Adds another ROI to this one and returns the result as a new ROI. The supplied ROI will be converted to a rendered form if necessary.

Parameters:
`roi` - An ROI.

Returns:
A new ROI containing the new ROI data.

Throws:
NullPointerException - if roi is null.

---

subtract

```
public ROI subtract(ROI roi)
```

Subtracts another ROI from this one and returns the result as a new ROI. The supplied ROI will be converted to a rendered form if necessary.

Parameters:
`roi` - An ROI.

Returns:
A new ROI containing the new ROI data.

Throws:
NullPointerException - if roi is null.

---

intersect

```
public ROI intersect(ROI roi)
```

Intersects the ROI with another ROI and returns the result as a new ROI. The supplied ROI will be converted to a rendered form if necessary.

Parameters:
`roi` - An ROI.

Returns:
A new ROI containing the new ROI data.

Throws:
NullPointerException - if roi is null.

---

exclusiveOr

```
public ROI exclusiveOr(ROI roi)
```

Exclusive-ors the ROI with another ROI and returns the result as a new ROI. The supplied ROI will be converted to a rendered form if necessary.

Parameters:
`roi` - An ROI.

Returns:
A new ROI containing the new ROI data.

Throws:

NullPointerException - if roi is null.

---

transform

```
public ROI transform(java.awt.geom.AffineTransform at,
                     Interpolation interp)
```

Performs an affine transformation and returns the result as a new ROI. The transformation is performed by an "Affine" RIF using the indicated interpolation method.

Parameters:

`at` - an AffineTransform specifying the transformation.
`interp` - the Interpolation to be used.

Returns:

a new ROI containing the transformed ROI data.

Throws:

NullPointerException - if at is null.
NullPointerException - if interp is null.

---

transform

```
public ROI transform(java.awt.geom.AffineTransform at)
```

Performs an affine transformation and returns the result as a new ROI. The transformation is performed by an "Affine" RIF using nearest neighbor interpolation.

Parameters:

`at` - an AffineTransform specifying the transformation.

Returns:

a new ROI containing the transformed ROI data.

Throws:

NullPointerException - if at is null.

---

performImageOp

```
public ROI performImageOp(java.awt.image.renderable.RenderedImageFactory RIF,
                          java.awt.image.renderable.ParameterBlock paramBlock,
                          int sourceIndex,
                          java.awt.RenderingHints renderHints)
```

Transforms an ROI using an imaging operation. The operation is specified by a `RenderedImageFactory`. The operation's `ParameterBlock`, minus the image source itself is supplied, along with an index indicating where to insert the ROI image. The `renderHints` argument allows rendering hints to be passed in.

Parameters:

`RIF` - A `RenderedImageFactory` that will be used to create the op.
`paramBlock` - A `ParameterBlock` containing all sources and parameters for the op except for the ROI itself.
`sourceIndex` - The index of the `ParameterBlock`'s sources where the ROI is to be inserted.
`renderHints` - A `RenderingHints` object containing rendering hints, or null.

Throws:

NullPointerException - if RIF is null.
NullPointerException - if paramBlock is null.

---

performImageOp

```
public ROI performImageOp(java.lang.String name,
                          java.awt.image.renderable.ParameterBlock paramBlock,
                          int sourceIndex,
                          java.awt.RenderingHints renderHints)
```

Transforms an ROI using an imaging operation. The operation is specified by name; the default JAI registry is used to resolve this into a RIF. The operation's `ParameterBlock`, minus the image source itself is supplied, along with an index indicating where to insert the ROI image. The `renderHints` argument allows rendering hints to be passed in.

Parameters:

`name` - The name of the operation to perform.
`paramBlock` - A `ParameterBlock` containing all sources and parameters for the op except for the ROI itself.
`sourceIndex` - The index of the `ParameterBlock`'s sources where the ROI is to be inserted.
`renderHints` - A `RenderingHints` object containing rendering hints, or null.

Throws:

NullPointerException - if name is null.
NullPointerException - if paramBlock is null.

getAsShape

```
public java.awt.Shape getAsShape()
```

Returns a `Shape` representation of the ROI, if possible. If none is available, null is returned. A proper instance of ROI (one that is not an instance of any subclass of ROI) will always return null.

Returns:
The ROI as a `Shape`.

getAsImage

```
public PlanarImage getAsImage()
```

Returns a `PlanarImage` representation of the ROI. This method will always succeed.

Returns:
The ROI as a `PlanarImage`.

getAsBitmask

```
public int[][] getAsBitmask(int x,
                            int y,
                            int width,
                            int height,
                            int[][] mask)
```

Returns a bitmask for a given rectangular region of the ROI indicating whether the pixel is included in the region of interest. The results are packed into 32-bit integers, with the MSB considered to lie on the left. The last entry in each row of the result may have bits that lie outside of the requested rectangle. These bits are guaranteed to be zeroed.

The `mask` array, if supplied, must be of length equal to or greater than `height` and each of its subarrays must have length equal to or greater than (width + 31)/32. If `null` is passed in, a suitable array will be constructed. If the mask is non-null but has insufficient size, an exception will be thrown.

Parameters:
`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.
`height` - The height of the rectangle.
`mask` - A two-dimensional array of ints at least (width + 31)/32 entries wide and (height) entries tall, or null.

Returns:
A reference to the `mask` parameter, or to a newly constructed array if `mask` is `null`.

getAsRectangleList

```
public java.util.LinkedList getAsRectangleList(int x,
                                               int y,
                                               int width,
                                               int height)
```

Returns a `LinkedList` of `Rectangle`s for a given rectangular region of the ROI. The `Rectangle`s in the list are merged into a minimal set.

Parameters:
`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.
`height` - The height of the rectangle.

Returns:
A `LinkedList` of `Rectangle`s.

getAsRectangleList

```
protected java.util.LinkedList getAsRectangleList(int x,
                                                  int y,
                                                  int width,
                                                  int height,
                                                  boolean mergeRectangles)
```

Returns a `LinkedList` of `Rectangle`s for a given rectangular region of the ROI.

Parameters:
`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.

`height` - The height of the rectangle.
`mergeRectangles` - `true` if the `Rectangle`s are to be merged into a minimal set.

Returns:
A `LinkedList` of `Rectangle`s.

---

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the `ROI`.

Parameters:
`out` - The `ObjectOutputStream`.

---

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the `ROI`.

Parameters:
`in` - The `ObjectInputStream`.

javax.media.jai

Class ROIShape.PolyShape.PolyEdge

```
java.lang.Object
  |
  +--javax.media.jai.ROIShape.PolyShape.PolyEdge
```

private class **ROIShape.PolyShape.PolyEdge**
extends java.lang.Object
implements java.util.Comparator

Inner class representing a polygon edge.

Field Detail

x

`public double x`

X cooridnate of intersection of edge with current scanline.

dx

`public double dx`

Change in X with respect to Y.

i

`public int i`

The edge number: edge i goes from vertex i to vertex i+1.

Constructor Detail

ROIShape.PolyShape.PolyEdge

```
ROIShape.PolyShape.PolyEdge(double x,
                            double dx,
                            int i)
```

Construct a `PolyEdge` object.

Parameters:

`x` - X cooridnate of edge intersection with scanline.
`dx` - The change in X with respect to Y.
`i` - The edge number.

Method Detail

compare

```
public int compare(java.lang.Object o1,
                   java.lang.Object o2)
```

Implementation of java.util.Comparator.compare. The argument `Object`s are assumed to be `PolyEdge`s and are sorted on the basis of their respective x components.

Specified by:

compare in interface java.util.Comparator

Parameters:

`o1` - The first `PolyEdge` object.
`o2` - The second `PolyEdge` object.

Returns:

-1 if o1 < o2, 1 if o1 > o2, 0 if o1 == o2.

javax.media.jai
Class ROIShape.PolyShape

```
java.lang.Object
  |
  +--javax.media.jai.ROIShape.PolyShape
```

private class **ROIShape.PolyShape**
extends java.lang.Object

Instance inner class used for scan conversion of a polygonal `Shape`.

Field Detail

POLYGON_UNCLASSIFIED

`private static final int` **`POLYGON_UNCLASSIFIED`**

A polygon which has yet to be classified as one of the following types.

POLYGON_DEGENERATE

`private static final int` **`POLYGON_DEGENERATE`**

A degenerate polygon, i.e., all vertices equal or on the same line.

POLYGON_CONVEX

`private static final int` **`POLYGON_CONVEX`**

A convex polygon.

POLYGON_CONCAVE

`private static final int` **`POLYGON_CONCAVE`**

A concave polygon (simple or non-simple).

poly

`private java.awt.Polygon` **`poly`**

The internal polygon.

clip

`private java.awt.Rectangle` **`clip`**

The clipping `Rectangle`.

type

`private int` **`type`**

The type of polygon.

insideRect

`private boolean` **`insideRect`**

Flag indicating whether the `Polygon` is inside the supplied clipping `Rectangle`.

Constructor Detail

ROIShape.PolyShape

```
ROIShape.PolyShape(java.awt.Polygon polygon,
                   java.awt.Rectangle clipRect)
```

Constructs a new PolyShape. The `Polygon` argument is clipped against the supplied `Rectangle`.

Parameters:

`polygon` - The `Polygon`.
`clipRect` - The clipping `Rectangle`.

Method Detail

getAsRectList

```
public java.util.LinkedList getAsRectList()
```

Perform scan conversion of the `PolyShape` to generate a `LinkedList` of `Rectangle`s.

Returns:

A `LinkedList` of `Rectangle`s representing the scan conversion of the `PolyShape`.

---

classifyPolygon

```
private int classifyPolygon()
```

Classify a `Polygon` as one of the pre-defined types for this class.

---

sgn

```
private int sgn(int i)
```

Calculate the sign of the argument.

Parameters:

`i` - The integer the sign of which is to be determined.

Returns:

1 for positive, -1 for negative, and 0 for zero arguments.

---

scanConvex

```
private java.util.LinkedList scanConvex(java.util.LinkedList rectList)
```

Perform scan conversion of a convex polygon.

Parameters:

`rectList` - A `LinkedList`; may be null.

Returns:

A `LinkedList` of `Rectangle`s representing the scan conversion of the convex polygon.

---

scanSegment

```
private java.awt.Rectangle scanSegment(int y,
                                       double leftX,
                                       double rightX)
```

Return a `Rectangle` for the supplied line and abscissa end points.

Parameters:

`y` - The line number.
`leftX` - The left end point of the segment.
`rightX` - The right end point of the segment.

Returns:

The run length `Rectangle` for the segment.

---

intersectX

```
private void intersectX(double x1,
                        int y1,
                        double x2,
                        int y2,
                        int y,
                        double[] x,
                        double[] dx)
```

For the line y + 0.5 calculate the intersection with the segment (x1, y1) to (x2, y2) as well as the slope dx/dy at the point of intersection.

Parameters:

`x1` - Abscissa of first segment end point.
`y1` - Ordinate of first segment end point.
`x2` - Abscissa of second segment end point.
`y2` - Ordinate of second segment end point.
`y` - The image line to intersect.
`x` - The abscissa of the point of intersection.
`dx` - The slope dx/dy of the point of intersection.

---

scanConcave

```
private java.util.LinkedList scanConcave(java.util.LinkedList rectList)
```

Perform scan conversion of a concave polygon.

Parameters:

`rectList` - A `LinkedList`; may be null.

Returns:

A `LinkedList` of `Rectangle`s representing the scan conversion of the concave polygon.

---

deleteEdge

```
private void deleteEdge(java.util.Vector edges,
                        int i)
```

Delete a `PolyEdge` from the Vector of active edges.

Parameters:

`edges` - The `Vector` of `PolyEdge`s.
`i` - The number of the edge to be deleted.

---

appendEdge

```
private void appendEdge(java.util.Vector edges,
                        int i,
                        int y)
```

Append a `PolyEdge` to the Vector of active edges.

Parameters:

`edges` - The `Vector` of `PolyEdge`s.
`i` - The number of the edge to be appended.
`y` - The y coordinate of the current scanline.

---

intArrayToDoubleArray

```
private double[] intArrayToDoubleArray(int[] intArray)
```

Convert an array of `int`s to an array of `double`s.

---

vectorToIntArray

```
private int[] vectorToIntArray(java.util.Vector vector)
```

Convert a `Vector` of `Integer`s to an array of `int`s.

Parameters:

`vector` - A `Vector` of `Integer`s.

Returns:

The array of `int`s.

javax.media.jai
Class ROIShape

```
java.lang.Object
  |
  +--javax.media.jai.ROI
        |
        +--javax.media.jai.ROIShape
```

public class **ROIShape**
extends ROI

A class representing a region of interest within an image as a `Shape`. Such regions are binary by definition. Using a `Shape` representation allows boolean operations to be performed quickly and with compact storage. If a `PropertyGenerator` responsible for generating the `ROI` property of a particular `OperationDescriptor` (e.g., a warp) cannot reasonably produce an `ROIShape` representing the region, it should call `getAsImage()` on its sources and produce its output `ROI` in image form.

Field Detail

theShape

`transient java.awt.Shape` **`theShape`**

The internal Shape that defines this mask.

Constructor Detail

ROIShape

`public` **`ROIShape`**`(java.awt.Shape s)`

Constructs an ROIShape from a Shape.
Parameters:
`s` - A Shape.
Throws:
java.lang.IllegalArgumentException - if s is null.

ROIShape

`public` **`ROIShape`**`(java.awt.geom.Area a)`

Constructs an ROIShape from an Area.
Parameters:
`a` - An Area.

Method Detail

getIntersection

```
private static java.awt.geom.Point2D.Double getIntersection(double x1,
                                                            double y1,
                                                            double x2,
                                                            double y2,
                                                            double u1,
                                                            double v1,
                                                            double u2,
                                                            double v2)
```

Calculate the point of intersection of two line segments. This method assumes that the line segments do in fact intersect.
Parameters:
`x1` - The abscissa of the first end point of the first segment.
`y1` - The ordinate of the first end point of the first segment.
`x2` - The abscissa of the second end point of the first segment.
`y2` - The ordinate of the second end point of the first segment.
`u1` - The abscissa of the first end point of the second segment.
`v1` - The ordinate of the first end point of the second segment.
`u2` - The abscissa of the second end point of the second segment.
`v2` - The ordinate of the second end point of the second segment.

Returns:
The point of intersection.

---

polygonToRunLengthList

```
private java.util.LinkedList polygonToRunLengthList(java.awt.Rectangle clip,
                                                    java.awt.Polygon poly)
```

Convert a `Polygon` into a `LinkedList` of `Rectangle`s representing run lengths of pixels contained within the `Polygon`.

Parameters:
`clip` - The clipping `Rectangle`.
`poly` - The `Polygon` to examine.

Returns:
The `LinkedList` of run length `Rectangle`s.

---

rectangleListToBitmask

```
private static int[][] rectangleListToBitmask(java.util.LinkedList rectangleList,
                                              java.awt.Rectangle clip,
                                              int[][] mask)
```

Convert a `LinkedList` of `Rectangle`s into an array of integers representing a bit mask.

Parameters:
`rectangleList` - The list of `Rectangle`s.
`clip` - The clipping `Rectangle`.
`mask` - A two-dimensional array of ints at least (width + 31)/32 entries wide and (height) entries tall, or null.

Returns:
An integer array representing a bit mask.

---

getBounds

```
public java.awt.Rectangle getBounds()
```

Returns the bounds of the mask as a `Rectangle`.

Overrides:
getBounds in class ROI

---

getBounds2D

```
public java.awt.geom.Rectangle2D getBounds2D()
```

Returns the bounds of the mask as a `Rectangle2D`.

Overrides:
getBounds2D in class ROI

---

contains

```
public boolean contains(java.awt.Point p)
```

Returns `true` if the mask contains a given Point.

Parameters:
`p` - a Point specifying the coordinates of the pixel to be queried.

Returns:
`true` if the pixel lies within the mask.

Throws:
NullPointerException - is p is null.

Overrides:
contains in class ROI

---

contains

```
public boolean contains(java.awt.geom.Point2D p)
```

Returns `true` if the mask contains a given Point2D.

Parameters:
`p` - A Point2D specifying the coordinates of the pixel to be queried.

Returns:
`true` if the pixel lies within the mask.
Throws:
NullPointerException - is p is null.
Overrides:
contains in class ROI

---

contains

```
public boolean contains(int x,
                        int y)
```

Returns `true` if the mask contains the point (x, y).
Parameters:
`x` - An int specifying the X coordinate of the pixel to be queried.
`y` - An int specifying the Y coordinate of the pixel to be queried.
Returns:
`true` if the pixel lies within the mask.
Overrides:
contains in class ROI

---

contains

```
public boolean contains(double x,
                        double y)
```

Returns `true` if the mask contains the point (x, y).
Parameters:
`x` - A double specifying the X coordinate of the pixel to be queried.
`y` - A double specifying the Y coordinate of the pixel to be queried.
Returns:
`true` if the pixel lies within the mask.
Overrides:
contains in class ROI

---

contains

```
public boolean contains(java.awt.Rectangle rect)
```

Returns `true` if a given `Rectangle` is entirely included within the mask.
Parameters:
`rect` - A `Rectangle` specifying the region to be tested for inclusion.
Returns:
`true` if the rectangle is entirely contained within the mask.
Throws:
NullPointerException - is rect is null.
Overrides:
contains in class ROI

---

contains

```
public boolean contains(java.awt.geom.Rectangle2D rect)
```

Returns `true` if a given `Rectangle2D` is entirely included within the mask.
Parameters:
`rect` - A `Rectangle2D` specifying the region to be tested for inclusion.
Returns:
`true` if the rectangle is entirely contained within the mask.
Throws:
NullPointerException - is rect is null.
Overrides:
contains in class ROI

---

contains

```
public boolean contains(int x,
                        int y,
                        int w,
                        int h)
```

Returns `true` if a given rectangle (x, y, w, h) is entirely included within the mask.

Parameters:

`x` - The int X coordinate of the upper left corner of the region.
`y` - The int Y coordinate of the upper left corner of the region.
`w` - The int width of the region.
`h` - The int height of the region.

Returns:

`true` if the rectangle is entirely contained within the mask.

Overrides:

contains in class ROI

---

contains

```
public boolean contains(double x,
                        double y,
                        double w,
                        double h)
```

Returns `true` if a given rectangle (x, y, w, h) is entirely included within the mask.

Parameters:

`x` - The double X coordinate of the upper left corner of the region.
`y` - The double Y coordinate of the upper left corner of the region.
`w` - The double width of the region.
`h` - The double height of the region.

Returns:

`true` if the rectangle is entirely contained within the mask.

Overrides:

contains in class ROI

---

intersects

```
public boolean intersects(java.awt.Rectangle r)
```

Returns `true` if a given `Rectangle` intersects the mask.

Parameters:

`r` - A `Rectangle` specifying the region to be tested for inclusion.

Returns:

`true` if the rectangle intersects the mask.

Throws:

NullPointerException - is r is null.

Overrides:

intersects in class ROI

---

intersects

```
public boolean intersects(java.awt.geom.Rectangle2D r)
```

Returns `true` if a given `Rectangle2D` intersects the mask.

Parameters:

`r` - A `Rectangle2D` specifying the region to be tested for inclusion.

Returns:

`true` if the rectangle intersects the mask.

Throws:

NullPointerException - is r is null.

Overrides:

intersects in class ROI

---

intersects

```
public boolean intersects(int x,
                          int y,
                          int w,
                          int h)
```

Returns `true` if a given rectangle (x, y, w, h) intersects the mask.

Parameters:

`x` - The int X coordinate of the upper left corner of the region.
`y` - The int Y coordinate of the upper left corner of the region.
`w` - The int width of the region.
`h` - The int height of the region.

Returns:

`true` if the rectangle intersects the mask.

Overrides:

intersects in class ROI

---

intersects

```
public boolean intersects(double x,
                          double y,
                          double w,
                          double h)
```

Returns `true` if a given rectangle (x, y, w, h) intersects the mask.

Parameters:

`x` - The double X coordinate of the upper left corner of the region.
`y` - The double Y coordinate of the upper left corner of the region.
`w` - The double width of the region.
`h` - The double height of the region.

Returns:

`true` if the rectangle intersects the mask.

Overrides:

intersects in class ROI

---

add

```
public ROI add(ROI roi)
```

Adds another mask to this one. This operation may force this mask to be rendered.

Parameters:

`roi` - A ROI.

Throws:

java.lang.IllegalArgumentException - is roi is null.

Overrides:

add in class ROI

---

subtract

```
public ROI subtract(ROI roi)
```

Subtracts another mask from this one. This operation may force this mask to be rendered.

Parameters:

`roi` - A ROI.

Throws:

java.lang.IllegalArgumentException - is roi is null.

Overrides:

subtract in class ROI

---

intersect

```
public ROI intersect(ROI roi)
```

Sets the mask to its intersection with another mask. This operation may force this mask to be rendered.

Parameters:

`roi` - A ROI.

Throws:

java.lang.IllegalArgumentException - is roi is null.

Overrides:
intersect in class ROI

---

exclusiveOr

```
public ROI exclusiveOr(ROI roi)
```

Sets the mask to its exclusive-or with another mask. This operation may force this mask to be rendered.

Parameters:
`roi` - A ROI.

Throws:
java.lang.IllegalArgumentException - is roi is null.

Overrides:
exclusiveOr in class ROI

---

getAsShape

```
public java.awt.Shape getAsShape()
```

Returns the internal Shape representation or null if a shape representation is not possible.

Overrides:
getAsShape in class ROI

---

getAsImage

```
public PlanarImage getAsImage()
```

Returns the shape as a PlanarImage. This requires performing an antialiased rendering of the internal Shape. A BufferedImage of type TYPE_BYTE_GRAY is used internally.

Overrides:
getAsImage in class ROI

---

transform

```
public ROI transform(java.awt.geom.AffineTransform at)
```

Transforms the current contents of the `ROI` by a given `AffineTransform`.

Parameters:
`at` - An `AffineTransform` object.

Throws:
NullPointerException - if at is null.

Overrides:
transform in class ROI

---

getAsBitmask

```
public int[][] getAsBitmask(int x,
                            int y,
                            int width,
                            int height,
                            int[][] mask)
```

Returns a bitmask for a given rectangular region of the ROI indicating whether the pixel is included in the region of interest. The results are packed into 32-bit integers, with the MSB considered to lie on the left. The last entry in each row of the result may have bits that lie outside of the requested rectangle. These bits are guaranteed to be zeroed.

The `mask` array, if supplied, must be of length equal to or greater than `height` and each of its subarrays must have length equal to or greater than (width + 31)/32. If `null` is passed in, a suitable array will be constructed. If the mask is non-null but has insufficient size, an exception will be thrown.

Parameters:
`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.
`height` - The height of the rectangle.
`mask` - A two-dimensional array of ints at least (width + 31)/32 entries wide and (height) entries tall, or null.

Returns:
A reference to the `mask` parameter, or to a newly constructed array if `mask` is `null`.

Overrides:
getAsBitmask in class ROI

getAsRectangleList

```
public java.util.LinkedList getAsRectangleList(int x,
                                               int y,
                                               int width,
                                               int height)
```

Returns a `LinkedList` of `Rectangle`s for a given rectangular region of the ROI. The `Rectangle`s in the list are merged into a minimal set.

Parameters:

`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.
`height` - The height of the rectangle.

Returns:

A `LinkedList` of `Rectangle`s.

Overrides:

getAsRectangleList in class ROI

getAsRectangleList

```
protected java.util.LinkedList getAsRectangleList(int x,
                                                  int y,
                                                  int width,
                                                  int height,
                                                  boolean mergeRectangles)
```

Returns a `LinkedList` of `Rectangle`s for a given rectangular region of the ROI.

Parameters:

`x` - The X coordinate of the upper left corner of the rectangle.
`y` - The Y coordinate of the upper left corner of the rectangle.
`width` - The width of the rectangle.
`height` - The height of the rectangle.
`mergeRectangles` - `true` if the `Rectangle`s are to be merged into a minimal set.

Returns:

A `LinkedList` of `Rectangle`s.

Overrides:

getAsRectangleList in class ROI

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the `ROIShape`.

Parameters:

`out` - The `ObjectOutputStream`.

Overrides:

writeObject in class ROI

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the `ROIShape`.

Parameters:

`in` - The `ObjectInputStream`.

Overrides:

readObject in class ROI

javax.media.jai
Class RasterAccessor

```
java.lang.Object
  |
  +--javax.media.jai.RasterAccessor
```

public class **RasterAccessor**
extends java.lang.Object

An adapter class for presenting image data in a ComponentSampleModel format, even if the data isn't stored that way. RasterAccessor is meant to make the common (ComponentSampleModel) case fast and other formats possible without forcing the OpImage writer to cover more than one case per data type.

Field Detail

COPY_MASK_SHIFT

`private static final int COPY_MASK_SHIFT`

Value indicating how far COPY_MASK info is shifted to avoid interfering with the data type info.

COPY_MASK_SIZE

`private static final int COPY_MASK_SIZE`

COPY_MASK

`public static final int COPY_MASK`

The bits of a FormatTag associated with how dataArrays are obtained.

UNCOPIED

`public static final int UNCOPIED`

Flag indicating data is raster's data.

COPIED

`public static final int COPIED`

Flag indicating data is a copy of the raster's data.

EXPANSION_MASK_SHIFT

`private static final int EXPANSION_MASK_SHIFT`

Value indicating how far EXPANSION_MASK info is shifted to avoid interfering with the data type info.

EXPANSION_MASK_SIZE

`private static final int EXPANSION_MASK_SIZE`

Value indicating how many bits the EXPANSION_MASK is

EXPANSION_MASK

`public static final int EXPANSION_MASK`

The bits of a FormatTag associated with how ColorModels are used.

DEFAULTEXPANSION

`public static final int` **`DEFAULTEXPANSION`**

Flag indicating ColorModel data should be used only in copied case

---

EXPANDED

`public static final int` **`EXPANDED`**

Flag indicating ColorModel data should be interpreted.

UNEXPANDED

`public static final int` **`UNEXPANDED`**

Flag indicating ColorModel info should be ignored

DATATYPE_MASK

`public static final int` **`DATATYPE_MASK`**

The bits of a FormatTagID associated with pixel datatype.

TAG_BYTE_UNCOPIED

`public static final int` **`TAG_BYTE_UNCOPIED`**

FormatTagID indicating data in byte arrays and uncopied.

TAG_USHORT_UNCOPIED

`public static final int` **`TAG_USHORT_UNCOPIED`**

FormatTagID indicating data in unsigned short arrays and uncopied.

TAG_SHORT_UNCOPIED

`public static final int` **`TAG_SHORT_UNCOPIED`**

FormatTagID indicating data in short arrays and uncopied.

TAG_INT_UNCOPIED

`public static final int` **`TAG_INT_UNCOPIED`**

FormatTagID indicating data in int arrays and uncopied.

TAG_FLOAT_UNCOPIED

`public static final int` **`TAG_FLOAT_UNCOPIED`**

FormatTagID indicating data in float arrays and uncopied.

TAG_DOUBLE_UNCOPIED

`public static final int` **`TAG_DOUBLE_UNCOPIED`**

FormatTagID indicating data in double arrays and uncopied.

TAG_INT_COPIED

`public static final int` **`TAG_INT_COPIED`**

FormatTagID indicating data in int arrays and copied.

TAG_FLOAT_COPIED

`public static final int` **`TAG_FLOAT_COPIED`**

FormatTagID indicating data in float arrays and copied.

TAG_DOUBLE_COPIED

`public static final int` **`TAG_DOUBLE_COPIED`**

FormatTagID indicating data in double arrays and copied.

TAG_BYTE_EXPANDED

`public static final int` **`TAG_BYTE_EXPANDED`**

FormatTagID indicating data in byte arrays and uncopied.

---

raster

`protected java.awt.image.Raster` **`raster`**

The raster that is the source of pixel data.

---

rectWidth

`protected int` **`rectWidth`**

The width of the rectangle this RasterAccessor addresses.

---

rectHeight

`protected int` **`rectHeight`**

The height of the rectangle this RasterAccessor addresses.

---

rectX

`protected int` **`rectX`**

The x of the rectangle this RasterAccessor addresses.

---

rectY

`protected int` **`rectY`**

The y of the rectangle this RasterAccessor addresses.

---

formatTagID

`protected int` **`formatTagID`**

Tag indicating the data type of the data and whether its copied

---

byteDataArrays

`protected byte[][]` **`byteDataArrays`**

The image data in a two-dimensional byte array. This value will be non-null only if getDataType() returns DataBuffer.TYPE_BYTE. byteDataArays.length will equal numBands. Note that often the numBands subArrays will all point to the same place in memory.

---

shortDataArrays

`protected short[][]` **`shortDataArrays`**

The image data in a two-dimensional short array. This value will be non-null only if getDataType() returns DataBuffer.TYPE_USHORT or DataBuffer.TYPE_SHORT. shortDataArays.length will equal numBands. Note that often the numBands subArrays will all point to the same place in memory.

---

intDataArrays

`protected int[][] **intDataArrays**`

The image data in a two-dimensional int array. This value will be non-null only if getDataType() returns DataBuffer.TYPE_INT. intDataArays.length will equal numBands. Note that often the numBands subArrays will all point to the same place in memory.

---

floatDataArrays

`protected float[][] **floatDataArrays**`

The image data in a two-dimensional float array. This value will be non-null only if getDataType() returns DataBuffer.TYPE_FLOAT. floatDataArays.length will equal numBands. Note that often the numBand subArrays will all point to the same place in memory.

---

doubleDataArrays

`protected double[][] **doubleDataArrays**`

The image data in a two-dimensional double array. This value will be non-null only if getDataType() returns DataBuffer.TYPE_DOUBLE. doubleDataArays.length will equal numBands. Note that often the numBand subArrays will all point to the same place in memory.

---

bandDataOffsets

`protected int[] **bandDataOffsets**`

The bandOffset + subRasterOffset + DataBufferOffset into each of the numBand data arrays

---

bandOffsets

`protected int[] **bandOffsets**`

Offset from a pixel’s offset to a band of that pixel

---

numBands

`protected int **numBands**`

The number of bands per pixel in the data array.

---

scanlineStride

`protected int **scanlineStride**`

The scanline stride of the image data in each data array

---

pixelStride

`protected int **pixelStride**`

The pixel stride of the image data in each data array

Constructor Detail

RasterAccessor

```
public RasterAccessor(java.awt.image.Raster raster,
                      java.awt.Rectangle rect,
                      RasterFormatTag rft,
                      java.awt.image.ColorModel theColorModel)
```

Constructs a RasterAccessor object out of a Raster, Rectangle and formatTagID returned from RasterFormat.findCompatibleTag().

The `RasterFormatTag` must agree with the raster’s `SampleModel` and `ColorModel`. It is best to obtain the correct tag using the `findCompatibleTags` static method.

Throws:

ClassCastException - if the data type of `RasterFormatTag` does not agree with the actual data type of the `Raster`.

Method Detail

findCompatibleTags

```
public static RasterFormatTag[] findCompatibleTags(java.awt.image.RenderedImage[] srcs,
                                                   java.awt.image.RenderedImage dst)
```

Finds the appropriate tags for the constructor, based on the SampleModel and ColorModel of all the source and destination.

---

findCompatibleTag

```
public static int findCompatibleTag(java.awt.image.SampleModel[] srcSampleModels,
                                    java.awt.image.SampleModel dstSampleModel)
```

Returns the most efficient FormatTagID that is compatible with the destination SampleModel and all source SampleModel. Since there is no `ColorModel` associated with a `SampleModel`, this method does not expand the data buffer as it has no access to the Raster's ColorModel.

---

getX

```
public int getX()
```

Returns the x coordinate of the upper-left corner of the RasterAccessor's accessible area.

---

getY

```
public int getY()
```

Returns the y coordinate of the upper-left corner of the RasterAccessor's accessible area.

---

getWidth

```
public int getWidth()
```

Returns the width of the RasterAccessor's accessible area.

---

getHeight

```
public int getHeight()
```

Returns the height of the RasterAccessor's accessible area.

---

getNumBands

```
public int getNumBands()
```

Returns the numBands of the presented area.

---

getByteDataArrays

```
public byte[][] getByteDataArrays()
```

Returns the image data as a byte array. Non-null only if getDataType = DataBuffer.TYPE_BYTE.

---

getByteDataArray

```
public byte[] getByteDataArray(int b)
```

Returns the image data as a byte array for a specific band. Non-null only if getDataType = DataBuffer.TYPE_BYTE.

---

getShortDataArrays

```
public short[][] getShortDataArrays()
```

Returns the image data as a short array. Non-null only if getDataType = DataBuffer.TYPE_USHORT or DataBuffer.TYPE_SHORT.

---

getShortDataArray

`public short[] getShortDataArray(int b)`

Returns the image data as a short array for a specific band. Non-null only if getDataType = DataBuffer.TYPE_USHORT or DataBuffer.TYPE_SHORT.

---

getIntDataArrays

`public int[][] getIntDataArrays()`

Returns the image data as an int array. Non-null only if getDataType = DataBuffer.TYPE_INT.

---

getIntDataArray

`public int[] getIntDataArray(int b)`

Returns the image data as an int array for a specific band. Non-null only if getDataType = DataBuffer.TYPE_INT.

---

getFloatDataArrays

`public float[][] getFloatDataArrays()`

Returns the image data as a float array. Non-null only if getDataType = DataBuffer.TYPE_FLOAT.

---

getFloatDataArray

`public float[] getFloatDataArray(int b)`

Returns the image data as a float array for a specific band. Non-null only if getDataType = DataBuffer.TYPE_FLOAT.

---

getDoubleDataArrays

`public double[][] getDoubleDataArrays()`

Returns the image data as a double array. Non-null only if getDataType = DataBuffer.TYPE_DOUBLE

---

getDoubleDataArray

`public double[] getDoubleDataArray(int b)`

Returns the image data as a double array for a specific band. Non-null only if getDataType = DataBuffer.TYPE_DOUBLE

---

getDataArray

`public java.lang.Object getDataArray(int b)`

Returns the image data as an Object for a specific band.

Parameters:

`b` - The index of the image band of interest.

Returns:

The data array for the requested band.

Throws:

ArrayIndexOutOfBoundsException - if b is out of bounds.

---

getBandOffsets

`public int[] getBandOffsets()`

Returns the bandDataOffsets into the dataArrays.

---

getOffsetsForBands

`public int[] getOffsetsForBands()`

Returns the offset of all band’s samples from any pixel offset.

Throws:

ArrayIndexOutOfBoundsException - if b is out of bounds.

getBandOffset

`public int getBandOffset(int b)`

Returns the offset of a specific band’s first sample into the DataBuffer including the DataBuffer’s offset.

Throws:

ArrayIndexOutOfBoundsException - if b is out of bounds.

getOffsetForBand

`public int getOffsetForBand(int b)`

Returns the offset of a specified band’s sample from any pixel offset.

Throws:

ArrayIndexOutOfBoundsException - if b is out of bounds.

getScanlineStride

`public int getScanlineStride()`

Returns the scanlineStride for the image data.

getPixelStride

`public int getPixelStride()`

Returns the pixelStride for the image data.

getDataType

`public int getDataType()`

Returns the data type of the RasterAccessor object. Note that this datatype is not necessarily the same data type as the underlying raster.

isDataCopy

`public boolean isDataCopy()`

Returns true if the RasterAccessors’s data is copied from it’s raster.

copyDataToRaster

`public void copyDataToRaster()`

Copies data back into the RasterAccessor’s raster. Note that the data is cast from the intermediate data format to the raster’s format. If clamping is needed, the call clampDataArrays() method needs to be called before calling the copyDataToRaster() method.

needsClamping

`public boolean needsClamping()`

Indicates if the RasterAccessor has a larger dynamic range than the underlying Raster. Except in special cases, where the op knows something special, this call will determine whether or not clampDataArrays() needs to be called.

clampDataArrays

`public void clampDataArrays()`

Clamps data array values to a range that the underlying raster can deal with. For example, if the underlying raster stores data as bytes, but the samples are unpacked into integer arrays by the RasterAccessor for an operation, the operation will need to call clampDataArrays() so that the data in the int arrays is restricted to the range 0..255 before a setPixels() call is made on the underlying raster. Note that some operations (for example, lookup) can guarantee that their results don’t need clamping so they can call RasterAccessor.copyDataToRaster() without first calling this function.

clampDataArray

```
private void clampDataArray(double[] hiVals,
                            double[] loVals)
```

---

toIntArray

```
private int[] toIntArray(double[] vals)
```

---

toFloatArray

```
private float[] toFloatArray(double[] vals)
```

---

clampIntArrays

```
private void clampIntArrays(int[] hiVals,
                            int[] loVals)
```

---

clampFloatArrays

```
private void clampFloatArrays(float[] hiVals,
                              float[] loVals)
```

---

clampDoubleArrays

```
private void clampDoubleArrays(double[] hiVals,
                               double[] loVals)
```

javax.media.jai
Class RasterFactory

```
java.lang.Object
  |
  +--javax.media.jai.RasterFactory
```

public class **RasterFactory**
extends java.lang.Object

A convenience class for the construction of various types of `WritableRaster` and `SampleModel` objects.

This class provides the capability of creating `Raster`s with the enumerated data types in the java.awt.image.DataBuffer.

In come cases, instances of `ComponentSampleModelJAI`, a subclass of `java.awt.image.ComponentSampleModel` are instantiated instead of `java.awt.image.BandedSampleModel` in order to work around bugs in the current release of the Java 2 SDK.

Constructor Detail

RasterFactory

```
public RasterFactory()
```

Method Detail

createInterleavedRaster

```
public static java.awt.image.WritableRaster createInterleavedRaster(int dataType,
                                                                    int width,
                                                                    int height,
                                                                    int numBands,
                                                                    java.awt.Point location)
```

Creates a `WritableRaster` based on a `PixelInterleavedSampleModel` with the specified data type, width, height, and number of bands.

The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:
`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`numBands` - The desired number of bands.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:
java.lang.IllegalArgumentException - if `numbands` is `<1`.

createInterleavedRaster

```
public static java.awt.image.WritableRaster createInterleavedRaster(int dataType,
                                                                    int width,
                                                                    int height,
                                                                    int scanlineStride,
                                                                    int pixelStride,
                                                                    int[] bandOffsets,
                                                                    java.awt.Point location)
```

Creates a `WritableRaster` based on a `PixelInterleavedSampleModel` with the specified data type, width, height, scanline stride, pixel stride, and band offsets. The number of bands is inferred from bandOffsets.length.

The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:
`dataType` - The data type of the `WritableRaster`, one of the enumerated dataType values in java.awt.image.DataBuffer.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`scanlineStride` - The desired scanline stride.
`pixelStride` - The desired pixel stride.

`bandOffsets` - An array of `int`s indicating the relative offsets of the bands within a pixel.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - if `bandOffsets` is `null`, `dataType` is not one of the enumerated dataType value of java.awt.image.DataBuffer.
java.lang.IllegalArgumentException - if the number of array elements required by the returned `WritableRaster` would exceed `Integer.MAX_VALUE`.

---

createBandedRaster

```
public static java.awt.image.WritableRaster createBandedRaster(int dataType,
                                                               int width,
                                                               int height,
                                                               int bands,
                                                               java.awt.Point location)
```

Creates a `WritableRaster` based on a `ComponentSampleModel` with the specified data type, width, height, and number of bands.

Note that the `Raster`'s `SampleModel` will be of type `ComponentSampleModel`, not `BandedSampleModel` as might be expected.

The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:

`dataType` - The data type of the `WritableRaster`, one of the enumerated dataType values in java.awt.image.DataBuffer.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`bands` - The desired number of bands.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - if `bands` is `<1`.

---

createBandedRaster

```
public static java.awt.image.WritableRaster createBandedRaster(int dataType,
                                                               int width,
                                                               int height,
                                                               int scanlineStride,
                                                               int[] bankIndices,
                                                               int[] bandOffsets,
                                                               java.awt.Point location)
```

Creates a `WritableRaster` based on a `ComponentSampleModel` with the specified data type, width, height, scanline stride, bank indices and band offsets. The number of bands is inferred from `bankIndices.length` and `bandOffsets.length`, which must be the same.

Note that the `Raster`'s `SampleModel` will be of type `ComponentSampleModel`, not `BandedSampleModel` as might be expected.

The upper left corner of the `WritableRaster` is given by the `location` argument. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:

`dataType` - The data type of the `WritableRaster`, one of the enumerated dataType values in java.awt.image.DataBuffer.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`scanlineStride` - The desired scanline stride.
`bankIndices` - An array of `int`s indicating the bank index for each band.
`bandOffsets` - An array of `int`s indicating the relative offsets of the bands within a pixel.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - if `bankIndices` is `null`, `bandOffsets` is `null`, if `bandOffsets.length` is `!=` `bankIndices.length`, if `dataType` is not one of the enumerated datatypes of java.awt.image.DataBuffer.

---

createPackedRaster

```
public static java.awt.image.WritableRaster createPackedRaster(int dataType,
                                                               int width,
                                                               int height,
                                                               int[] bandMasks,
                                                               java.awt.Point location)
```

Creates a `WritableRaster` based on a `SinglePixelPackedSampleModel` with the specified data type, width, height, and band masks. The number of bands is inferred from `bandMasks.length`.

The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:

`dataType` - The data type of the `WritableRaster`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT` or `TYPE_INT`.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - is thrown if the `dataType` is not of either TYPE_BYTE or TYPE_USHORT or TYPE_INT.

---

createPackedRaster

```
public static java.awt.image.WritableRaster createPackedRaster(int dataType,
                                                               int width,
                                                               int height,
                                                               int numBands,
                                                               int bitsPerBand,
                                                               java.awt.Point location)
```

Creates a `WritableRaster` based on a packed `SampleModel` with the specified data type, width, height, number of bands, and bits per band. If the number of bands is one, the `SampleModel` will be a `MultiPixelPackedSampleModel`.

If the number of bands is more than one, the `SampleModel` will be a `SinglePixelPackedSampleModel`, with each band having `bitsPerBand` bits. In either case, the requirements on `dataType` and `bitsPerBand` imposed by the corresponding `SampleModel` must be met.

The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used. The `dataType` parameter should be one of the enumerated values defined in the `DataBuffer` class.

Parameters:

`dataType` - The data type of the `WritableRaster`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT` or `TYPE_INT`.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`numBands` - The desired number of bands.
`bitsPerBand` - The number of bits per band.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - is thrown if the `dataType` is not of either TYPE_BYTE or TYPE_USHORT or TYPE_INT.
java.lang.IllegalArgumentException - is thrown if bitsPerBand is negative or zero.

---

createInterleavedRaster

```
public static java.awt.image.WritableRaster createInterleavedRaster(java.awt.image.DataBuffer dataBuffer,
                                                                    int width,
                                                                    int height,
                                                                    int scanlineStride,
                                                                    int pixelStride,
                                                                    int[] bandOffsets,
                                                                    java.awt.Point location)
```

Creates a `WritableRaster` based on a `PixelInterleavedSampleModel` with the specified `DataBuffer`, width, height, scanline stride, pixel stride, and band offsets. The number of bands is inferred from `bandOffsets.length`. The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used.

Parameters:

`dataBuffer` - The `DataBuffer` to be used.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`scanlineStride` - The desired scanline stride.

pixelStride - The desired pixel stride.
bandOffsets - An array of ints indicating the relative offsets of the bands within a pixel.
location - A Point indicating the starting coordinates of the WritableRaster.

Throws:

java.lang.IllegalArgumentException - if bandOffsets is null, if pixelStride*width is > scanlineStride, if dataTypeof the DataBuffer is not one the enumerated dataType value of java.awt.image.DataBuffer.

---

createBandedRaster

```
public static java.awt.image.WritableRaster createBandedRaster(java.awt.image.DataBuffer dataBuffer,
                                                               int width,
                                                               int height,
                                                               int scanlineStride,
                                                               int[] bankIndices,
                                                               int[] bandOffsets,
                                                               java.awt.Point location)
```

Creates a WritableRaster based on a ComponentSampleModel with the specified DataBuffer, width, height, scanline stride, bank indices, and band offsets. The number of bands is inferred from bankIndices.length and bandOffsets.length, which must be the same. The upper left corner of the WritableRaster is given by the location argument. If location is null, (0, 0) will be used.

Note that the Raster's SampleModel will be of type ComponentSampleModel, not BandedSampleModel as might be expected.

Parameters:

dataBuffer - The DataBuffer to be used.
width - The desired width of the WritableRaster.
height - The desired height of the WritableRaster.
scanlineStride - The desired scanline stride.
bankIndices - An array of ints indicating the bank index for each band.
bandOffsets - An array of ints indicating the relative offsets of the bands within a pixel.
location - A Point indicating the starting coordinates of the WritableRaster.

Throws:

java.lang.IllegalArgumentException - if bankIndices is null, if bandOffsets is null, if bandOffsets.length is != bankIndices.length, if dataType is not one of the enumerated datatypes of java.awt.image.DataBuffer.

---

createPackedRaster

```
public static java.awt.image.WritableRaster createPackedRaster(java.awt.image.DataBuffer dataBuffer,
                                                               int width,
                                                               int height,
                                                               int scanlineStride,
                                                               int[] bandMasks,
                                                               java.awt.Point location)
```

Creates a WritableRaster based on a SinglePixelPackedSampleModel with the specified DataBuffer, width, height, scanline stride, and band masks. The number of bands is inferred from bandMasks.length. The upper left corner of the WritableRaster is given by the location argument. If location is null, (0, 0) will be used.

Parameters:

dataBuffer - The DataBuffer to be used.
width - The desired width of the WritableRaster.
height - The desired height of the WritableRaster.
scanlineStride - The desired scanline stride.
bandMasks - An array of ints indicating the bitmasks for each band within a pixel.
location - A Point indicating the starting coordinates of the WritableRaster.

Throws:

java.lang.IllegalArgumentException - is thrown if the dataType is not of either TYPE_BYTE or TYPE_USHORT or TYPE_INT.

---

createPackedRaster

```
public static java.awt.image.WritableRaster createPackedRaster(java.awt.image.DataBuffer dataBuffer,
                                                               int width,
                                                               int height,
                                                               int bitsPerPixel,
                                                               java.awt.Point location)
```

Creates a `WritableRaster` based on a `MultiPixelPackedSampleModel` with the specified `DataBuffer`, width, height, and bits per pixel. The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used.

Parameters:

`dataBuffer` - The `DataBuffer` to be used.
`width` - The desired width of the `WritableRaster`.
`height` - The desired height of the `WritableRaster`.
`bitsPerPixel` - The desired pixel depth.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

Throws:

java.lang.IllegalArgumentException - is thrown if the `dataType` of the `dataBuffer` is not of either TYPE_BYTE or TYPE_USHORT or TYPE_INT.

---

createRaster

```
public static java.awt.image.Raster createRaster(java.awt.image.SampleModel sampleModel,
                                                 java.awt.image.DataBuffer dataBuffer,
                                                 java.awt.Point location)
```

Creates a `WritableRaster` with the specified `SampleModel` and `DataBuffer`. The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used.

Parameters:

`sampleModel` - The `SampleModel` to be used.
`dataBuffer` - The `DataBuffer` to be used.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

---

createWritableRaster

```
public static java.awt.image.WritableRaster createWritableRaster(java.awt.image.SampleModel sampleModel,
                                                                 java.awt.Point location)
```

Creates a `WritableRaster` with the specified `SampleModel`. The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used.

Parameters:

`sampleModel` - The `SampleModel` to use.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

---

createWritableRaster

```
public static java.awt.image.WritableRaster createWritableRaster(java.awt.image.SampleModel sampleModel,
                                                                 java.awt.image.DataBuffer dataBuffer,
                                                                 java.awt.Point location)
```

Creates a `WritableRaster` with the specified `SampleModel` and `DataBuffer`. The upper left corner of the `WritableRaster` is given by the `location` argument. If `location` is `null`, (0, 0) will be used.

Parameters:

`sampleModel` - The `SampleModel` to be used.
`dataBuffer` - The `DataBuffer` to be used.
`location` - A `Point` indicating the starting coordinates of the `WritableRaster`.

---

createWritableChild

```
public static java.awt.image.WritableRaster createWritableChild(java.awt.image.WritableRaster raster,
                                                                int parentX,
                                                                int parentY,
                                                                int width,
                                                                int height,
                                                                int childMinX,
                                                                int childMinY,
                                                                int[] bandList)
```

Returns a new WritableRaster which shares all or part of the supplied WritableRaster’s DataBuffer. The new WritableRaster will possess a reference to the supplied WritableRaster, accessible through its getParent() and getWritableParent() methods.

This method provides a workaround for a bug in the implementation of WritableRaster.createWritableChild in the initial relase of the Java2 platform.

The `parentX`, `parentY`, `width` and `height` parameters form a Rectangle in this WritableRaster’s coordinate space, indicating the area of pixels to be shared. An error will be thrown if this Rectangle is not contained with the bounds of the supplied WritableRaster.

The new WritableRaster may additionally be translated to a different coordinate system for the plane than that used by the supplied WritableRaster. The childMinX and childMinY parameters give the new (x, y) coordinate of the upper-left pixel of the returned WritableRaster; the coordinate (childMinX, childMinY) in the new WritableRaster will map to the same pixel as the coordinate (parentX, parentY) in the supplied WritableRaster.

The new WritableRaster may be defined to contain only a subset of the bands of the supplied WritableRaster, possibly reordered, by means of the bandList parameter. If bandList is null, it is taken to include all of the bands of the supplied WritableRaster in their current order.

To create a new WritableRaster that contains a subregion of the supplied WritableRaster, but shares its coordinate system and bands, this method should be called with childMinX equal to parentX, childMinY equal to parentY, and bandList equal to null.

Parameters:

`raster` - The parent WritableRaster.
`parentX` - X coordinate of the upper left corner of the shared rectangle in this WritableRaster's coordinates.
`parentY` - Y coordinate of the upper left corner of the shared rectangle in this WritableRaster's coordinates.
`width` - Width of the shared rectangle starting at (`parentX`, `parentY`).
`height` - Height of the shared rectangle starting at (`parentX`, `parentY`).
`childMinX` - X coordinate of the upper left corner of the returned WritableRaster.
`childMinY` - Y coordinate of the upper left corner of the returned WritableRaster.
`bandList` - Array of band indices, or null to use all bands.

Throws:

java.awt.image.RasterFormatException - if the subregion is outside of the raster bounds.

---

createBandedSampleModel

```
public static java.awt.image.SampleModel createBandedSampleModel(int dataType,
                                                                 int width,
                                                                 int height,
                                                                 int numBands,
                                                                 int[] bankIndices,
                                                                 int[] bandOffsets)
```

Creates a banded `SampleModel` with a given data type, width, height, number of bands, bank indices, and band offsets.

Note that the returned `SampleModel` will be of type `ComponentSampleModel`, not `BandedSampleModel` as might be expected. Its behavior will be equivalent to that of a `BandedSampleModel`, and in particular its pixel stride will always be 1.

Parameters:

`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `SampleModel`.
`height` - The desired height of the `SampleModel`.
`numBands` - The desired number of bands.
`bankIndices` - An array of `int`s indicating the bank index for each band.
`bandOffsets` - An array of `int`s indicating the relative offsets of the bands within a pixel.

Throws:

java.lang.IllegalArgumentException - if `numBands` is `<1`, if `bandOffsets.length` is `!=` `bankIndices.length`.

---

createBandedSampleModel

```
public static java.awt.image.SampleModel createBandedSampleModel(int dataType,
                                                                 int width,
                                                                 int height,
                                                                 int numBands)
```

Creates a banded `SampleModel` with a given data type, width, height, and number of bands. The bank indices and band offsets are set to default values.

Note that the returned `SampleModel` will be of type `ComponentSampleModel`, not `BandedSampleModel` as might be expected. Its behavior will be equivalent to that of a `BandedSampleModel`, and in particular its pixel stride will always be 1.

Parameters:

`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `SampleModel`.
`height` - The desired height of the `SampleModel`.
`numBands` - The desired number of bands.

---

createPixelInterleavedSampleModel

```
public static java.awt.image.SampleModel createPixelInterleavedSampleModel(int dataType,
                                                                            int width,
                                                                            int height,
                                                                            int pixelStride,
                                                                            int scanlineStride,
                                                                            int[] bandOffsets)
```

Creates a pixel interleaved `SampleModel` with a given data type, width, height, pixel and scanline strides, and band offsets.

Parameters:

`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `SampleModel`.
`height` - The desired height of the `SampleModel`.
`pixelStride` - The desired pixel stride.
`scanlineStride` - The desired scanline stride.
`bandOffsets` - An array of `int`s indicating the relative offsets of the bands within a pixel.

Throws:

java.lang.IllegalArgumentException - if `bandOffsets` is `null`, if the `pixelStride*width` is `>` than `scanlineStride`, if the `dataType` is not one of the above mentioned datatypes.

---

createPixelInterleavedSampleModel

```
public static java.awt.image.SampleModel createPixelInterleavedSampleModel(int dataType,
                                                                            int width,
                                                                            int height,
                                                                            int numBands)
```

Creates a pixel interleaved `SampleModel` with a given data type, width, height, and number of bands. The pixel stride, scanline stride, and band offsets are set to default values.

Parameters:

`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `SampleModel`.
`height` - The desired height of the `SampleModel`.
`numBands` - The desired number of bands.

Throws:

java.lang.IllegalArgumentException - if `numBands` is `<1`.

---

createComponentSampleModel

```
public static java.awt.image.SampleModel createComponentSampleModel(java.awt.image.SampleModel sm,
                                                                     int dataType,
                                                                     int width,
                                                                     int height,
                                                                     int numBands)
```

Creates a component `SampleModel` with a given data type, width, height, and number of bands that is "compatible" with a given SampleModel.

Parameters:

`sm` - The `SampleModel` to be compatible with.
`dataType` - The data type of the `SampleModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_SHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`width` - The desired width of the `SampleModel`.
`height` - The desired height of the `SampleModel`.
`numBands` - The desired number of bands.

---

createComponentColorModel

```
public static java.awt.image.ComponentColorModel createComponentColorModel(int dataType,
                                                                            java.awt.color.ColorSpace colorSpace,
                                                                            boolean useAlpha,
                                                                            boolean premultiplied,
                                                                            int transparency)
```

Creates a component-based `ColorModel` with a given data type, color space, and transparency type. Currently this method does not support data type `DataBuffer.TYPE_SHORT`.

Parameters:

`dataType` - The data type of the `ColorModel`, one of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, `TYPE_INT`, `TYPE_FLOAT`, or `TYPE_DOUBLE`.
`colorSpace` - An instance of `ColorSpace`.
`useAlpha` - `true` if alpha is to be used.
`premultiplied` - `true` if alpha values are premultiplied. If `useAlpha` is `false`, the value of `premultiplied` is ignored.
`transparency` - One of `Transparency.OPAQUE`, `Transparency.BITMASK`, or `Transparency.TRANSLUCENT`. If `useAlpha` is `false`, the value of `transparency` is ignored. If `useAlpha` is `true`, `transparency` must not equal `Transparency.OPQAUE`.

Throws:

NullPointerException - if `colorSpace` is `null`.
java.lang.IllegalArgumentException - if `transparency` has an unknown value, if `useAlpha == true` but `transparency == Transparency.OPAQUE`, or if `dataType` is not one of the standard types listed above.

javax.media.jai

Class RasterFormatTag

```
java.lang.Object
  |
  +--javax.media.jai.RasterFormatTag
```

public final class **RasterFormatTag**
extends java.lang.Object

This class encapsulates the information needed for RasterAccessor to understand how a Raster is laid out. It's designed so that one RasterFormatTag can be constructed per source and that RasterFormatTag can cache information that the RasterAccessor would otherwise have to extract from the Raster each time it's constructed (generally each time OpImage.computeRect() is called.) Additionally, it can cache various arrays (i.e. bankIndices[] and bandOffsets[]) that that would otherwise be cloned everytime they were requested. Because of the way SampleModel.createCompatibleSampleModel() is designed not all fields of a particular SampleModel will match those of the SampleModel returned by SampleModel.createCompatibleSampleModel(). Values like pixelStride and numBands won't change, but values like bankIndicies[] and bandOffsets[] might if the underlying Raster is not pixelSequential. Rasters which are pixelSequential meet the following conditions 1) The SampleModel is a ComponentSampleModel. 2) The pixelStride is equal to the number of bands. 3) All the bankIndices[] are equal. 4) All the bandOffsets[] values are less than pixelStride 5) No two bandOffsets[] values are equal. For that reason, RasterFormatTags representing non pixelSequential rasters don't attempt to cache the bandOffsets[] or bankIndices[]. For such rasters, this information should be taken directly from the raster itself. Note that any RasterFormatTag that will cause data to be copied from the Raster will be pixelSequential as that is the format in which data is returned from Raster.getPixels() returns.

Field Detail

COPY_MASK

`private static final int` **`COPY_MASK`**

UNCOPIED

`private static final int` **`UNCOPIED`**

COPIED

`private static final int` **`COPIED`**

formatTagID

`private int` **`formatTagID`**

bankIndices

`private int[]` **`bankIndices`**

numBands

`private int` **`numBands`**

bandOffsets

`private int[]` **`bandOffsets`**

pixelStride

`private int` **`pixelStride`**

isPixelSequential

```
private boolean isPixelSequential
```

Constructor Detail

RasterFormatTag

```
public RasterFormatTag(java.awt.image.SampleModel sampleModel,
                       int formatTagID)
```

Constructs a RasterFormatTag given a sampleModel and a formatTagID. Generally, this constructor is called by RasterAccessor.findCompatibleTags(RenderedImage[] srcs, RenderedImage dst) and it takes care of setting the values correctly. In special cases, OpImages need to construct a RasterFormatTag without creating a RenderedImage. In this case a RasterFormatTag can be created using a formatTagID returned from RasterAccessor.findCompatibleTag(SampleModel[] srcs, SampleModel dst) and a sampleModel that was either passed in to the findCompatibleTag() call or one that was created using createCompatibleSampleModel() on one of the passed in SampleModels. Attempting to use arbitrary SampleModels with arbitrary formatTagIDs has undefined results.

Method Detail

isPixelSequential

```
public final boolean isPixelSequential()
```

Returns whether or not the SampleModel represented by the RasterFormatTag is PixelSequential. Note that RasterFormatTag's that indicate data should be copied out of the Raster by the RasterAccessor will always return true for isPixelSequential(). RasterFormatTags that indicate no copying is needed will only return true, if 1) The SampleModel is a ComponentSampleModel. 2) The pixelStride is equal to the number of bands. 3) All the bankIndices[] are equal. 4) All the bandOffsets[] values are less than pixelStride 5) No two bandOffset values are equal.

---

getFormatTagID

```
public final int getFormatTagID()
```

Returns the FormatTagID used to construct this RasterFormatTag. Valid values are defined in javax.media.jai.RasterAccessor.

---

getBankIndices

```
public final int[] getBankIndices()
```

Returns the bankIndices for the Raster if isPixelSequential() is true. Returns null otherwise. In the COPIED case, the bankIndices will all be 0.

---

getNumBands

```
public final int getNumBands()
```

Returns the number of bands in the underlying Raster

---

getBandOffsets

```
public final int[] getBandOffsets()
```

Returns the bandOffsets for the Raster if isPixelSequential() is true. Returns null otherwise. In the COPIED case, bankIndices will be numBands sequential integers starting with 0.

---

getPixelStride

```
public final int getPixelStride()
```

Returns the pixelStride of the underlying Raster

javax.media.jai
Class RegistryInitData

```
java.lang.Object
  |
  +--javax.media.jai.RegistryInitData
```

---

class **RegistryInitData**
extends java.lang.Object

Field Detail

descTable

`java.util.Hashtable descTable`

---

rifTable

`java.util.Hashtable rifTable`

---

crifTable

`java.util.Hashtable crifTable`

---

cifTable

`java.util.Hashtable cifTable`

---

prodPref

`java.util.Vector prodPref`

---

rifPref

`java.util.Vector rifPref`

---

cifPref

`java.util.Vector cifPref`

Constructor Detail

RegistryInitData

```
RegistryInitData(java.util.Hashtable descTable,
                 java.util.Hashtable rifTable,
                 java.util.Hashtable crifTable,
                 java.util.Hashtable cifTable,
                 java.util.Vector prodPref,
                 java.util.Vector rifPref,
                 java.util.Vector cifPref)
```

javax.media.jai
Class RemoteImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.RemoteImage
```

---

public class **RemoteImage**
extends PlanarImage

A sub-class of `PlanarImage` which represents an image on a remote server machine.

The image may be constructed from a RenderedImage or from an imaging chain in either the rendered or renderable mode. Network errors (detected via throws of RemoteExceptions) are dealt with through retries; when the limit of retries is exceeded, a null Raster may be returned.

Note that the registry of the server will be used. In particular if an `OperationRegistry` was present in the `RenderingHints` used to construct a `RenderedOp` or `RenderableOp` it will not be serialized and transmitted to the server.

Image layout attributes, once requested, are cached locally for speed.

Field Detail

DEFAULT_TIMEOUT

`static final int` **`DEFAULT_TIMEOUT`**

The amount of time to wait between retries.

DEFAULT_NUM_RETRIES

`static final int` **`DEFAULT_NUM_RETRIES`**

The default number of retries.

VAR_MIN_X

`static final int` **`VAR_MIN_X`**

Index of local variable.

VAR_MIN_Y

`static final int` **`VAR_MIN_Y`**

Index of local variable.

VAR_WIDTH

`static final int` **`VAR_WIDTH`**

Index of local variable.

VAR_HEIGHT

`static final int` **`VAR_HEIGHT`**

Index of local variable.

VAR_TILE_WIDTH

`static final int` **`VAR_TILE_WIDTH`**

Index of local variable.

VAR_TILE_HEIGHT

`static final int VAR_TILE_HEIGHT`

Index of local variable.

VAR_TILE_GRID_X_OFFSET

`static final int VAR_TILE_GRID_X_OFFSET`

Index of local variable.

VAR_TILE_GRID_Y_OFFSET

`static final int VAR_TILE_GRID_Y_OFFSET`

Index of local variable.

VAR_SAMPLE_MODEL

`static final int VAR_SAMPLE_MODEL`

Index of local variable.

VAR_COLOR_MODEL

`static final int VAR_COLOR_MODEL`

Index of local variable.

VAR_SOURCES

`static final int VAR_SOURCES`

Index of local variable.

NUM_VARS

`static final int NUM_VARS`

Index of local variable.

NULL_PROPERTY_CLASS

`private static final java.lang.Class NULL_PROPERTY_CLASS`

---

remoteImage

`protected com.sun.media.jai.rmi.RMIImage remoteImage`

The RMIImage our data will come from.

---

id

`private java.lang.Long id`

---

fieldValid

`protected boolean[] fieldValid`

Valid bits for locally cached variables.

---

propertyNames

`protected java.lang.String[] propertyNames`

Locally cached version of properties.

---

timeout

```
protected int timeout
```

The amount of time between retries (milliseconds).

---

numRetries

```
protected int numRetries
```

The number of retries.

---

imageBounds

```
private java.awt.Rectangle imageBounds
```

Constructor Detail

RemoteImage

```
public RemoteImage(java.lang.String serverName,
                   java.awt.image.RenderedImage source)
```

Constructs a RemoteImage from a RenderedImage.

The RenderedImage source should ideally be a lightweight reference to an image available locally on the server or over a further network link.

Although it is legal to use any RenderedImage, one should be aware that this will require copying of the image data via transmission over a network link.

The name of the server must be supplied in the form appropriate to the implementation. In the reference port of JAI, RMI is used to implement remote imaging so that the server name must be supplied in the format

```
 host:port
```

where the port number is optional and may be supplied only if the host name is supplied. If this parameter is null the default is to search for the RMIImage service on the local host at the default *rmiregistry* port (1099).

Parameters:

`serverName` - The name of the server in the approriate format.
`source` - A RenderedImage source which must not be null.

Throws:

java.lang.IllegalArgumentException - if `source` is `null`.

RemoteImage

```
public RemoteImage(java.lang.String serverName,
                   RenderedOp source)
```

Constructs a RemoteImage from a RenderedOp, i.e., an imaging directed acyclic graph (DAG).

This DAG will be copied over to the server where it will be transformed into an OpImage chain using the server's local OperationRegistry and available RenderedImageFactory objects.

The name of the server must be supplied in the form appropriate to the implementation. In the reference port of JAI, RMI is used to implement remote imaging so that the server name must be supplied in the format

```
 host:port
```

where the port number is optional and may be supplied only if the host name is supplied. If this parameter is null the default is to search for the RMIImage service on the local host at the default *rmiregistry* port (1099).

Note that the properties of the `RemoteImage` will be those of the `RenderedOp` node and not of its rendering.

Parameters:

`serverName` - The name of the server in the approriate format.
`source` - A RenderedOp source which must not be null.

Throws:

java.lang.IllegalArgumentException - if `source` is `null`.

RemoteImage

```
public RemoteImage(java.lang.String serverName,
                   RenderableOp source,
                   java.awt.image.renderable.RenderContext renderContext)
```

Constructs a RemoteImage from a RenderableOp and RenderContext. The entire RenderableOp DAG will be copied over to the server.

The name of the server must be supplied in the form appropriate to the implementation. In the reference port of JAI, RMI is used to implement remote imaging so that the server name must be supplied in the format

```
 host:port
```

where the port number is optional and may be supplied only if the host name is supplied. If this parameter is null the default is to search for the RMIImage service on the local host at the default *rmiregistry* port (1099).

Note that the properties of the `RemoteImage` will be those of the `RenderableOp` node and not of its rendering.

Parameters:

`serverName` - The name of the server in the approriate format.
`source` - A RenderableOp source which must not be null.
`renderContext` - The rendering context which may be null.

Throws:

java.lang.IllegalArgumentException - if `source` is `null`.

Method Detail

getRMIImage

`private void **getRMIImage**(java.lang.String serverName)`

Construct an RMIImage on the indicated server.

The name of the server must be supplied in the form

```
 host:port
```

where the port number is optional and may be supplied only if the host name is supplied. If this parameter is null the default is to search for the RMIImage service on the local host at the default *rmiregistry* port (1099).

The result is cached in the instance variable "remoteImage".

Parameters:

`serverName` - The name of the server in the format described.

---

getRMIID

`private void **getRMIID**()`

Get the unique ID to be used to refer to this object on the server. The result is cached in the instance variable "id".

---

setRMIProperties

`private void **setRMIProperties**(java.lang.String serverName)`

Cache the argument and the RMI ID as local properties. This is a gross hack to permit chaining of remote images.

Parameters:

`serverName` - The server name as described in the constructors.

---

finalize

`protected void **finalize**()`

Overrides:

finalize in class PlanarImage

---

setTimeout

`public void **setTimeout**(int timeout)`

Set the amount of time between retries.

Parameters:

`timeout` - The time interval between retries (milliseconds). If this is non-positive the time interval is not changed.

---

getTimeout

`public int **getTimeout**()`

Gets the amount of time between retries.

---

setNumRetries

```
public void setNumRetries(int numRetries)
```

Set the number of retries.

Parameters:

`numRetries` - The number of retries. If this is non-positive the number of retries is not changed.

---

getNumRetries

```
public int getNumRetries()
```

Gets the number of retries.

---

requestField

```
protected void requestField(int fieldIndex,
                            int retries,
                            int timeout)
```

Cause an instance variable of the remote object to be cached locally, retrying a given number of times with a given timeout.

Parameters:

`fieldIndex` - the index of the desired field.
`retries` - the maximum number of retries; must be positive.
`timeout` - the timeout interval between retries, in milliseconds; must be positive.

Throws:

`ArrayIndexOutOfBoundsException` - if fieldIndex is is negative or >= NUM_VARS.
`IllegalArgumentException` - if retries or timeout is non-positive.

---

requestField

```
protected void requestField(int fieldIndex)
```

Causes an instance variable of the remote object to be cached locally, retrying indefinitely with a default timeout of 1 second.

Parameters:

`fieldIndex` - the index of the desired field.

Throws:

`ArrayIndexOutOfBoundsException` - if fieldIndex is is negative or >= NUM_VARS.

---

getMinX

```
public int getMinX()
```

Returns the X coordinate of the leftmost column of the image.

Overrides:

getMinX in class PlanarImage

---

getMaxX

```
public int getMaxX()
```

Returns the X coordinate of the column immediately to the right of the rightmost column of the image.

Overrides:

getMaxX in class PlanarImage

---

getMinY

```
public int getMinY()
```

Returns the Y coordinate of the uppermost row of the image.

Overrides:

getMinY in class PlanarImage

---

getMaxY

```
public int getMaxY()
```

Returns the Y coordinate of the row immediately below the bottom row of the image.

Overrides:

getMaxY in class PlanarImage

getWidth

`public int **getWidth**()`

Returns the width of the RemoteImage in pixels.
Overrides:
getWidth in class PlanarImage

getHeight

`public int getHeight()`

Returns the height of the RemoteImage in pixels.
Overrides:
getHeight in class PlanarImage

getTileWidth

`public int getTileWidth()`

Returns the width of a tile in pixels.
Overrides:
getTileWidth in class PlanarImage

getTileHeight

`public int getTileHeight()`

Returns the height of a tile in pixels.
Overrides:
getTileHeight in class PlanarImage

getTileGridXOffset

`public int getTileGridXOffset()`

Returns the X offset of the tile grid.
Overrides:
getTileGridXOffset in class PlanarImage

getTileGridYOffset

`public int getTileGridYOffset()`

Returns the Y offset of the tile grid.
Overrides:
getTileGridYOffset in class PlanarImage

getSampleModel

`public java.awt.image.SampleModel getSampleModel()`

Returns the SampleModel associated with this image.
Overrides:
getSampleModel in class PlanarImage

getColorModel

`public java.awt.image.ColorModel getColorModel()`

Returns the ColorModel associated with this image.
Overrides:
getColorModel in class PlanarImage

getSources

```
public java.util.Vector getSources()
```

Returns a vector of RenderedImages that are the sources of image data for this RenderedImage. Note that this method will often return null.

Overrides:
getSources in class PlanarImage

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Gets a property from the property set of this image. If the property name is not recognized, java.awt.Image.UndefinedProperty will be returned.

Parameters:
`name` - the name of the property to get, as a String.

Returns:
a reference to the property Object, or the value java.awt.Image.UndefinedProperty.

Overrides:
getProperty in class PlanarImage

---

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns a list of names recognized by getProperty.

Overrides:
getPropertyNames in class PlanarImage

---

getTile

```
public java.awt.image.Raster getTile(int x,
                                     int y)
```

Returns tile (x, y). Note that x and y are indexes into the tile array not pixel locations. The Raster that is returned is a copy.

Parameters:
`x` - the X index of the requested tile in the tile array
`y` - the Y index of the requested tile in the tile array

Overrides:
getTile in class PlanarImage

---

getData

```
public java.awt.image.Raster getData()
```

Returns the image as one large tile.

Overrides:
getData in class PlanarImage

---

getData

```
public java.awt.image.Raster getData(java.awt.Rectangle rect)
```

Returns an arbitrary rectangular region of the RemoteImage.

The `rect` parameter may be `null`, in which case the entire image data is returned in the `Raster`.

If `rect` is non-`null` but does not intersect the image bounds at all, an `IllegalArgumentException` will be thrown.

Overrides:
getData in class PlanarImage

---

copyData

```
public java.awt.image.WritableRaster copyData(java.awt.image.WritableRaster raster)
```

Returns an arbitrary rectangular region of the RemoteImage in a user-supplied WritableRaster. The rectangular region is the entire image if the argument is null or the intersection of the argument bounds with the image bounds if the region is non-null. If the argument is non-null but has bounds which have an empty intersection with the image bounds the return value will be null. The return value may also be null if the argument is non-null but is incompatible with the Raster returned from the remote image.

Overrides:
copyData in class PlanarImage

javax.media.jai
Class RenderableGraphics

```
java.lang.Object
  |
  +--java.awt.Graphics
        |
        +--java.awt.Graphics2D
              |
              +--javax.media.jai.RenderableGraphics
```

public class **RenderableGraphics**
extends java.awt.Graphics2D
implements java.awt.image.renderable.RenderableImage

An implementation of `Graphics2D` with `RenderableImage` semantics. In other words, content may be drawn into the image using the `Graphics2D` interface and later be turned into `RenderedImage`s with different resolutions and characteristics.

A `RenderableGraphics` occupies a region of the plane specified at the time of construction.

The contents of `RenderableImage`s that are drawn onto a `RenderableGraphics` are accessed only at the time of rendering, not the time of drawing.

Since the methods of this class all derive from `Graphics2D` and `RenderableImage`, they are not all commented individually.

See Also:
`Graphics2D`, `RenderableImage`

Field Detail

GRAPHICS2D_CLASS

`private static final java.lang.Class` **`GRAPHICS2D_CLASS`**

dimensions

`private java.awt.geom.Rectangle2D` **`dimensions`**

opArgList

`private java.util.LinkedList` **`opArgList`**

origin

`private java.awt.Point` **`origin`**

clip

`private java.awt.Shape` **`clip`**

color

`private java.awt.Color` **`color`**

font

`private java.awt.Font` **`font`**

background

```
private java.awt.Color background
```

---

composite

```
private java.awt.Composite composite
```

---

paint

```
private java.awt.Paint paint
```

---

stroke

```
private java.awt.Stroke stroke
```

---

renderingHints

```
private java.awt.RenderingHints renderingHints
```

---

transform

```
private java.awt.geom.AffineTransform transform
```

Constructor Detail

RenderableGraphics

```
public RenderableGraphics(java.awt.geom.Rectangle2D dimensions)
```

Constructs a `RenderableGraphics` given a bounding `Rectangle2D`.

Parameters:

`dimensions` - The bounding `Rectangle2D`.

RenderableGraphics

```
private RenderableGraphics(java.awt.geom.Rectangle2D dimensions,
                           java.util.LinkedList opArgList,
                           java.awt.Point origin,
                           java.awt.Graphics2D g)
```

Constructs a `RenderableGraphics` given a bounding `Rectangle2D`, an origin, and a `Graphics2D` object from which to initialize the `RenderableGraphics` state. The `Graphics2D` may be null.

Parameters:

`dimensions` - The bounding `Rectangle2D`.
`opArgList` - The list of operations and arguments.
`dimensions` - The origin.
`dimensions` - The `Graphics2D` state source; may be null.

Method Detail

getBogusGraphics2D

```
private java.awt.Graphics2D getBogusGraphics2D()
```

Creates a bogus `Graphics2D` object to be used to retrieve information dependent on system aspects which are image-independent.

The `dispose()` method of the `Graphics2D` object returned should be called to free the associated resources as\ soon as possible.

Returns:

A `Graphics2D` object.

---

createTiledImage

```
private TiledImage createTiledImage(java.awt.RenderingHints hints,
                                    java.awt.Rectangle bounds)
```

Create a TiledImage to be used as the canvas.

Parameters:

`hints` - RenderingHints from which to derive an ImageLayout.
`bounds` - The bounding box of the TiledImage.

Returns:

A TiledImage.

---

queueOpArg

```
private void queueOpArg(java.lang.String name,
                        java.lang.Class[] argTypes,
                        java.lang.Object[] args)
```

Queue a `Graphics2D` operation and its argument list in the linked list of operations and arguments. The name of the operation and the array of class types of its arguments are used to determine the associated `Method` object. The `Method` object and array of `Object` arguments are appended to the list as an ordered pair of the form (`Method`,`Object`[]).

Parameters:

`name` - The name of the `Graphics2D` operation.
`argTypes` - An array of the `Classes` of the arguments of the specified operation.
`args` - The arguments of the operation as an array of `Object`s.

---

evaluateOpList

```
private void evaluateOpList(java.awt.Graphics2D g2d)
```

Evaulate the queue of `Graphics2D` operations on the specified `Graphics2D` object.

Parameters:

`g2d` - The `Graphics2D` on which to evaluate the operation queue.

---

create

```
public java.awt.Graphics create()
```

Overrides:

create in class java.awt.Graphics

---

getColor

```
public java.awt.Color getColor()
```

Overrides:

getColor in class java.awt.Graphics

---

setColor

```
public void setColor(java.awt.Color c)
```

Overrides:

setColor in class java.awt.Graphics

---

setPaintMode

```
public void setPaintMode()
```

Overrides:

setPaintMode in class java.awt.Graphics

---

setXORMode

```
public void setXORMode(java.awt.Color c1)
```

Overrides:

setXORMode in class java.awt.Graphics

getFont

```
public java.awt.Font getFont()
```

Overrides:
getFont in class java.awt.Graphics

---

setFont

```
public void setFont(java.awt.Font font)
```

Overrides:
setFont in class java.awt.Graphics

---

getFontMetrics

```
public java.awt.FontMetrics getFontMetrics(java.awt.Font f)
```

Overrides:
getFontMetrics in class java.awt.Graphics

---

getClipBounds

```
public java.awt.Rectangle getClipBounds()
```

Overrides:
getClipBounds in class java.awt.Graphics

---

clipRect

```
public void clipRect(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
clipRect in class java.awt.Graphics

---

setClip

```
public void setClip(int x,
                    int y,
                    int width,
                    int height)
```

Overrides:
setClip in class java.awt.Graphics

---

getClip

```
public java.awt.Shape getClip()
```

Overrides:
getClip in class java.awt.Graphics

---

setClip

```
public void setClip(java.awt.Shape clip)
```

Overrides:
setClip in class java.awt.Graphics

---

copyArea

```
public void copyArea(int x,
                     int y,
                     int width,
                     int height,
                     int dx,
                     int dy)
```

Overrides:
copyArea in class java.awt.Graphics

---

drawLine

```
public void drawLine(int x1,
                     int y1,
                     int x2,
                     int y2)
```

Overrides:
drawLine in class java.awt.Graphics

---

fillRect

```
public void fillRect(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
fillRect in class java.awt.Graphics

---

clearRect

```
public void clearRect(int x,
                      int y,
                      int width,
                      int height)
```

Overrides:
clearRect in class java.awt.Graphics

---

drawRoundRect

```
public void drawRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

Overrides:
drawRoundRect in class java.awt.Graphics

---

fillRoundRect

```
public void fillRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

Overrides:
fillRoundRect in class java.awt.Graphics

---

draw3DRect

```
public void draw3DRect(int x,
                       int y,
                       int width,
                       int height,
                       boolean raised)
```

Overrides:
draw3DRect in class java.awt.Graphics2D

---

fill3DRect

```
public void fill3DRect(int x,
                       int y,
                       int width,
                       int height,
                       boolean raised)
```

Overrides:
fill3DRect in class java.awt.Graphics2D

---

drawOval

```
public void drawOval(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
drawOval in class java.awt.Graphics

---

fillOval

```
public void fillOval(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
fillOval in class java.awt.Graphics

---

drawArc

```
public void drawArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

Overrides:
drawArc in class java.awt.Graphics

---

fillArc

```
public void fillArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

Overrides:
fillArc in class java.awt.Graphics

---

drawPolyline

```
public void drawPolyline(int[] xPoints,
                         int[] yPoints,
                         int nPoints)
```

Overrides:
drawPolyline in class java.awt.Graphics

---

drawPolygon

```
public void drawPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

Overrides:
drawPolygon in class java.awt.Graphics

---

fillPolygon

```
public void fillPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

Overrides:
fillPolygon in class java.awt.Graphics

---

drawString

```
public void drawString(java.lang.String str,
                       int x,
                       int y)
```

Overrides:
drawString in class java.awt.Graphics2D

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

dispose

```
public void dispose()
```

Overrides:
dispose in class java.awt.Graphics

---

addRenderingHints

```
public void addRenderingHints(java.util.Map hints)
```

Overrides:
addRenderingHints in class java.awt.Graphics2D

---

draw

```
public void draw(java.awt.Shape s)
```

Overrides:
draw in class java.awt.Graphics2D

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         java.awt.geom.AffineTransform xform,
                         java.awt.image.ImageObserver obs)
```

Overrides:
drawImage in class java.awt.Graphics2D

---

drawRenderedImage

```
public void drawRenderedImage(java.awt.image.RenderedImage img,
                              java.awt.geom.AffineTransform xform)
```

Overrides:
drawRenderedImage in class java.awt.Graphics2D

drawRenderableImage

```
public void drawRenderableImage(java.awt.image.renderable.RenderableImage img,
                                java.awt.geom.AffineTransform xform)
```

Overrides:
drawRenderableImage in class java.awt.Graphics2D

drawImage

```
public void drawImage(java.awt.image.BufferedImage img,
                      java.awt.image.BufferedImageOp op,
                      int x,
                      int y)
```

Overrides:
drawImage in class java.awt.Graphics2D

drawString

```
public void drawString(java.lang.String s,
                       float x,
                       float y)
```

Overrides:
drawString in class java.awt.Graphics2D

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       int x,
                       int y)
```

Overrides:
drawString in class java.awt.Graphics2D

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       float x,
                       float y)
```

Overrides:
drawString in class java.awt.Graphics2D

drawGlyphVector

```
public void drawGlyphVector(java.awt.font.GlyphVector v,
                            float x,
                            float y)
```

Overrides:
drawGlyphVector in class java.awt.Graphics2D

fill

```
public void fill(java.awt.Shape s)
```

Overrides:
fill in class java.awt.Graphics2D

hit

```
public boolean hit(java.awt.Rectangle rect,
                   java.awt.Shape s,
                   boolean onStroke)
```

Overrides:
hit in class java.awt.Graphics2D

getDeviceConfiguration

```
public java.awt.GraphicsConfiguration getDeviceConfiguration()
```

Overrides:
getDeviceConfiguration in class java.awt.Graphics2D

getFontRenderContext

```
public java.awt.font.FontRenderContext getFontRenderContext()
```

Overrides:
getFontRenderContext in class java.awt.Graphics2D

setComposite

```
public void setComposite(java.awt.Composite comp)
```

Overrides:
setComposite in class java.awt.Graphics2D

setPaint

```
public void setPaint(java.awt.Paint paint)
```

Overrides:
setPaint in class java.awt.Graphics2D

setStroke

```
public void setStroke(java.awt.Stroke s)
```

Overrides:
setStroke in class java.awt.Graphics2D

setRenderingHint

```
public void setRenderingHint(java.awt.RenderingHints.Key hintKey,
                             java.lang.Object hintValue)
```

Overrides:
setRenderingHint in class java.awt.Graphics2D

getRenderingHint

```
public java.lang.Object getRenderingHint(java.awt.RenderingHints.Key hintKey)
```

Overrides:
getRenderingHint in class java.awt.Graphics2D

setRenderingHints

```
public void setRenderingHints(java.util.Map hints)
```

Overrides:
setRenderingHints in class java.awt.Graphics2D

getRenderingHints

```
public java.awt.RenderingHints getRenderingHints()
```

Overrides:
getRenderingHints in class java.awt.Graphics2D

translate

```
public void translate(int x,
                      int y)
```

Overrides:
translate in class java.awt.Graphics2D

---

translate

```
public void translate(double x,
                      double y)
```

Overrides:
translate in class java.awt.Graphics2D

---

rotate

```
public void rotate(double theta)
```

Overrides:
rotate in class java.awt.Graphics2D

---

rotate

```
public void rotate(double theta,
                   double x,
                   double y)
```

Overrides:
rotate in class java.awt.Graphics2D

---

scale

```
public void scale(double sx,
                  double sy)
```

Overrides:
scale in class java.awt.Graphics2D

---

shear

```
public void shear(double shx,
                  double shy)
```

Overrides:
shear in class java.awt.Graphics2D

---

transform

```
public void transform(java.awt.geom.AffineTransform Tx)
```

Overrides:
transform in class java.awt.Graphics2D

---

setTransform

```
public void setTransform(java.awt.geom.AffineTransform Tx)
```

Overrides:
setTransform in class java.awt.Graphics2D

---

getTransform

```
public java.awt.geom.AffineTransform getTransform()
```

Overrides:
getTransform in class java.awt.Graphics2D

---

getPaint

`public java.awt.Paint **getPaint**()`

Overrides:

getPaint in class java.awt.Graphics2D

---

getComposite

`public java.awt.Composite **getComposite**()`

Overrides:

getComposite in class java.awt.Graphics2D

---

setBackground

`public void **setBackground**(java.awt.Color color)`

Overrides:

setBackground in class java.awt.Graphics2D

---

getBackground

`public java.awt.Color **getBackground**()`

Overrides:

getBackground in class java.awt.Graphics2D

---

getStroke

`public java.awt.Stroke **getStroke**()`

Overrides:

getStroke in class java.awt.Graphics2D

---

clip

`public void **clip**(java.awt.Shape s)`

Overrides:

clip in class java.awt.Graphics2D

---

getSources

`public java.util.Vector **getSources**()`

Specified by:

getSources in interface java.awt.image.renderable.RenderableImage

---

getProperty

`public java.lang.Object **getProperty**(java.lang.String name)`

Specified by:

getProperty in interface java.awt.image.renderable.RenderableImage

---

getPropertyNames

`public java.lang.String[] **getPropertyNames**()`

Specified by:

getPropertyNames in interface java.awt.image.renderable.RenderableImage

---

isDynamic

`public boolean **isDynamic**()`

Specified by:

isDynamic in interface java.awt.image.renderable.RenderableImage

getWidth

```
public float getWidth()
```

Specified by:
getWidth in interface java.awt.image.renderable.RenderableImage

getHeight

```
public float getHeight()
```

Specified by:
getHeight in interface java.awt.image.renderable.RenderableImage

getMinX

```
public float getMinX()
```

Specified by:
getMinX in interface java.awt.image.renderable.RenderableImage

getMinY

```
public float getMinY()
```

Specified by:
getMinY in interface java.awt.image.renderable.RenderableImage

createScaledRendering

```
public java.awt.image.RenderedImage createScaledRendering(int w,
                                                           int h,
                                                           java.awt.RenderingHints hints)
```

Specified by:
createScaledRendering in interface java.awt.image.renderable.RenderableImage

createDefaultRendering

```
public java.awt.image.RenderedImage createDefaultRendering()
```

Specified by:
createDefaultRendering in interface java.awt.image.renderable.RenderableImage

createRendering

```
public java.awt.image.RenderedImage createRendering(java.awt.image.renderable.RenderContext renderContext)
```

Creates a RenderedImage that represents a rendering of this image using a given RenderContext. This is the most general way to obtain a rendering of a RenderableImage.

The created RenderedImage may have a property identified by the String HINTS_OBSERVED to indicate which RenderingHints (from the RenderContext) were used to create the image. In addition any RenderedImages that are obtained via the getSources() method on the created RenderedImage may have such a property.

The bounds of the `RenderedImage` are determined from the dimensions parameter passed to the `RenderableGraphics` constructor. These bounds will be transformed by any `AffineTransform` from the `RenderContext`. The `RenderingHints` from the `RenderContext` may be used to specify the tile width and height, `SampleModel`, and `ColorModel` by supplying an `ImageLayout` hint. The precedence for determining tile width and height is to use firstly values provided explicitely via the `ImageLayout`, secondly the width and height of the `SampleModel` in the hint, and thirdly the bounds of the `RenderableGraphics` object after transformation.

If either the `SampleModel` or `ColorModel` is null, an attempt will be made to derive a compatible value for the null object from the non-null object. If they are both null, a 3-band byte `TiledImage` with a null `ColorModel` and a `PixelInterleavedSampleModel` will be created.

Specified by:
createRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`renderContext` - the RenderContext to use to produce the rendering.

Returns:
a RenderedImage containing the rendered data.

javax.media.jai
Class RenderableImageAdapter

```
java.lang.Object
  |
  +--javax.media.jai.RenderableImageAdapter
```

---

public final class **RenderableImageAdapter**
extends java.lang.Object
implements java.awt.image.renderable.RenderableImage, PropertySource

An adapter class for externally-generated RenderableImages. All methods are simply forwarded to the image being adapted. The purpose of this class is simply to ensure that the PropertySource interface is available for all JAI images.

Field Detail

im

`private java.awt.image.renderable.RenderableImage im`

A reference to the external RenderableImage.

Constructor Detail

RenderableImageAdapter

`public RenderableImageAdapter(java.awt.image.renderable.RenderableImage im)`

Constructs a RenderableImageAdapter from a RenderableImage.
Throws:
NullPointerException - if `im` is `null`.

Method Detail

wrapRenderableImage

`public static RenderableImageAdapter wrapRenderableImage(java.awt.image.renderable.RenderableImage im)`

Adapts a RenderableImage into a RenderableImageAdapter. If the image is already an instance of RenderableImageAdapter, it is returned unchanged.
Parameters:
`im` - a RenderableImage.
Returns:
a RenderableImageAdapter.
Throws:
NullPointerException - if `im` is `null`.

---

getSources

`public final java.util.Vector getSources()`

Specified by:
getSources in interface java.awt.image.renderable.RenderableImage

---

getProperty

`public final java.lang.Object getProperty(java.lang.String name)`

Gets a property from the property set of this image. If the property name is not recognized, java.awt.Image.UndefinedProperty will be returned.
Specified by:
getProperty in interface java.awt.image.renderable.RenderableImage
Parameters:
`name` - the name of the property to get, as a String.
Returns:
a reference to the property Object, or the value java.awt.Image.UndefinedProperty.

Throws:

java.lang.IllegalArgumentException - if `name` is `null`.

---

getPropertyNames

`public final java.lang.String[] **getPropertyNames**()`

Returns a list of the properties recognized by this image. If no properties are available, `null` will be returned.

Specified by:

getPropertyNames in interface java.awt.image.renderable.RenderableImage

Returns:

an array of `String`s representing valid property names.

---

getPropertyNames

`public java.lang.String[] **getPropertyNames**(java.lang.String prefix)`

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names match, `null` will be returned. The comparison is done in a case-independent manner.

Specified by:

getPropertyNames in interface PropertySource

Returns:

an array of `String`s giving the valid property names.

Throws:

NullPointerException - if `prefix` is `null`.

---

getWidth

`public final float **getWidth**()`

Gets the width in user coordinate space. By convention, the usual width of a RenderableImage is equal to the image's aspect ratio (width divided by height).

Specified by:

getWidth in interface java.awt.image.renderable.RenderableImage

Returns:

the width of the image in user coordinates.

---

getHeight

`public final float **getHeight**()`

Gets the height in user coordinate space. By convention, the usual height of a RenderedImage is equal to 1.0F.

Specified by:

getHeight in interface java.awt.image.renderable.RenderableImage

Returns:

the height of the image in user coordinates.

---

getMinX

`public final float **getMinX**()`

Gets the minimum X coordinate of the rendering-independent image.

Specified by:

getMinX in interface java.awt.image.renderable.RenderableImage

---

getMinY

`public final float **getMinY**()`

Gets the minimum Y coordinate of the rendering-independent image.

Specified by:

getMinY in interface java.awt.image.renderable.RenderableImage

---

isDynamic

```
public final boolean isDynamic()
```

Returns true if successive renderings (that is, calls to createRendering() or createScaledRendering()) with the same arguments may produce different results. This method may be used to determine whether an existing rendering may be cached and reused.

Specified by:
isDynamic in interface java.awt.image.renderable.RenderableImage

---

createScaledRendering

```
public final java.awt.image.RenderedImage createScaledRendering(int w,
                                                                int h,
                                                                java.awt.RenderingHints hints)
```

Gets a RenderedImage instance of this image with width w, and height h in pixels. The RenderContext is built automatically with an appropriate usr2dev transform and an area of interest of the full image. All the rendering hints come from hints passed in.

Specified by:
createScaledRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`w` - the width of rendered image in pixels.
`h` - the height of rendered image in pixels.
`hints` - a RenderingHints object containing rendering hints.

Returns:
a RenderedImage containing the rendered data.

---

createDefaultRendering

```
public final java.awt.image.RenderedImage createDefaultRendering()
```

Gets a RenderedImage instance of this image with a default width and height in pixels. The RenderContext is built automatically with an appropriate usr2dev transform and an area of interest of the full image. All the rendering hints come from hints passed in. Implementors of this interface must be sure that there is a defined default width and height.

Specified by:
createDefaultRendering in interface java.awt.image.renderable.RenderableImage

Returns:
a RenderedImage containing the rendered data.

---

createRendering

```
public final java.awt.image.RenderedImage createRendering(java.awt.image.renderable.RenderContext renderContext)
```

Gets a RenderedImage instance of this image from a RenderContext. This is the most general way to obtain a rendering of a RenderableImage.

Specified by:
createRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`renderContext` - the RenderContext to use to produce the rendering.

Returns:
a RenderedImage containing the rendered data.

javax.media.jai

Class RenderableOp

```
java.lang.Object
  |
  +--javax.media.jai.RenderableOp
```

public class **RenderableOp**
extends java.lang.Object
implements PropertySource, java.awt.image.renderable.RenderableImage, java.io.Serializable

A JAI version of RenderableImageOp. Instead of taking a ContextualRenderedImageFactory directly, we make use of the operation registry.

See Also:
`OperationRegistry`, `ContextualRenderedImageFactory`, `RenderableImageOp`

Field Detail

theRegistry

`private transient OperationRegistry` **`theRegistry`**

The `OperationRegistry` that is used to render this node.

operationName

`private java.lang.String` **`operationName`**

The name of the operation this node represents.

paramBlock

`private transient java.awt.image.renderable.ParameterBlock` **`paramBlock`**

The input arguments for this operation, including sources and/or parameters.

thePropertySource

`protected transient PropertySource` **`thePropertySource`**

boundingBox

`protected transient java.awt.geom.Rectangle2D` **`boundingBox`**

crif

`protected transient java.awt.image.renderable.ContextualRenderedImageFactory` **`crif`**

localProperties

`private transient java.util.Hashtable` **`localProperties`**

Locally-stored properties.

volatilePropertyInfo

`private java.util.Vector` **`volatilePropertyInfo`**

Cache of information in "thePropertySource" which is lost in the serialization/deserialization process. This includes the PropertyGenerators added via addPropertyGenerator() and the names of properties specified via suppressProperty(). The nature of each Vector element is determined by its class, i.e., String (name of a suppressed property) or PropertyGenerator.

Constructor Detail

RenderableOp

```
public RenderableOp(OperationRegistry registry,
                    java.lang.String opName,
                    java.awt.image.renderable.ParameterBlock pb)
```

Constructs a RenderableOp given the name of the operation to be performed and a ParameterBlock containing RenderableImage sources and other parameters. Any RenderedImage sources referenced by the ParameterBlock will be ignored.

Parameters:

`registry` - The `OperationRegistry` to be used for instantiation. if `null`, the default registry is used.

`opName` - The operation name.

`pb` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.

Throws:

NullPointerException - if `opName` is `null`.

RenderableOp

```
public RenderableOp(java.lang.String opName,
                    java.awt.image.renderable.ParameterBlock pb)
```

Constructs a RenderableOp given the name of the operation to be performed and a ParameterBlock containing RenderableImage sources and other parameters. Any RenderedImage sources referenced by the ParameterBlock will be ignored.

Parameters:

`opName` - The operation name.

`pb` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.

Throws:

NullPointerException - if `opName` is `null`.

Method Detail

getRegistry

```
public OperationRegistry getRegistry()
```

Returns the `OperationRegistry` that is used by this node. If the registry had not been set, the default registry is returned.

---

setRegistry

```
public void setRegistry(OperationRegistry registry)
```

Sets the `OperationRegistry` that is used by this node. If the specified registry is `null`, the default registry is used.

---

getOperationName

```
public java.lang.String getOperationName()
```

Returns the name of the operation this node represents as a `String`.

---

setOperationName

```
public void setOperationName(java.lang.String opName)
```

Sets the name of the operation this node represents. The parameter is saved by reference.

Parameters:

`opName` - The new operation name to be set.

Throws:

NullPointerException - if `opName` is `null`.

---

getParameterBlock

```
public java.awt.image.renderable.ParameterBlock getParameterBlock()
```

Returns the `ParameterBlock` of this node.

---

setParameterBlock

`public void` **`setParameterBlock`**`(java.awt.image.renderable.ParameterBlock pb)`

Sets the `ParameterBlock` of this node. If the speicifed new `ParameterBlock` is `null`, it is assumed that this node has no input sources and parameters. The parameter is saved by reference.

This method does not validate the content of the supplied `ParameterBlock`. The caller should ensure that the sources and parameters in the `ParameterBlock` are suitable for the operation this node represents; otherwise some form of error or exception may occur at the time of rendering.

Parameters:

`pb` - The new `ParameterBlock` to be set; it may be `null`.

---

getSources

`public java.util.Vector` **`getSources`**`()`

Returns a vector of RenderableImages that are the sources of image data for this RenderableImage. Note that this method may return an empty vector, to indicate that the image has sources but none of them is a RenderableImage, or null to indicate the image has no source of any type.

Specified by:

getSources in interface java.awt.image.renderable.RenderableImage

Returns:

a (possibly empty) Vector of RenderableImages, or null.

---

getRenderableSources

`private java.util.Vector` **`getRenderableSources`**`()`

---

createPropertySource

`private void` **`createPropertySource`**`()`

Creates a `PropertySource` if none exists.

---

createVolatilePropertyVector

`private void` **`createVolatilePropertyVector`**`()`

Creates a volatile property info Vector if none exists.

---

createLocalProperties

`private void` **`createLocalProperties`**`()`

Initialize the localProperties Hashtable if needed.

---

getPropertyNames

`public java.lang.String[]` **`getPropertyNames`**`()`

Returns the names of properties available from this node. These properties are a combination of those derived from prior nodes in the imaging chain, those set locally, and those generated by the rendering.

Specified by:

getPropertyNames in interface PropertySource

Returns:

An array of `String`s containing valid property names.

---

getPropertyNames

`public java.lang.String[]` **`getPropertyNames`**`(java.lang.String prefix)`

Returns an array of `String`s recognized as names by this property source that begin with the supplied prefix. If no property names match, `null` will be returned. The comparison is done in a case-independent manner.

Specified by:

getPropertyNames in interface PropertySource

Returns:

an array of `String`s giving the valid property names.

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Gets a property from the property set of this image. If the property name is not recognized, java.awt.Image.UndefinedProperty will be returned.

Specified by:

getProperty in interface PropertySource

Parameters:

`name` - the name of the property to get, as a String.

Returns:

a reference to the property Object, or the value java.awt.Image.UndefinedProperty.

setProperty

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
```

Sets a local property on a node. The synthetic properties (containg image width, height, and position) may not be set. Local property settings override properties derived from prior nodes in the imaging chain.

If the node is serialized then serializable properties will also be serialized but non-serializable properties will be lost.

Parameters:

`name` - a String representing the property name.
`value` - the property's value, as an Object.

isDynamic

```
public boolean isDynamic()
```

Returns false, i.e., successive renderings with the same arguments will produce identical results.

Specified by:

isDynamic in interface java.awt.image.renderable.RenderableImage

addPropertyGenerator

```
public void addPropertyGenerator(PropertyGenerator pg)
```

Adds a PropertyGenerator to the node. The property values emitted by this property generator override any previous definitions.

Parameters:

`pg` - a PropertyGenerator to be added to this node's property environment.

suppressProperty

```
public void suppressProperty(java.lang.String name)
```

Removes a named property from the property environment of this node. Subsequent calls to getProperty(name) will return null, and name will not appear on the list of properties emitted by getPropertyNames().

Parameters:

`name` - a String naming the property to be suppressed.

getWidth

```
public float getWidth()
```

Return the rendering-independenmt width of the image.

Specified by:

getWidth in interface java.awt.image.renderable.RenderableImage

Returns:

the image width as a float.

getHeight

```
public float getHeight()
```

Return the rendering-independent height of the image.

Specified by:

getHeight in interface java.awt.image.renderable.RenderableImage

Returns:
the image height as a float.

---

getMinX

```
public float getMinX()
```

Gets the minimum X coordinate of the rendering-independent image data.

Specified by:
getMinX in interface java.awt.image.renderable.RenderableImage

---

getMinY

```
public float getMinY()
```

Gets the minimum Y coordinate of the rendering-independent image data.

Specified by:
getMinY in interface java.awt.image.renderable.RenderableImage

---

createDefaultRendering

```
public java.awt.image.RenderedImage createDefaultRendering()
```

Returns a RenderedImage instance of this image equivalent to what would be obtained by invoking createRendering() with the identity transform, an area of interest equal to the image bounds, and no rendering hints.

This method does not validate sources and parameters supplied in the `ParameterBlock` against the specification of the operation this node represents. It is the caller's responsibility to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, some kind of exception or error will occur.

Specified by:
createDefaultRendering in interface java.awt.image.renderable.RenderableImage

Returns:
The default RenderedImage.

---

createScaledRendering

```
public java.awt.image.RenderedImage createScaledRendering(int w,
                                                          int h,
                                                          java.awt.RenderingHints hints)
```

Gets a RenderedImage instance of this image with width w, and height h in pixels. The RenderContext is built automatically with an appropriate usr2dev transform and an area of interest of the full image. All the rendering hints come from hints passed in.

If w == 0, it will be taken to equal Math.round(h*(getWidth()/getHeight())). Similarly, if h == 0, it will be taken to equal Math.round(w*(getHeight()/getWidth())). One of w or h must be non-zero or else an IllegalArgumentException will be thrown.

This method does not validate sources and parameters supplied in the `ParameterBlock` against the specification of the operation this node represents. It is the caller's responsibility to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, some kind of exception or error will occur.

Specified by:
createScaledRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`w` - the width of rendered image in pixels, or 0.
`h` - the height of rendered image in pixels, or 0.
`hints` - a RenderingHints object containg hints.

Returns:
a RenderedImage containing the rendered data.

Throws:
java.lang.IllegalArgumentException - if both w and h are zero.

---

createRendering

```
public java.awt.image.RenderedImage createRendering(java.awt.image.renderable.RenderContext renderContext)
```

Gets a RenderedImage that represented a rendering of this image using a given RenderContext. This is the most general way to obtain a rendering of a RenderableImage.

This method does not validate sources and parameters supplied in the `ParameterBlock` against the specification of the operation this node represents. It is the caller's responsibility to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, some kind of exception or error will occur.

`JAI.createRenderable()` is the method that does the validation. Therefore, it is strongly recommended that all `RenderableOp`s are created using `JAI.createRenderable()`.

The `RenderContext` may contain a `Shape` that represents the area-of-interest (aoi). If the aoi is specifed, it is still legal to return an image that's larger than this aoi. Therefore, by default, the aoi, if specified, is ignored at the rendering.

Specified by:
createRendering in interface java.awt.image.renderable.RenderableImage

Parameters:
`renderContext` - the RenderContext to use to produce the rendering.

Returns:
a RenderedImage containing the rendered data.

---

getRenderedImage

```
private java.awt.image.RenderedImage getRenderedImage(java.awt.image.renderable.RenderContext renderContext)
```

---

findCRIF

```
private java.awt.image.renderable.ContextualRenderedImageFactory findCRIF()
```

Use registry to find an appropriate CRIF

---

getSource

```
public java.lang.Object getSource(int index)
```

Returns one of the node's sources as an Object.

Parameters:
`index` - the index of the source.

---

setSource

```
public void setSource(java.lang.Object source,
                      int index)
```

Sets one of the node's sources to an Object.

Parameters:
`source` - the source, as an Object.
`index` - the index of the source.

---

getByteParameter

```
public byte getByteParameter(int index)
```

Returns one of the node's parameters, as a byte.

Parameters:
`index` - the index of the parameter.

---

getCharParameter

```
public char getCharParameter(int index)
```

Returns one of the node's parameters, as a char.

Parameters:
`index` - the index of the parameter.

---

getShortParameter

```
public short getShortParameter(int index)
```

Returns one of the node's parameters, as a short.

Parameters:
`index` - the index of the parameter.

---

getIntParameter

```
public int getIntParameter(int index)
```

Returns one of the node’s parameters, as an int.
Parameters:
`index` - the index of the parameter.

---

getLongParameter

```
public long getLongParameter(int index)
```

Returns one of the node’s parameters, as a long.
Parameters:
`index` - the index of the parameter.

---

getFloatParameter

```
public float getFloatParameter(int index)
```

Returns one of the node’s parameters, as a float.
Parameters:
`index` - the index of the parameter.

---

getDoubleParameter

```
public double getDoubleParameter(int index)
```

Returns one of the node’s parameters, as a double.
Parameters:
`index` - the index of the parameter.

---

getObjectParameter

```
public java.lang.Object getObjectParameter(int index)
```

Returns one of the node’s parameters, as an Object.
Parameters:
`index` - the index of the parameter.

---

setParameter

```
public void setParameter(byte param,
                         int index)
```

Sets one of the node’s parameters to a byte.
Parameters:
`param` - the parameter, as a byte.
`index` - the index of the parameter.

---

setParameter

```
public void setParameter(char param,
                         int index)
```

Sets one of the node’s parameters to a char.
Parameters:
`param` - the parameter, as a char.
`index` - the index of the parameter.

---

setParameter

```
public void setParameter(short param,
                         int index)
```

Sets one of the node’s parameters to a short.
Parameters:
`param` - the parameter, as a short.
`index` - the index of the parameter.

setParameter

```
public void setParameter(int param,
                         int index)
```

Sets one of the node's parameters to an int.

Parameters:

`param` - the parameter, as an int.
`index` - the index of the parameter.

setParameter

```
public void setParameter(long param,
                         int index)
```

Sets one of the node's parameters to a long.

Parameters:

`param` - the parameter, as a long.
`index` - the index of the parameter.

setParameter

```
public void setParameter(float param,
                         int index)
```

Sets one of the node's parameters to a float.

Parameters:

`param` - the parameter, as a float.
`index` - the index of the parameter.

setParameter

```
public void setParameter(double param,
                         int index)
```

Sets one of the node's parameters to a double.

Parameters:

`param` - the parameter, as a double.
`index` - the index of the parameter.

setParameter

```
public void setParameter(java.lang.Object param,
                         int index)
```

Sets one of the node's parameters to an Object.

Parameters:

`param` - the parameter, as an Object.
`index` - the index of the parameter.

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serialize the RenderableOp.

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the RenderableOp.

javax.media.jai
Class RenderedImageAdapter

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.RenderedImageAdapter
```

Direct Known Subclasses:
WritableRenderedImageAdapter

---

public class **RenderedImageAdapter**
extends PlanarImage

A PlanarImage wrapper for a non-writable RenderedImage. The tile layout, sample model, and so forth are preserved. Calls to getTile() and so forth are forwarded.

From JAI's point of view, this image is a PlanarImage of unknown type, with no sources. The source image is assumed to be immutable. If the RenderedImage source implements WritableRenderedImage, a WritableRenderedImageAdapter should be used.

The class and all its methods are marked 'final' in order to allow dynamic inlining to take place. This should eliminate any performance penalty associated with the use of an adapter class.

Since the methods of this class all derive from PlanarImage, they are not commented in detail.

See Also:
`PlanarImage`, `RenderedImage`, `WritableRenderedImage`, `WritableRenderedImageAdapter`

Field Detail

theImage

`protected java.awt.image.RenderedImage` **`theImage`**

The RenderedImage being adapted.

Constructor Detail

RenderedImageAdapter

`public` **`RenderedImageAdapter`**`(java.awt.image.RenderedImage im)`

Constructs a RenderedImageAdapter.
Parameters:
`im` - a RenderedImage to be 'wrapped' as a PlanarImage.
Throws:
java.lang.IllegalArgumentException - if `im` is `null`.

Method Detail

getProperty

`public final java.lang.Object` **`getProperty`**`(java.lang.String name)`

Forwards call to the true source.
Overrides:
getProperty in class PlanarImage

---

getPropertyNames

`public final java.lang.String[]` **`getPropertyNames`**`()`

Forwards call to the true source.
Overrides:
getPropertyNames in class PlanarImage

---

getTile

```
public final java.awt.image.Raster getTile(int x,
                                           int y)
```

Forwards call to the true source.

Overrides:

getTile in class PlanarImage

---

getData

```
public final java.awt.image.Raster getData()
```

Forwards call to the true source.

Overrides:

getData in class PlanarImage

---

getData

```
public final java.awt.image.Raster getData(java.awt.Rectangle rect)
```

Forwards call to the true source.

Overrides:

getData in class PlanarImage

---

copyData

```
public final java.awt.image.WritableRaster copyData(java.awt.image.WritableRaster raster)
```

Forwards call to the true source.

Overrides:

copyData in class PlanarImage

javax.media.jai
Class RenderedOp

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.RenderedOp
```

public class **RenderedOp**
extends PlanarImage
implements java.io.Serializable

A node in a rendered imaging chain. A `RenderedOp` stores an operation name (as a `String`), a `ParameterBlock` containing sources and miscellaneous parameters, and a `RenderingHints` containing rendering hints. A set of nodes may be joined together via the source `Vector`s within their `ParameterBlock`s to form a directed acyclic graph (DAG). The topology i.e., connectivity of the graph may be altered by changing the `ParameterBlock`s; the operation name, parameters, and rendering hints may also be changed.

Such chains are useful as arguments to a `RemoteImage`; they convey the structure of an imaging chain in a compact representation and at a suitably high level of abstraction to allow the server some leeway in materializing the results.

When any `RenderedImage` method is called on a `RenderedOp`, (any of `getWidth()`, `getHeight()`, `getMinX()`, `getMinY()`, `getNumXTiles()`, `getNumYTiles()`, `getMinTileX()`, `getMinTileY()`, `getTileWidth()`, `getTileHeight()`, `getTileGridXOffset()`, `getTileGridYOffset()`,`getTile()`, `getData()`, `copyData()`, `getColorModel()`, `getSampleModel()`, or `getProperty()` with a synthesized property name), the `RenderedOp` is implicitly rendered and becomes "frozen." Its operation name, `ParameterBlock`, and rendering hints may no longer be changed.

Furthermore, when some of the methods from `PlanarImage` are called, such as "get" and "set" methods on its source objects, the `RenderedOp` is implictly rendered and becomes "frozen."

Serialization of a "frozen" node has the effect of "thawing" it; in other words, the instance variable holding the reference to the rendering of the node is transient and is not placed into the serialized byte stream. This allows working chains to be passed to a remote server using remote method invocation (RMI). Note that `RenderedOp` nodes used to instantiate operations which have a corresponding `OperationDescriptor` the `isImmediate()` method of which returns `true` are rendered upon deserialization.

A node may be rendered explicitly by means of the `createInstance()` method. This method returns a `PlanarImage` rendering without freezing the node. This allows a chain to be manipulated dynamically and rendered multiple times.

The translation between `RenderedOp` chains and `OpImage` chains makes use of two levels of indirection provided by the `OperationRegistry` and `RenderedImageFactory` (RIF) facilities. First, the local `OperationRegistry` is used to map the operation name into a RIF. This RIF then constructs one or more `OpImage`s to do the actual work (or returns a `RenderedImage` by other means). The local `OperationRegistry` is used in order to take advantage of RIFs that are known to a server without having to burden the client.

`RenderedOp` represents a single `PlanarImage`; its companion class, `CollectionOp` represents `CollectionImage` nodes.

The `RenderedOp` synthesizes several poperty values, which may not be removed. These are: `image_width`, `image_height`, `image_min_x_coord`, and `image_min_y_coord`.

See Also:
`CollectionOp`

Field Detail

theImage

`protected transient PlanarImage **theImage**`

The rendering of the current image, not preserved over RMI.

theRegistry

`private transient OperationRegistry **theRegistry**`

The `OperationRegistry` that is used to render this node.

operationName

`private java.lang.String` **`operationName`**

The name of the operation this node represents.

---

paramBlock

`private transient java.awt.image.renderable.ParameterBlock` **`paramBlock`**

The input arguments for this operation, including sources and/or parameters.

---

renderHints

`private transient java.awt.RenderingHints` **`renderHints`**

The rendering hints to use for this operation.

---

thePropertySource

`protected transient PropertySource` **`thePropertySource`**

The `PropertySource` containing the combined properties of all of the node's sources.

---

synthProps

`private static java.util.Vector` **`synthProps`**

Names of synthesized properties.

---

localProperties

`private transient java.util.Hashtable` **`localProperties`**

Locally-stored properties.

---

synthProperties

`private java.util.Hashtable` **`synthProperties`**

Synthesized properties.

---

volatilePropertyInfo

`private java.util.Vector` **`volatilePropertyInfo`**

Cache of information in "thePropertySource" which is lost in the serialization/deserialization process. This includes the PropertyGenerators added via addPropertyGenerator() and the names of properties specified via suppressProperty(). The nature of each Vector element is determined by its class, i.e., String (name of a suppressed property) or PropertyGenerator.

Constructor Detail

RenderedOp

```
public RenderedOp(OperationRegistry registry,
                  java.lang.String opName,
                  java.awt.image.renderable.ParameterBlock pb,
                  java.awt.RenderingHints hints)
```

Constructs a `RenderedOp` that will be used to instantiate a particular rendered operation from a given operation registry, an operation name, a `ParameterBlock`, and a set of rendering hints. All input parameters are saved by reference.

Parameters:

`registry` - The `OperationRegistry` to be used for instantiation. if `null`, the default registry is used.
`opName` - The operation name.
`pb` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.
`hints` - The rendering hints. If `null`, it is assumed that no hints are associated with the rendering.

Throws:

NullPointerException - if `opName` is `null`.

RenderedOp

```
public RenderedOp(java.lang.String opName,
                  java.awt.image.renderable.ParameterBlock pb,
                  java.awt.RenderingHints hints)
```

Constructs a `RenderedOp` that will be used to instantiate a particular rendered operation from a given operation registry, an operation name, a `ParameterBlock`, and a set of rendering hints. The default operation registry is used. All input parameters are saved by reference.

Parameters:

`opName` - The operation name.
`pb` - The sources and other parameters. If `null`, it is assumed that this node has no sources and parameters.
`hints` - The rendering hints. If `null`, it is assumed that no hints are associated with the rendering.

Throws:

NullPointerException - if `opName` is `null`.

Method Detail

```
static void ()
```

---

getRegistry

```
public OperationRegistry getRegistry()
```

Returns the `OperationRegistry` that is used by this node. If the registry had not been set, the default registry is returned.

---

setRegistry

```
public void setRegistry(OperationRegistry registry)
```

Sets the `OperationRegistry` that is used by this node. If the specified registry is `null`, the default registry is used. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:

`registry` - The new `OperationRegistry` to be set; it may be `null`.

---

getOperationName

```
public java.lang.String getOperationName()
```

Returns the name of the operation this node represents as a `String`.

---

setOperationName

```
public void setOperationName(java.lang.String opName)
```

Sets the name of the operation this node represents. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:

`opName` - The new operation name to be set.

Throws:

NullPointerException - if `opName` is `null`.

---

getParameterBlock

```
public java.awt.image.renderable.ParameterBlock getParameterBlock()
```

Returns the `ParameterBlock` of this node.

---

setParameterBlock

```
public void setParameterBlock(java.awt.image.renderable.ParameterBlock pb)
```

Sets the `ParameterBlock` of this node. If this node has been rendered and frozen, this method has no effect. If the speicifed new `ParameterBlock` is `null`, it is assumed that this node has no input sources and parameters. The parameter is saved by reference.

This method does not validate the content of the supplied `ParameterBlock`. The caller should ensure that the sources and parameters in the `ParameterBlock` are suitable for the operation this node represents; otherwise some form of error or exception may occur at the time of rendering.

Parameters:

`pb` - The new `ParameterBlock` to be set; it may be `null`.

---

getRenderingHints

`public java.awt.RenderingHints` **`getRenderingHints`**`()`

Returns the `RenderingHints` of this node. It may be `null`.

---

setRenderingHints

`public void` **`setRenderingHints`**`(java.awt.RenderingHints hints)`

Sets the `RenderingHints` of this node. If this node has been rendered and frozen, this method has no effect. The parameter is saved by reference.

Parameters:

`hints` - The new `RenderingHints` to be set; it may be `null`.

---

createInstance

`public PlanarImage` **`createInstance`**`()`

Instantiate a `PlanarImage` that computes the result of this `RenderedOp`. The default `OperationRegistry` is used to translate operation names into actual `OpImages`.

During this method, all the sources supplied in the `ParameterBlock` are checked. If any of the sources is a `RenderedOp`, a rendering of that source is created. This propagates all the way up to the top of the op chain. If any of the sources is a `Collection`, then the collection is passed to the operation as-is. If there is a `RenderedOp` anywhere in the collection, it is up to the individual operation to create the rendering for that `RenderedOp`.

This method does not validate the sources and parameters stored in the `ParameterBlock` against the specification of the operation this node represents. It is the responsibility of the caller to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, some kind of exception or error will occur.

Returns:

The resulting image as a `PlanarImage`.

---

createInstance

`private PlanarImage` **`createInstance`**`(boolean isChainFrozen)`

This method performs the actions described by the documentation of createInstance() optionally freezing the image chain as a function of the parameter.

---

createRendering

`private void` **`createRendering`**`()`

Creates an `RenderedImage` rendering if none exists.

---

getRendering

`public PlanarImage` **`getRendering`**`()`

Returns the `PlanarImage` rendering associated with this `RenderedOp` node.

This method does not validate the sources and parameters stored in the `ParameterBlock` against the specification of the operation this node represents. It is the caller's responsibility to ensure that the data in the `ParameterBlock` are suitable for this operation. Otherwise, an exception or error will occur.

---

createPropertySource

`private void` **`createPropertySource`**`()`

Creates a `PropertySource` if none exists.

---

createVolatilePropertyVector

`private void` **`createVolatilePropertyVector`**`()`

Creates a volatile property info Vector if none exists.

---

getPropertyNames

`public java.lang.String[]` **`getPropertyNames`**`()`

Returns the names of properties available from this node. These properties are a combination of those derived from prior nodes in the imaging chain, those set locally, and a number of locally derived, immutable properties based on the rendering associated with this node -- height, width, and so forth.

Returns:
An array of `String`s containing valid property names.
Overrides:
getPropertyNames in class PlanarImage

---

createSynthProperties

`private void` **`createSynthProperties`**`()`

Initialize the synthProperties Hashtable if needed.

---

createLocalProperties

`private void` **`createLocalProperties`**`()`

Initialize the localProperties Hashtable if needed.

---

getProperty

`public java.lang.Object` **`getProperty`**`(java.lang.String name)`

Returns the property associated with the specified property name, or `java.awt.Image.UndefinedProperty` if the specified property is not set on the image.

Parameters:
`name` - A `String` naming the property.
Throws:
java.lang.IllegalArgumentException - if `name` is `null`.
Overrides:
getProperty in class PlanarImage

---

setProperty

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
```

Sets a local property on a node. The synthetic properties (containing image width, height, and position) may not be set. Local property settings override properties derived from prior nodes in the imaging chain.

Properties may be set on a `RenderedOp` node only prior to the creation of a rendering at that node. In general this means that `setProperty()` should be called only immediately after construction of a node.

If the node is serialized then serializable properties will also be serialized but non-serializable properties will be lost.

Parameters:
`name` - A `String` representing the property name.
`value` - The property's value, as an `Object`.
Throws:
`RuntimeException` - if method is called on a rendered node.
`RuntimeException` - if `name` conflicts with Synthetic property.
`IllegalArgumentException` - if `name` is `null`.
`IllegalArgumentException` - if `value` is `null`.
Overrides:
setProperty in class PlanarImage

---

addPropertyGenerator

```
public void addPropertyGenerator(PropertyGenerator pg)
```

Adds a `PropertyGenerator` to the node. The property values emitted by this property generator override any previous definitions.

Parameters:
`pg` - A `PropertyGenerator` to be added to this node's property environment.

Throws:
`IllegalArgumentException` - if `pg` is `null`.

---

suppressProperty

```
public void suppressProperty(java.lang.String name)
```

Removes a named property from the property environment of this node. Subsequent calls to `getProperty(name)` will return null, and name will not appear on the list of properties emitted by `getPropertyNames()`.

Parameters:
`name` - A `String` naming the property to be suppressed.

Throws:
`IllegalArgumentException` - if `name` is `null`.
`RuntimeException` - if `name` conflicts with Synthetic property.

---

writeObject

```
private void writeObject(java.io.ObjectOutputStream out)
                  throws java.io.IOException
```

Serializes the `RenderedOp`.

---

readObject

```
private void readObject(java.io.ObjectInputStream in)
                 throws java.io.IOException,
                        java.lang.ClassNotFoundException
```

Deserialize the `RenderedOp`.

---

getNumSources

```
public int getNumSources()
```

Returns the number of sources stored in the `ParameterBlock` of this node. This may differ from the number of sources of the rendered image.

Overrides:
getNumSources in class PlanarImage

---

getSources

```
public java.util.Vector getSources()
```

Returns the sources stored in the `ParameterBlock` of this node. This may differ from the source vector of the rendered image.

Overrides:
getSources in class PlanarImage

---

getNodeSource

```
public java.lang.Object getNodeSource(int index)
```

---

addNodeSource

```
public void addNodeSource(java.lang.Object source)
```

Adds a source to the `ParameterBlock` of this node. If this node has been rendered, this method has no effect.

Parameters:
`source` - The source to be added to the `ParameterBlock`

setSources

```
public void setSources(java.util.List sourceList)
```

Replaces the sources in the `ParameterBlock` of this node with a new list of sources. If this node has been rendered, this method has no effect.

Parameters:

`sourceList` - A `List` of sources.

Throws:

java.lang.IllegalArgumentException - if `sourceList` is `null`.

Overrides:

setSources in class PlanarImage

setNodeSource

```
public void setNodeSource(java.lang.Object source,
                          int index)
```

Sets the specified source stored in the `ParameterBlock` of this node to a new source object. If this node has been rendered, this method has no effect.

Parameters:

`source` - The Source to be set.
`index` - The Index at which it is to be set.

Throws:

java.lang.IllegalArgumentException - if `source` is `null`.
ArrayIndexOutOfBoundsException - if `index` is invalid.

removeSources

```
public void removeSources()
```

Removes all the sources stored in the `ParameterBlock` of this node. If this node has been rendered, this method has no effect.

Overrides:

removeSources in class PlanarImage

getNumParameters

```
public int getNumParameters()
```

Returns the number of parameters stored in the `ParameterBlock` of this node.

getParameters

```
public java.util.Vector getParameters()
```

Returns the parameters stored in the `ParameterBlock` of this node.

getByteParameter

```
public byte getByteParameter(int index)
```

Returns the specified parameter stored in the `ParameterBlock` of this node as a `byte`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied

Parameters:

`index` - The index of the parameter.

Throws:

`ArrayIndexOutOfBoundsException` - if `index` is invalid.

getCharParameter

```
public char getCharParameter(int index)
```

Returns the specified parameter stored in the `ParameterBlock` of this node as a `char`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied

Parameters:

`index` - The index of the parameter.

Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getShortParameter

`public short getShortParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as a `short`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getIntParameter

`public int getIntParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as an `int`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getLongParameter

`public long getLongParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as a `long`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getFloatParameter

`public float getFloatParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as a `float`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getDoubleParameter

`public double getDoubleParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as a `double`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

---

getObjectParameter

`public java.lang.Object getObjectParameter(int index)`

Returns the specified parameter stored in the `ParameterBlock` of this node as an `Object`. An `ArrayIndexOutOfBoundsException` may occur if an invalid index is supplied
Parameters:
`index` - The index of the parameter.
Throws:
`ArrayIndexOutOfBoundsException` - if `index` is invalid.

setParameter

```
public void setParameter(byte param,
                         int index)
```

Sets one of the node's parameters to a `byte`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `byte`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(char param,
                         int index)
```

Sets one of the node's parameters to a `char`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `char`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(short param,
                         int index)
```

Sets one of the node's parameters to a `short`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `short`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(int param,
                         int index)
```

Sets one of the node's parameters to an `int`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as an `int`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(long param,
                         int index)
```

Sets one of the node's parameters to a `long`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `long`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(float param,
                         int index)
```

Sets one of the node's parameters to a `float`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `float`.
`index` - The index of the parameter.

setParameter

```
public void setParameter(double param,
                         int index)
```

Sets one of the node's parameters to a `double`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as a `double`.
`index` - The index of the parameter.

---

setParameter

```
public void setParameter(java.lang.Object param,
                         int index)
```

Sets one of the node's parameters to an `Object`. If the node has been rendered, this has no effect. If the `index` lies beyond the current source list, the list is extended with nulls as needed.

Parameters:

`param` - The parameter, as an `Object`.
`index` - The index of the parameter.

---

getMinX

```
public int getMinX()
```

Renders the node if it has not already been rendered, and returns the X coordinate of the leftmost column of the rendered image.

Overrides:

getMinX in class PlanarImage

---

getMaxX

```
public int getMaxX()
```

Renders the node if it has not already been rendered, and returns the X coordinate of the column immediately to the right of the rightmost column of the rendered image.

Overrides:

getMaxX in class PlanarImage

---

getMinY

```
public int getMinY()
```

Renders the node if it has not already been rendered, and returns the X coordinate of the uppermost row of the rendered image.

Overrides:

getMinY in class PlanarImage

---

getMaxY

```
public int getMaxY()
```

Renders the node if it has not already been rendered, and returns the Y coordinate of the row immediately below the bottom row of the rendered image.

Overrides:

getMaxY in class PlanarImage

---

getWidth

```
public int getWidth()
```

Renders the node if it has not already been rendered, and returns the width of the rendered image.

Overrides:

getWidth in class PlanarImage

---

getHeight

`public int **getHeight**()`

Renders the node if it has not already been rendered, and returns the height of the rendered image.

Overrides:

getHeight in class PlanarImage

---

getTileWidth

`public int **getTileWidth**()`

Renders the node if it has not already been rendered, and returns the tile width of the rendered image.

Overrides:

getTileWidth in class PlanarImage

---

getTileHeight

`public int **getTileHeight**()`

Renders the node if it has not already been rendered, and returns the tile height of the rendered image.

Overrides:

getTileHeight in class PlanarImage

---

getTileGridXOffset

`public int **getTileGridXOffset**()`

Renders the node if it has not already been rendered, and returns the tile grid X offset of the rendered image.

Overrides:

getTileGridXOffset in class PlanarImage

---

getTileGridYOffset

`public int **getTileGridYOffset**()`

Renders the node if it has not already been rendered, and returns the tile grid Y offset of the rendered image.

Overrides:

getTileGridYOffset in class PlanarImage

---

getSampleModel

`public java.awt.image.SampleModel **getSampleModel**()`

Renders the node if it has not already been rendered, and returns the `SampleModel` of the rendered image.

Overrides:

getSampleModel in class PlanarImage

---

getColorModel

`public java.awt.image.ColorModel **getColorModel**()`

Renders the node if it has not already been rendered, and returns the `ColorModel` of the rendered image.

Overrides:

getColorModel in class PlanarImage

---

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Renders the node if it has not already been rendered, and returns the specified tile of the rendered image.

Parameters:

`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Returns:

The requested tile as a `Raster`.

Overrides:

getTile in class PlanarImage

getTiles

`public java.awt.image.Raster[] **getTiles**(java.awt.Point[] tileIndices)`

Renders the node if it has not already been rendered, and returns the tiles indicated by the `tileIndices` of the rendered image as an array of `Raster`s.

Parameters:

`tileIndices` - An array of Points representing TileIndices.

Returns:

An array of Raster containing the tiles corresponding to the given TileIndices.

Overrides:

getTiles in class PlanarImage

prefetchTiles

`public void **prefetchTiles**(java.awt.Point[] tileIndices)`

Renders the node if it has not already been rendered. Hints that the given tiles of the rendered image might be needed in the near future.

Parameters:

`tileIndices` - A list of tileIndices indicating which tiles to prefetch.

Overrides:

prefetchTiles in class PlanarImage

getData

`public java.awt.image.Raster **getData**()`

Renders the node if it has not already been rendered, and returns the entire rendered image as a `Raster`.

Overrides:

getData in class PlanarImage

getData

`public java.awt.image.Raster **getData**(java.awt.Rectangle rect)`

Renders the node if it has not already been rendered, and returns a specified rectangular region of the rendered image as a `Raster`.

Overrides:

getData in class PlanarImage

copyData

`public java.awt.image.WritableRaster **copyData**()`

Renders the node if it has not already been rendered, and copies and returns the entire rendered image into a single raster.

Overrides:

copyData in class PlanarImage

copyData

`public java.awt.image.WritableRaster **copyData**(java.awt.image.WritableRaster raster)`

Renders the node if it has not already been rendered, and copies a specified rectangle of the rendered image into the given `WritableRaster`.

Parameters:

`raster` - A `WritableRaster` to be filled with image data.

Returns:

A reference to the supplied `WritableRaster`.

Overrides:

copyData in class PlanarImage

getSource

`public PlanarImage **getSource**(int index)`

Renders the node if it has not already been rendered, and returns the specified `PlanarImage` source of the rendered image. If there is no source corresponding to the specified index, this method will throw an `ArrayIndexOutOfBoundsException`. The source returned may differ from the source stored in the `ParameterBlock` of this node.

Parameters:
`index` - The index of the desired source.
Returns:
A `PlanarImage` source.
Overrides:
getSource in class PlanarImage

---

addSource

```
public void addSource(PlanarImage source)
```

Renders the node if it has not already been rendered, and adds a `PlanarImage` source to the list of sources of the rendered image.
Parameters:
`source` - A `PlanarImage` to be added as a source.
Throws:
`IllegalArgumentException` - if `source` is `null`.
Overrides:
addSource in class PlanarImage

---

removeSource

```
public boolean removeSource(PlanarImage source)
```

Renders the node if it has not already been rendered, and removes a `PlanarImage` source from the list of sources of the rendered image.
Parameters:
`source` - A `PlanarImage` to be removed.
Returns:
`true` if the element was present, false otherwise.
Throws:
`IllegalArgumentException` - if `source` is `null`.
Overrides:
removeSource in class PlanarImage

---

setSource

```
public void setSource(PlanarImage source,
                      int index)
```

Renders the node if it has not already been rendered, and sets the specified source of the rendered image to the supplied `PlanarImage`. An `ArrayIndexOutOfBoundsException` may be thrown if an invalid `index` is supplied
Parameters:
`source` - The source, as a `PlanarImage`.
`index` - The index of the source.
Overrides:
setSource in class PlanarImage

---

getSinks

```
public java.util.Vector getSinks()
```

Renders the node if it has not already been rendered, and returns a `Vector` containing the currently available `PlanarImage` sinks of the rendered image, or `null` if no sinks are present.
Overrides:
getSinks in class PlanarImage

---

addSink

```
public void addSink(PlanarImage sink)
```

Renders the node if it has not already been rendered, and adds a `PlanarImage` sink to the list of sinks of the rendered image.
Throws:
`IllegalArgumentException` - if `sink` is `null`.
Overrides:
addSink in class PlanarImage

removeSink

`public boolean` **`removeSink`**`(PlanarImage sink)`

Renders the node if it has not already been rendered, and removes a `PlanarImage` sink from the list of sinks of the rendered image.

Throws:

`IllegalArgumentException` - if `sink` is `null`.

Overrides:

removeSink in class PlanarImage

javax.media.jai

Class ScaleOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.WarpOpImage
                    |
                    +--javax.media.jai.ScaleOpImage
```

public abstract class **ScaleOpImage**
extends WarpOpImage

A class extending `WarpOpImage` for use by further extension classes that perform image scaling. Image scaling operations require rectilinear backwards mapping and padding by the resampling filter dimensions.

When applying scale factors of scaleX, scaleY to a source image with width of src_width and height of src_height, the resulting image is defined to have the following bounds: `dst min X = floor(src min X * scaleX + transX) dst min Y = floor(src min Y * scaleY + transY) dst width = ceil(src width * scaleX) dst height = ceil(src height * scaleY)`

When interpolations which require padding the source such as Bilinear or Bicubic interpolation are specified, the source needs to be extended such that it has the extra pixels needed to compute all the destination pixels. This extension is performed via the `BorderExtender` class. The type of border extension can be specified as a `RenderingHint` to the `JAI.create` method.

If no `BorderExtender` is specified, the source will not be extended. The scaled image size is still calculated according to the formula specified above. However since there is not enough source to compute all the destination pixels, only that subset of the destination image's pixels which can be computed, will be written in the destination. The rest of the destination will be set to zeros.

See Also:
`WarpOpImage`, `OpImage`

Field Detail

scaleX

`protected float scaleX`

The horizontal scale factor.

scaleY

`protected float scaleY`

The vertical scale factor.

transX

`protected float transX`

Thee horizontal translation factor

transY

`protected float transY`

The vertical translation factor

invScaleX

`protected float invScaleX`

Cached value equal to 1/scaleX.

invScaleY
`protected float` **`invScaleY`**

Cached value equal to 1/scaleY.

---

extender
`protected BorderExtender` **`extender`**

The `BorderExtender`, or `null`.

---

scaleXRational
`protected com.sun.media.jai.util.Rational` **`scaleXRational`**

Rational representations

---

scaleYRational
`protected com.sun.media.jai.util.Rational` **`scaleYRational`**

Rational representations

---

scaleXRationalNum
`protected long` **`scaleXRationalNum`**

---

scaleXRationalDenom
`protected long` **`scaleXRationalDenom`**

---

scaleYRationalNum
`protected long` **`scaleYRationalNum`**

---

scaleYRationalDenom
`protected long` **`scaleYRationalDenom`**

---

invScaleXRational
`protected com.sun.media.jai.util.Rational` **`invScaleXRational`**

---

invScaleYRational
`protected com.sun.media.jai.util.Rational` **`invScaleYRational`**

---

invScaleXRationalNum
`protected long` **`invScaleXRationalNum`**

---

invScaleXRationalDenom
`protected long` **`invScaleXRationalDenom`**

---

invScaleYRationalNum
`protected long` **`invScaleYRationalNum`**

---

invScaleYRationalDenom

`protected long` **`invScaleYRationalDenom`**

---

transXRational

`protected com.sun.media.jai.util.Rational` **`transXRational`**

---

transYRational

`protected com.sun.media.jai.util.Rational` **`transYRational`**

---

transXRationalNum

`protected long` **`transXRationalNum`**

---

transXRationalDenom

`protected long` **`transXRationalDenom`**

---

transYRationalNum

`protected long` **`transYRationalNum`**

---

transYRationalDenom

`protected long` **`transYRationalDenom`**

---

rationalTolerance

`protected static float` **`rationalTolerance`**

---

lpad

`private int` **`lpad`**

---

rpad

`private int` **`rpad`**

---

tpad

`private int` **`tpad`**

---

bpad

`private int` **`bpad`**

Constructor Detail

ScaleOpImage

```
public ScaleOpImage(java.awt.image.RenderedImage source,
                    BorderExtender extender,
                    TileCache cache,
                    ImageLayout layout,
                    float scaleX,
                    float scaleY,
                    float transX,
                    float transY,
                    Interpolation interp,
                    boolean cobbleSources)
```

Constructs a ScaleOpImage from a `RenderedImage` source, an optional `BorderExtender`, x and y scale and translation factors, and an `Interpolation` object. The image dimensions are determined by forward-mapping the source bounds, and are passed to the superclass constructor by means of the `layout` parameter. Other fields of the layout are passed through unchanged. If `layout` is `null`, a new `ImageLayout` will be constructor to hold the bounds information. Note that the scale factors are represented internally as Rational numbers in order to workaround inexact device specific representation of floating point numbers. For instance the floating point number 1.2 is internally represented as 1.200001, which can throw the calculations off during a forward/backward map.

The Rational approximation is valid upto the sixth decimal place.

Parameters:

`source` - a `RenderedImage`.
`extender` - a `BorderExtender`, or `null`.
`cache` - a `TileCache` object to store tiles from this `OpImage`, or `null`. If `null`, a default cache will be used.
`layout` - an `ImageLayout` optionally containing the tile grid layout, `SampleModel`, and `ColorModel`, or `null`.
`scaleX` - scale factor along x axis.
`scaleY` - scale factor along y axis.
`transX` - translation factor along x axis.
`transY` - translation factor along y axis.
`interp` - an `Interpolation` object to use for resampling.
`cobbleSources` - a boolean indicating whether `computeRect` expects contiguous sources.

Throws:

java.lang.IllegalArgumentException - if combining the source bounds with the layout parameter results in negative output width or height.

Method Detail

layoutHelper

```
private static ImageLayout layoutHelper(java.awt.image.RenderedImage source,
                                        float scaleX,
                                        float scaleY,
                                        float transX,
                                        float transY,
                                        ImageLayout il)
```

---

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:

`sourceRect` - the `Rectangle` in source coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the potentially affected destination region. or `null` if the region is unknown.

Throws:

java.lang.IllegalArgumentException - if the source index is negative or greater than that of the last source.
NullPointerException - if `sourceRect is null.`

Overrides:

mapSourceRect in class WarpOpImage

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Returns a conservative estimate of the region of a specified source that is required in order to compute the pixels of a given destination rectangle.

Parameters:

destRect - the Rectangle in destination coordinates.
sourceIndex - the index of the source image.

Returns:

a Rectangle indicating the required source region.

Throws:

java.lang.IllegalArgumentException - if the source index is negative or greater than that of the last source.
NullPointerException - if destRect is null.

Overrides:
mapDestRect in class WarpOpImage

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. If source cobbling was requested at construction time, the source tile boundaries are overlayed onto the destination, cobbling is performed for areas that intersect multiple source tiles, and computeRect(Raster[], WritableRaster, Rectangle) is called for each of the resulting regions. Otherwise, computeRect(PlanarImage[], WritableRaster, Rectangle) is called once to compute the entire active area of the tile.

The image bounds may be larger than the bounds of the source image. In this case, samples for which there are no no corresponding sources are set to zero.

The following steps are performed in order to compute the tile: - The destination tile is backward mapped to compute the needed source. - This source is then split on tile boundaries to produce rectangles that do not cross tile boundaries. - These source rectangles are then forward mapped to produce destination rectangles, and the computeRect method is called for each corresponding pair of source and destination rectangles. - For higher order interpolations, some source cobbling across tile boundaries does occur.

Parameters:
tileX - The X index of the tile.
tileY - The Y index of the tile.

Returns:
The tile as a Raster.

Overrides:
computeTile in class WarpOpImage

javax.media.jai

Class SequentialImage

```
java.lang.Object
  |
  +--javax.media.jai.SequentialImage
```

public class **SequentialImage**
extends java.lang.Object

A class representing an image that is associated with a time stamp and a camera position. This class is used with `ImageSequence`.

See Also:
`ImageSequence`

Field Detail

image

```
public PlanarImage image
```

The image.

timeStamp

```
public float timeStamp
```

The time stamp associated with the image.

cameraPosition

```
public java.lang.Object cameraPosition
```

The camera position associated with the image. The type of this parameter is `Object` so that the application may choose any class to represent a camera position based on the individual's needs.

Constructor Detail

SequentialImage

```
public SequentialImage(PlanarImage pi,
                       float ts,
                       java.lang.Object cp)
```

Constructor.

Throws:

NullPointerException - if `pi` is `null`.
NullPointerException - if `cp` is `null`.

javax.media.jai
Class Snapshot

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.Snapshot
```

---

final class **Snapshot**
extends PlanarImage

A non-public class that holds a portion of the state associated with a SnapshotImage. A Snapshot provides the appearance of a PlanarImage with fixed contents. In order to provide this illusion, however, the Snapshot relies on the fact that it belongs to a linked list of Snapshots rooted in a particular SnapshotImage; it cannot function independently.

Field Detail

parent

`SnapshotImage **parent**`

The creator of this image.

---

next

`Snapshot **next**`

The next Snapshot in a doubly-linked list.

---

prev

`Snapshot **prev**`

The previous Snapshot in a doubly-linked list.

---

tiles

`java.util.Hashtable **tiles**`

A set of cached TileCopy elements.

---

disposed

`boolean **disposed**`

True if dispose() has been called.

Constructor Detail

Snapshot

`**Snapshot**(SnapshotImage parent)`

Constructs a Snapshot that will provide a synchronous view of a SnapshotImage at a particular moment in time.

Parameters:

`parent` - a SnapshotImage this image will be viewing.

Method Detail

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Returns the version of a tile "seen" by this Snapshot. The tile "seen" is the oldest copy of the tile made after the creation of this Snapshot; it may be held in the tiles Hashtable of this Snapshot or one of its successors. If no later Snapshot holds a copy of the tile, the current version of the tile from the source image is returned.

getTile is synchronized in order to prevent calls to dispose(), which will cause the list of Snapshots to change, from occurring at the same time as the walking of the list.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
Returns:
the tile as a Raster.
Overrides:
getTile in class PlanarImage

---

setNext

```
void setNext(Snapshot next)
```

Sets the next Snapshot in the list to a given Snapshot.
Parameters:
`next` - the next Snapshot in the list.

---

setPrev

```
void setPrev(Snapshot prev)
```

Sets the previous Snapshot in the list to a given Snapshot.
Parameters:
`prev` - the previous Snapshot in the list.

---

hasTile

```
boolean hasTile(int tileX,
                int tileY)
```

Returns true if this Snapshot already stores a version of a specified tile.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
Returns:
true if this Snapshot holds a copy of the tile.

---

addTile

```
void addTile(java.awt.image.Raster tile,
             int tileX,
             int tileY)
```

Stores a given tile in this Snapshot. The caller should not attempt to store more than one version of a given tile.
Parameters:
`tile` - a Raster containing the tile data.
`tileX` - the tile’s column within the image tile grid.
`tileY` - the tile’s row within the image tile grid.

---

dispose

```
public void dispose()
```

This image will no longer be referenced by the user.
Overrides:
dispose in class PlanarImage

javax.media.jai

Class SnapshotImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.SnapshotImage
```

public class **SnapshotImage**
extends PlanarImage
implements java.awt.image.TileObserver

A class providing an arbitrary number of synchronous views of a possibly changing WritableRenderedImage. SnapshotImage is responsible for stabilizing changing sources in order to allow deferred execution of operations dependent on such sources.

Any RenderedImage may be used as the source of a SnapshotImage; if it is a WritableRenderedImage, the SnapshotImage will register itself as a TileObserver and make copies of tiles that are about to change. Multiple versions of each tile are maintained internally, as long as they are in demand. SnapshotImage is able to track demand and should be able to simply forward requests for tiles to the source most of the time, without the need to make a copy.

When used as a source, calls to getTile will simply be passed along to the source. In other words, SnapshotImage is completely transparent. However, by calling createSnapshot() an instance of a non-public PlanarImage subclass (called Sanpshot in this implementation) will be created and returned. This image will always return tile data with contents as of the time of its construction.

When a particular Snapshot is no longer needed, its dispose() method may be called. The dispose() method will be called automatically when the Snapshot is finalized by the garbage collector. Disposing of the Snapshot allows tile data held by the Snapshot that is not needed by any other Snapshot to be disposed of as well.

This implementation of SnapshotImage makes use of a doubly-linked list of Snapshot objects. A new Snapshot is added to the tail of the list whenever createSnapshot() is called. Each Snapshot has a cache containing copies of any tiles that were writable at the time of its construction, as well as any tiles that become writable between the time of its construction and the construction of the next Snapshot.

When asked for a tile, a Snapshot checks its local cache and returns its version of the tile if one is found. Otherwise, it forwards the request onto its successor. This process continues until the latest Snapshot is reached; if it does not contain a copy of the tile, the tile is requested from the real source image.

When a Snapshot is no longer needed, its dispose() method attempts to push the contents of its tile cache back to the previous Snapshot in the linked list. If that image possesses a version of the same tile, the tile is not pushed back and may be discarded.

See Also:
`RenderedImage`, `TileObserver`, `WritableRenderedImage`, `PlanarImage`

Field Detail

source

`private PlanarImage` **`source`**

The real image source.

tail

`private Snapshot` **`tail`**

The last entry in the list of Snapshots, initially null.

activeTiles

`private java.util.HashSet` **`activeTiles`**

The set of active tiles, represented as a HashSet of Points.

Constructor Detail

SnapshotImage

```
public SnapshotImage(PlanarImage source)
```

Constructs a SnapshotImage from a PlanarImage source.
Parameters:
`source` - a PlanarImage source.
Throws:
java.lang.IllegalArgumentException - if source is null.

Method Detail

getTrueSource

```
protected PlanarImage getTrueSource()
```

Returns the PlanarImage source of this SnapshotImage.
Returns:
a PlanarImage that is the source of data for this image.

---

setTail

```
void setTail(Snapshot tail)
```

Sets the reference to the most current Snapshot to a given Snapshot.
Parameters:
`tail` - a reference to the new most current Snapshot.

---

getTail

```
Snapshot getTail()
```

Returns a reference to the most current Snapshot.
Returns:
the Snapshot at the tail end of the list.

---

createTileCopy

```
private java.awt.image.Raster createTileCopy(int tileX,
                                             int tileY)
```

Creates and returns a Raster copy of a given source tile.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
Returns:
a newly-constructed Raster containing a copy of the tile data.

---

createSnapshot

```
public PlanarImage createSnapshot()
```

Creates a snapshot of this image. This snapshot may be used indefinitely, and will always appear to have the pixel data that this image has currently. The snapshot is semantically a copy of this image but may be implemented in a more efficient manner. Multiple snapshots taken at different times may share tiles that have not changed, and tiles that are currently static in this image’s source do not need to be copied at all.
Returns:
a PlanarImage snapshot.
Overrides:
createSnapshot in class PlanarImage

---

tileUpdate

```
public void tileUpdate(java.awt.image.WritableRenderedImage source,
                       int tileX,
                       int tileY,
                       boolean willBeWritable)
```

Receives the information that a tile is either about to become writable, or is about to become no longer writable.

Specified by:

tileUpdate in interface java.awt.image.TileObserver

Parameters:

`source` - the WritableRenderedImage for which we are an observer.
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
`willBeWritable` - true if the tile is becoming writable.

---

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Returns a non-snapshotted tile from the source.

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:

the tile as a Raster.

Overrides:

getTile in class PlanarImage

javax.media.jai
Class SnapshotProxy

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.SnapshotProxy
```

final class **SnapshotProxy**
extends PlanarImage

A proxy for Snapshot that calls Snapshot.dispose() when finalized. No references to a SnapshotProxy are held internally, only user references. Thus it will be garbage collected when the last user reference is relinquished. The Snapshot's dispose() method is called from SnapshotProxy.finalize(), ensuring that all of the resources held by the Snapshot will become collectable.

Field Detail

parent

`Snapshot` **`parent`**

The parent Snapshot to which we forward getTile() calls.

Constructor Detail

SnapshotProxy

`SnapshotProxy``(Snapshot parent)`

Construct a new proxy for a given Snapshot.

Parameters:

`parent` - the Snapshot to which method calls will be forwarded.

Method Detail

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Forwards a tile request to the parent Snapshot.

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:

the tile as a Raster.

Overrides:

getTile in class PlanarImage

dispose

`public void` **`dispose`**`()`

Disposes of resources held by this proxy.

Overrides:

dispose in class PlanarImage

javax.media.jai
Class SourcelessOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.SourcelessOpImage
```

public abstract class **SourcelessOpImage**
extends OpImage

An abstract base class for image operators that have no image sources.

SourcelessOpImage is intended as a convenient superclass for OpImages that have no source image. Some examples are constant color images, file readers, protocol-based network readers, and mathematically-defined imagery such as fractals.

The `computeTile` method of this class will call the `computeRect(PlanarImage[], WritableRaster, Rectangle)` method of the subclass to perform the computation. The first argument will be `null` as there are no source images.

See Also:
`PointOpImage`

Constructor Detail

SourcelessOpImage

```
public SourcelessOpImage(int minX,
                         int minY,
                         int width,
                         int height,
                         java.awt.image.SampleModel sampleModel,
                         TileCache cache,
                         ImageLayout layout)
```

Constructs a `SourcelessOpImage`. The image bounds and `SampleModel` are set explicitly; other layout parameters may be set using the `layout` parameter. The min X, min Y, width, height, and `SampleModel` fields of the `layout` parameter are ignored.

If `sampleModel` is `null`, no exceptions will be thrown. However, the caller must be sure to set the `sampleModel` instance variable before construction terminates. This feature allows subclasses that require external computation such as file loading to defer the determination of their `SampleModel` until after the call to `super`.

Similarly, `minX`, `minY`, `width`, and `height` may be dummy values if care is taken to manually set all values that depend on them, namely the tile grid offset, tile size, and `SampleModel` width and height.

The tile dimensions, tile grid X and Y offsets, and `ColorModel` of the output will be set in the standard way by the `OpImage` constructor.

Parameters:
`cache` - a `TileCache` object to store tiles from this `OpImage`, or `null`. If `null`, a default cache will be used.
`layout` - an `ImageLayout` describing the layout.

Method Detail

layoutHelper

```
private static ImageLayout layoutHelper(int minX,
                                        int minY,
                                        int width,
                                        int height,
                                        java.awt.image.SampleModel sampleModel,
                                        ImageLayout il)
```

computesUniqueTiles

```
public boolean computesUniqueTiles()
```

Returns false as SourcelessOpImages often return Rasters via computeTile() tile that are internally cached. Some subclasses may want to override this method and return true.

Overrides:

computesUniqueTiles in class OpImage

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. Since the operation has no sources, there is no need to worry about cobbling.

Subclasses should implement the `computeRect(PlanarImage[], WritableRaster, Rectangle)` method to perform the actual computation.

Parameters:

`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Overrides:

computeTile in class OpImage

---

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Throws an IllegalArgumentException since the image has no image sources.

Parameters:

`sourceRect` - ignored.
`sourceIndex` - ignored.

Throws:

java.lang.IllegalArgumentException - since the image has no sources.

Overrides:

mapSourceRect in class OpImage

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Throws an IllegalArgumentException since the image has no image sources.

Parameters:

`destRect` - ignored.
`sourceIndex` - ignored.

Throws:

java.lang.IllegalArgumentException - since the image has no sources.

Overrides:

mapDestRect in class OpImage

javax.media.jai

Class StatisticsOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.StatisticsOpImage
```

public abstract class **StatisticsOpImage**
extends OpImage

An abstract base class for image operators that compute statistics on a given region of an image, and with a given sampling rate.

`StatisticsOpImage` simply passes pixels through unchanged from its parent image. However, the desired statistics are computed on demand and made available as a property or set of properties on the image.

All instances of `StatisticsOpImage` make use of a region of interest, specified as a `ROI` object. Additionally, they may perform spatial subsampling of the region of interest according to `xPeriod` and `yPeriod` parameters that may vary from 1 (sample every pixel of the region of interest) upwards. This allows the speed and quality of statistics gathering to be traded off against one another.

Subclasses provide implementations of the `getStatisticsNames`, `createStatistics`, and `accumulateStatistics` methods.

See Also:
`OpImage`

Field Detail

roi

`protected ROI` **`roi`**

The region of interest over which to compute the statistics.

xStart

`protected int` **`xStart`**

The X coordinate of the initial sample.

yStart

`protected int` **`yStart`**

The Y coordinate of the initial sample.

xPeriod

`protected int` **`xPeriod`**

The horizontal sampling rate.

yPeriod

`protected int` **`yPeriod`**

The vertical sampling rate.

maxWidth

`protected int` **`maxWidth`**

The largest allowable width of the source argument to accumulateStatistics. Subclasses may set this value by means of the corresponding constructor argument.

---

maxHeight

```
protected int maxHeight
```

The largest allowable height of the source argument to accumulateStatistics. Subclasses may set this value by means of the corresponding constructor argument.

---

properties

```
protected java.util.Hashtable properties
```

A `Hashtable` containing all the properties generated, hashed by property names.

Constructor Detail

StatisticsOpImage

```
public StatisticsOpImage(java.awt.image.RenderedImage source,
                         ROI roi,
                         int xStart,
                         int yStart,
                         int xPeriod,
                         int yPeriod,
                         int maxWidth,
                         int maxHeight)
```

Constructs a StatisticsOpImage. The image layout is copied from the source image.

Parameters:

`source` - A RenderedImage.
`roi` - The region of interest, as a ROI.
`xStart` - The initial X sample coordinate.
`yStart` - The initial Y sample coordinate.
`xPeriod` - The X sampling rate.
`yPeriod` - The Y sampling rate.
`maxWidth` - The largest allowed width for processing.
`maxHeight` - The largest allowed height for processing.

Method Detail

layoutHelper

```
private static ImageLayout layoutHelper(java.awt.image.RenderedImage source)
```

---

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Maps the source rectangle into destination space unchanged.

Parameters:

`sourceRect` - the Rectangle in source coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the required source region.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `sourceRect` is `null`.

Overrides:

mapSourceRect in class OpImage

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Maps the destination rectangle into source space unchanged.

Parameters:

`destRect` - the Rectangle in destination coordinates.
`sourceIndex` - the index of the source image.

Returns:
a `Rectangle` indicating the valid destination region.
Throws:
java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `destRect` is `null`.
Overrides:
mapDestRect in class OpImage

---

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Returns a tile for reading. The tile request is simply forwarded to the source image.
Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.
Returns:
the tile as a Raster.
Overrides:
getTile in class OpImage

---

getTiles

```
public java.awt.image.Raster[] getTiles(java.awt.Point[] tileIndices)
```

Returns a list of tiles. The request is simply forwarded to the source image.
Parameters:
`tileIndices` - The indices of the tiles requested.
Overrides:
getTiles in class OpImage

---

tileIntersectsROI

```
private final boolean tileIntersectsROI(int tileX,
                                        int tileY)
```

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
```

Returns one of the available statistics as a property. If the property name is not recognized, java.awt.Image.UndefinedProperty will be returned.
Overrides:
getProperty in class PlanarImage

---

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns a list of property names that are recognized by this image.
Returns:
an array of Strings containing valid property names.
Overrides:
getPropertyNames in class PlanarImage

---

getStatisticsNames

```
protected abstract java.lang.String[] getStatisticsNames()
```

Returns a list of names of statistics understood by this class.

---

createStatistics

```
protected abstract java.lang.Object createStatistics(java.lang.String name)
```

Returns an object that will be used to gather the named statistic.

Parameters:

`name` - the name of the statistic to be gathered.

---

accumulateStatistics

```
protected abstract void accumulateStatistics(java.lang.String name,
                                             java.awt.image.Raster source,
                                             java.lang.Object stats)
```

Accumulates statistics on the specified region into the previously created statistics object. The region of interest and X and Y sampling rate should be respected.

Parameters:

`name` - the name of the statistic to be gathered.
`source` - a Raster containing source pixels. The dimensions of the Raster will not exceed maxWidth x maxHeight.
`stats` - a statistics object generated by a previous call to createStatistics.

javax.media.jai
Class Storage

```
java.lang.Object
  |
  +--javax.media.jai.Storage
```

class **Storage**
extends java.lang.Object

Field Detail

name

`java.lang.String name`

path

`java.lang.String path`

product

`java.lang.String product`

registerName

`java.lang.String registerName`

Constructor Detail

Storage

```
Storage(java.lang.String name,
        java.lang.String path,
        java.lang.String product,
        java.lang.String registerName)
```

javax.media.jai

Class Store

```
java.lang.Object
   |
  +--javax.media.jai.Store
```

class **Store**
extends java.lang.Object

Field Detail

product

```
java.lang.String product
```

object1

```
java.lang.Object object1
```

object2

```
java.lang.Object object2
```

Constructor Detail

Store

```
Store(java.lang.String product,
      java.lang.Object object1,
      java.lang.Object object2)
```

javax.media.jai
Interface TileCache

public abstract interface **TileCache**

A class implementing a caching mechanism for image tiles.

TileCache provides a central place for `OpImage`s to cache tiles they have computed. The tile cache is created with a given tileCapacity measured in Rasters amd a given memoryCapacity measured in bytes.

Method Detail

add

```
public void add(java.awt.image.RenderedImage owner,
                int tileX,
                int tileY,
                java.awt.image.Raster data)
```

Adds a tile to the cache.

Parameters:

`owner` - The `RenderedImage` that the tile belongs to.
`tileX` - The X index of the tile in the owner's tile grid.
`tileY` - The Y index of the tile in the owner's tile grid.
`data` - A `Raster` containing the tile data.

remove

```
public void remove(java.awt.image.RenderedImage owner,
                   int tileX,
                   int tileY)
```

Advises the cache that a tile is no longer needed. It is legal to implement this method as a no-op.

Parameters:

`owner` - The `RenderedImage` that the tile belongs to.
`tileX` - The X index of the tile in the owner's tile grid.
`tileY` - The Y index of the tile in the owner's tile grid.

getTile

```
public java.awt.image.Raster getTile(java.awt.image.RenderedImage owner,
                                     int tileX,
                                     int tileY)
```

Retrieves a tile. Returns `null` if the tile is not present in the cache.

Parameters:

`owner` - The `RenderedImage` that the tile belongs to.
`tileX` - The X index of the tile in the owner's tile grid.
`tileY` - The Y index of the tile in the owner's tile grid.

removeTiles

```
public void removeTiles(java.awt.image.RenderedImage owner)
```

Advises the cache that all tiles associated with a given image are no longer needed. It is legal to implement this method as a no-op.

Parameters:

`owner` - The `RenderedImage` owner of the tiles to be removed.

flush

```
public void flush()
```

Advises the cache that all of its tiles may be discarded. It is legal to implement this method as a no-op.

setTileCapacity

`public void setTileCapacity(int tileCapacity)`

Sets the tile capacity to a desired number of tiles. If the capacity is smaller than the current capacity, tiles are flushed from the cache.

Parameters:

`tileCapacity` - The new capacity, in tiles.

---

getTileCapacity

`public int getTileCapacity()`

Returns the tile capacity in tiles.

---

setMemoryCapacity

`public void setMemoryCapacity(long memoryCapacity)`

Sets the memory capacity to a desired number of bytes. If the memory capacity is smaller than the amount of memory currently used by the cache, tiles are flushed until the TileCache’s memory usage is less than memoryCapacity.

Parameters:

`memoryCapacity` - The new capacity, in bytes.

---

getMemoryCapacity

`public long getMemoryCapacity()`

Returns the memory capacity in bytes.

javax.media.jai

Class TileCopy

```
java.lang.Object
  |
  +--javax.media.jai.TileCopy
```

---

final class **TileCopy**
extends java.lang.Object

A (Raster, X, Y) tuple.

Field Detail

tile

`java.awt.image.Raster` **`tile`**

The tile's Raster data.

---

tileX

`int` **`tileX`**

The tile's column within the image tile grid.

---

tileY

`int` **`tileY`**

The tile's row within the image tile grid.

Constructor Detail

TileCopy

```
TileCopy(java.awt.image.Raster tile,
         int tileX,
         int tileY)
```

Constructs a TileCopy object given the tile's Raster data and its location in the tile grid.

Parameters:

`tile` - the Raster containing the tile's data.
`tileX` - the tile's X position in the tile grid.
`tileY` - the tile's X position in the tile grid.

javax.media.jai
Interface TileScheduler

public abstract interface **TileScheduler**

A class implementing a mechanism for scheduling tile calculation. In various implementations tile computation may make use of multithreading and multiple simultaneous network connections for improved performance.

Method Detail

scheduleTile

```
public java.awt.image.Raster scheduleTile(OpImage target,
                                          int tileX,
                                          int tileY)
```

Schedules a tile for computation. Called by `OpImage.getTile()`. After performing a dependency analysis, this method makes `OpImage.computeTile()` calls for source tiles needed to calculate the ultimate destination tile.

Parameters:

`target` - An `OpImage` whose tile is to be computed.
`tileX` - The X index of the tile to be computed.
`tileY` - The Y index of the tile to be computed.

Returns:

A `Raster` containing the contents of the tile.

scheduleTiles

```
public java.awt.image.Raster[] scheduleTiles(OpImage target,
                                             java.awt.Point[] tileIndices)
```

Schedules a list of tiles for computation. Called by `OpImage.getTiles`. After performing a dependency analysis, this method makes `OpImage.computeTile()` calls for source tiles needed to calculate the ultimate destination tile.

Parameters:

`target` - The OpImage to schedule tiles from.
`tileIndices` - A list of tileIndices indicating which tiles to schedule for computation.

Returns:

An array of `Raster` containing a computed raster for every tileIndex passed in.

prefetchTiles

```
public void prefetchTiles(PlanarImage target,
                          java.awt.Point[] tileIndices)
```

Hints to the TileScheduler that the given tiles from the given PlanarImage might be needed in the near future. Some TileScheduler implementations will spawn a low priority thread to compute the tiles while others may ignore the hint.

Parameters:

`target` - The OpImage to prefetch tiles from.
`tileIndices` - A list of tileIndices indicating which tiles to prefetch.

javax.media.jai

Class TiledImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.TiledImage
```

public class **TiledImage**
extends PlanarImage
implements java.awt.image.WritableRenderedImage

A concrete implementation of WritableRenderedImage.

`TiledImage` is the main class for writable images in JAI. `TiledImage` provides a straghtforward implementation of the `WritableRenderedImage` interface, taking advantage of that interface's ability to describe images with multiple tiles. The tiles of a `WritableRenderedImage` must share a `SampleModel`, which determines their width, height, and pixel format. The tiles form a regular grid, which may occupy any rectangular region of the plane. Tile pixels that exceed the image's stated bounds have undefined values.

The contents of a `TiledImage` are defined by a single `PlanarImage` source provided by means of one of the `set()` methods. The `set()` methods provide a way to selectively overwrite a portion of a `TiledImage`, possibly using a soft-edged mask.

`TiledImage` also supports direct manipulation of pixels by means of the `getWritableTile()` method. This method returns a WritableRaster that can be modified directly. Such changes become visible to readers according to the regular thread synchronization rules of the Java virtual machine; JAI makes no additional guarantees. When a writer is finished modifying a tile, it should call `releaseWritableTile()`. A shortcut is to call setData(), which copies a rectangular region from a supplied Raster directly into the `TiledImage`.

A final way to modify the contents of a `TiledImage` is through calls to `createGraphics()`. This returns a `Graphics2D` object that can be used to draw line art, text, and images in the usual AWT manner.

A `TiledImage` does not attempt to maintain synchronous state on its own. That task is left to `SnapshotImage`. If a synchronous (unchangeable) view of a `TiledImage` is desired, its `createSnapshot()` method must be used. Otherwise, changes due to calls to set() or direct writing of tiles by objects that call `getWritableTile()` will be visible.

`TiledImage` does not actually cause its tiles to be computed until their contents are demanded. Once a tile has been computed, its contents may be discarded if it can be determined that it can be recomputed identically from the source. The `lockTile()` method forces a tile to be computed and maintained for the lifetime of the `TiledImage`.

See Also:
`SnapshotImage`, `RenderedImage`, `WritableRenderedImage`

Field Detail

tilesX

`protected int` **`tilesX`**

The number of tiles in the X direction.

tilesY

`protected int` **`tilesY`**

The number of tiles in the Y direction.

minTileX

`protected int` **`minTileX`**

The index of the leftmost column of tiles.

minTileY

`protected int` **`minTileY`**

The index of the uppermost row of tiles.

tiles

`protected java.awt.image.WritableRaster[][] **tiles**`

The tile array.

---

writers

`protected int[][] **writers**`

The number of writers of each tile; -1 indicates a locked tile.

---

tileObservers

`protected java.util.Vector **tileObservers**`

The current set of TileObservers.

---

src

`protected PlanarImage **src**`

The source image for uncomputed tiles.

---

imageBounds

`private java.awt.Rectangle **imageBounds**`

---

parent

`private TiledImage **parent**`

---

ancestorSampleModel

`private java.awt.image.SampleModel **ancestorSampleModel**`

---

bandList

`private int[] **bandList**`

---

numWritableTiles

`private int[] **numWritableTiles**`

---

srcROI

`private ROI **srcROI**`

---

overlapBounds

`private java.awt.Rectangle **overlapBounds**`

Constructor Detail

TiledImage

```
public TiledImage(int minX,
                  int minY,
                  int width,
                  int height,
                  int tileGridXOffset,
                  int tileGridYOffset,
                  java.awt.image.SampleModel sampleModel,
                  java.awt.image.ColorModel colorModel)
```

Constructs a `TiledImage` with a given layout, `SampleModel`, and `ColorModel`.
Parameters:
`minX` - The X coordinate of the upper-left pixel
`minY` - The Y coordinate of the upper-left pixel.
`width` - The width of the image.
`height` - The height of the image.
`tileGridXOffset` - The X coordinate of the upper-left pixel of tile (0, 0).
`tileGridYOffset` - The Y coordinate of the upper-left pixel of tile (0, 0).
`sampleModel` - A `SampleModel` with which to be compatible.
`colorModel` - A `ColorModel` to associate with the image.

TiledImage

```
private TiledImage(TiledImage parent,
                   int minX,
                   int minY,
                   int width,
                   int height,
                   int tileGridXOffset,
                   int tileGridYOffset,
                   java.awt.image.SampleModel sampleModel,
                   java.awt.image.ColorModel colorModel)
```

TiledImage

```
public TiledImage(java.awt.Point origin,
                  java.awt.image.SampleModel sampleModel,
                  int tileWidth,
                  int tileHeight)
```

Constructs a `TiledImage` with a `SampleModel` that is compatible with a given `SampleModel`, and given tile dimensions. The width and height are taken from the `SampleModel`, and the image begins at a specified point.
Parameters:
`origin` - A `Point` indicating the image's upper left corner.
`sampleModel` - A `SampleModel` with which to be compatible.
`tileWidth` - The desired tile width.
`tileHeight` - The desired tile height.

TiledImage

```
public TiledImage(java.awt.image.SampleModel sampleModel,
                  int tileWidth,
                  int tileHeight)
```

Constructs a `TiledImage` starting at the global coordinate origin.
Parameters:
`sampleModel` - A `SampleModel` with which to be compatible.
`tileWidth` - The desired tile width.
`tileHeight` - The desired tile height.

Method Detail

initTileGrid

```
private void initTileGrid(TiledImage parent)
```

---

createInterleaved

```
public static TiledImage createInterleaved(int minX,
                                           int minY,
                                           int width,
                                           int height,
                                           int numBands,
                                           int dataType,
                                           int tileWidth,
                                           int tileHeight,
                                           int[] bandOffsets)
```

Returns a `TiledImage` making use of an interleaved `SampleModel` with a given layout, number of bands, and data type.
Parameters:
`minX` - The X coordinate of the upper-left pixel
`minY` - The Y coordinate of the upper-left pixel.
`width` - The width of the image.
`height` - The height of the image.
`numBands` - The number of bands in the image.
`dataType` - The data type, from among the constants `DataBuffer.TYPE_*`.
`tileWidth` - The tile width.
`tileHeight` - The tile height.
`bandOffsets` - An array of non-duplicated integers between 0 and `numBands - 1` of length `numBands` indicating the relative offset of each band.

---

createBanded

```
public static TiledImage createBanded(int minX,
                                      int minY,
                                      int width,
                                      int height,
                                      int dataType,
                                      int tileWidth,
                                      int tileHeight,
                                      int[] bankIndices,
                                      int[] bandOffsets)
```

Returns a `TiledImage` making use of an banded `SampleModel` with a given layout, number of bands, and data type.
Parameters:
`minX` - The X coordinate of the upper-left pixel
`minY` - The Y coordinate of the upper-left pixel.
`width` - The width of the image.
`height` - The height of the image.
`dataType` - The data type, from among the constants `DataBuffer.TYPE_*`.
`tileWidth` - The tile width.
`tileHeight` - The tile height.
`bankIndices` - An array of `int`s indicating the index of the bank to use for each band. Bank indices may be duplicated.
`bandOffsets` - An array of integers indicating the starting offset of each band within its bank. Bands stored in the same bank must have sufficiently different offsets so as not to overlap.

---

overlayPixels

```
private void overlayPixels(java.awt.image.WritableRaster tile,
                           java.awt.image.RenderedImage im,
                           java.awt.Rectangle rect)
```

Overlays a rectangular area of pixels from an image onto a tile.
Parameters:
`tile` -
`im` -
`rect` -

---

overlayPixels

```
private void overlayPixels(java.awt.image.WritableRaster tile,
                           java.awt.image.RenderedImage im,
                           java.awt.geom.Area a)
```

Overlays a set of pixels described by an Area from an image onto a tile.
Parameters:
`tile` -
`im` -
`a` -

---

overlayPixels

```
private void overlayPixels(java.awt.image.WritableRaster tile,
                           java.awt.image.RenderedImage im,
                           java.awt.Rectangle rect,
                           int[][] bitmask)
```

Overlays a set of pixels described by a bitmask onto a tile.

---

set

```
public void set(java.awt.image.RenderedImage im)
```

Overlays a given `RenderedImage` on top of the current contents of the `TiledImage`. The source image must have a `SampleModel` compatible with that of this image. If the source image does not overlap this image then invoking this method will have no effect.

Parameters:

`im` - A `RenderedImage` source to overlay.

---

set

```
public void set(java.awt.image.RenderedImage im,
                ROI roi)
```

Overlays a given `RenderedImage` on top of the current contents of the `TiledImage`. The source image must have a `SampleModel` compatible with that of this image. If the source image and the region of interest do not both overlap this image then invoking this method will have no effect.

Parameters:

`im` - A `RenderedImage` source to overlay.
`roi` - The region of interest.

---

getGraphics

```
public java.awt.Graphics getGraphics()
```

Creates a `Graphics` object that can be used to paint text and graphics onto the `TiledImage`. The `TiledImage` must be of integral data type or an `UnsupportedOperationException` will be thrown.

Overrides:

getGraphics in class PlanarImage

---

createGraphics

```
public java.awt.Graphics2D createGraphics()
```

Creates a `Graphics2D` object that can be used to paint text and graphics onto the `TiledImage`. The `TiledImage` must be of integral data type or an `UnsupportedOperationException` will be thrown.

---

getSubImage

```
public TiledImage getSubImage(int x,
                              int y,
                              int w,
                              int h,
                              int[] bandSelect)
```

Returns a `TiledImage` that shares the tile `Raster`s of this image. The returned image occupies a sub-area of the parent image, and possesses a possibly permuted subset of the parent's bands. The two images share a common coordinate system.

The image bounds are clipped against the bounds of the parent image.

Parameters:

`x` - the minimum X coordinate of the subimage.
`y` - the minimum Y coordinate of the subimage.
`w` - the width of the subimage.
`h` - the height of the subimage.
`bandSelect` - an array of band indices; if null, all bands are selected.

---

getSubImage

```
public TiledImage getSubImage(int x,
                              int y,
                              int w,
                              int h)
```

Returns a `TiledImage` that shares the tile `Raster`s of this image. The returned image occupies a subarea of the parent image. The two images share a common coordinate system.

The image bounds are clipped against the bounds of the parent image.

Parameters:

`x` - the minimum X coordinate of the subimage.
`y` - the minimum Y coordinate of the subimage.
`w` - the width of the subimage.
`h` - the height of the subimage.

---

getSubImage

```
public TiledImage getSubImage(int[] bandSelect)
```

Returns a `TiledImage` that shares the tile `Raster`s of this image. The returned image occupies the same area as the parent image, and possesses a possibly permuted subset of the parent's bands.

Parameters:

`bandSelect` - an array of band indices.

---

createTile

```
private void createTile(int tileX,
                        int tileY)
```

Forces the requested tile to be computed if has not already been so and if a source is available.

---

getTile

```
public java.awt.image.Raster getTile(int tileX,
                                     int tileY)
```

Retrieves a particular tile from the image for reading only. The tile will be computed if it hasn't been previously. Any attempt to write to the tile will produce undefined results.

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Overrides:

getTile in class PlanarImage

---

getWritableTile

```
public java.awt.image.WritableRaster getWritableTile(int tileX,
                                                     int tileY)
```

Retrieves a particular tile from the image for reading and writing. If the tile is locked, null will be returned. Otherwise, the tile will be computed if it hasn't been previously. Writes to the tile will become visible to readers of this image as they occur.

Specified by:

getWritableTile in interface java.awt.image.WritableRenderedImage

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:

The requested tile or null if the tile is locked.

---

releaseWritableTile

```
public void releaseWritableTile(int tileX,
                                int tileY)
```

Indicates that a writer is done updating a tile. The effects of attempting to release a tile that has not been grabbed, or releasing a tile more than once are undefined.

Specified by:

releaseWritableTile in interface java.awt.image.WritableRenderedImage

Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

---

lockTile

```
protected boolean lockTile(int tileX,
                           int tileY)
```

Forces a tile to be computed, and its contents stored indefinitely. A tile may not be locked if it is currently writable. This method should only be used within JAI, in order to optimize memory allocation.

Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:
Whether the tile was successfully locked.

---

isTileLocked

```
protected boolean isTileLocked(int tileX,
                               int tileY)
```

Returns `true` if a tile is locked.

Parameters:
`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

Returns:
Whether the tile is locked.

---

setData

```
public void setData(java.awt.image.Raster r)
```

Sets a region of a `TiledImage` to be a copy of a supplied `Raster`. The `Raster`'s coordinate system is used to position it within the image. The computation of all overlapping tiles will be forced prior to modification of the data of the affected area.

Specified by:
setData in interface java.awt.image.WritableRenderedImage

Parameters:
`r` - a `Raster` containing pixels to be copied into the `TiledImage`.

---

setData

```
public void setData(java.awt.image.Raster r,
                    ROI roi)
```

Sets a region of a `TiledImage` to be a copy of a supplied `Raster`. The `Raster`'s coordinate system is used to position it within the image. The computation of all overlapping tiles will be forced prior to modification of the data of the affected area.

Parameters:
`r` - a `Raster` containing pixels to be copied into the `TiledImage`.
`roi` - The region of interest.

---

addTileObserver

```
public void addTileObserver(java.awt.image.TileObserver observer)
```

Informs this `TiledImage` that another object is interested in being notified whenever any tile becomes writable or ceases to be writable. A tile becomes writable when it is not currently writable and `getWritableTile()` is called. A tile ceases to be writable when `releaseTile()` is called and the number of calls to `getWritableTile()` and `releaseWritableTile()` are identical.

It is the responsibility of the `TiledImage` to inform all registered `TileObserver` objects of such changes in tile writability before the writer has a chance to make any modifications.

Specified by:
addTileObserver in interface java.awt.image.WritableRenderedImage

Parameters:
`observer` - An object implementing the `TileObserver` interface.

removeTileObserver

```
public void removeTileObserver(java.awt.image.TileObserver observer)
```

Informs this `TiledImage` that a particular TileObserver no longer wishes to receive updates on tile writability status. The result of attempting to remove a listener that is not registered is undefined.

Specified by:

removeTileObserver in interface java.awt.image.WritableRenderedImage

Parameters:

`observer` - An object implementing the `TileObserver` interface.

---

getWritableTileIndices

```
public java.awt.Point[] getWritableTileIndices()
```

Returns a list of tiles that are currently held by one or more writers.

Specified by:

getWritableTileIndices in interface java.awt.image.WritableRenderedImage

Returns:

An array of `Point`s representing tile indices.

---

hasTileWriters

```
public boolean hasTileWriters()
```

Returns `true` if any tile is being held by a writer, `false` otherwise. This provides a quick way to check whether it is necessary to make copies of tiles -- if there are no writers, it is safe to use the tiles directly, while registering to learn of future writers.

Specified by:

hasTileWriters in interface java.awt.image.WritableRenderedImage

---

isTileWritable

```
public boolean isTileWritable(int tileX,
                              int tileY)
```

Returns `true` if a tile has writers.

Specified by:

isTileWritable in interface java.awt.image.WritableRenderedImage

Parameters:

`tileX` - the X index of the tile.
`tileY` - the Y index of the tile.

---

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      int s)
```

Sets a sample of a pixel to a given value.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.
`s` - The value to which to set the sample.

---

getSample

```
public int getSample(int x,
                     int y,
                     int b)
```

Returns the value of a given sample of a pixel as an `int`.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      float s)
```

Sets a sample of a pixel to a given `float` value.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.
`s` - The value to which to set the sample.

getSampleFloat

```
public float getSampleFloat(int x,
                            int y,
                            int b)
```

Returns the value of a given sample of a pixel as a `float`.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      double s)
```

Sets a sample of a pixel to a given `double` value.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.
`s` - The value to which to set the sample.

getSampleDouble

```
public double getSampleDouble(int x,
                              int y,
                              int b)
```

Returns the value of a given sample of a pixel as a `double`.

Parameters:

`x` - The X coordinate of the pixel.
`y` - The Y coordinate of the pixel.
`b` - The band of the sample within the pixel.

javax.media.jai

Class TiledImageGraphics

```
java.lang.Object
  |
  +--java.awt.Graphics
        |
        +--java.awt.Graphics2D
              |
              +--javax.media.jai.TiledImageGraphics
```

class **TiledImageGraphics**
extends java.awt.Graphics2D

A concrete (i.e., non-abstract) class implementing all the methods of `Graphics2D` (and thus of `Graphics`) with a `TiledImage` as the implicit drawing canvas. The actual implementation will use Java2D to do most of the work by packaging up the image tiles in a form that Java2D can understand.

Since the public methods of this class all derive from `Graphics2D`, they are not commented individually.

The `ColorModel` for the canvas will be that of the associated `TiledImage` unless that `ColorModel` is null. If the `TiledImage` `ColorModel` is null, an attempt will first be made to deduce the `ColorModel` from the `SampleModel` of the `TiledImage` using the `createColorModel()` method of `PlanarImage`. If the `ColorModel` is still null, the default RGB `ColorModel` returned by the `getRGBdefault()` method of `ColorModel` will be used if the `TiledImage` has a compatible `SampleModel`. If no acceptable `ColorModel` can be derived an `UnsupportedOperationException` will be thrown.

See Also:
`Graphics`, `Graphics2D`, `ColorModel`, `SampleModel`, `TiledImage`

Field Detail

GRAPHICS2D_CLASS

`private static final java.lang.Class` **`GRAPHICS2D_CLASS`**

PAINT_MODE

`private static final int` **`PAINT_MODE`**

XOR_MODE

`private static final int` **`XOR_MODE`**

tiledImage

`private TiledImage` **`tiledImage`**

properties

`java.util.Hashtable` **`properties`**

renderingHints

`private java.awt.RenderingHints` **`renderingHints`**

tileWidth

`private int` **`tileWidth`**

tileHeight
`private int` **`tileHeight`**

---

tileXMinimum
`private int` **`tileXMinimum`**

---

tileXMaximum
`private int` **`tileXMaximum`**

---

tileYMinimum
`private int` **`tileYMinimum`**

---

tileYMaximum
`private int` **`tileYMaximum`**

---

colorModel
`private java.awt.image.ColorModel` **`colorModel`**

---

origin
`private java.awt.Point` **`origin`**

---

clip
`private java.awt.Shape` **`clip`**

---

color
`private java.awt.Color` **`color`**

---

font
`private java.awt.Font` **`font`**

---

paintMode
`private int` **`paintMode`**

XORColor
`private java.awt.Color` **`XORColor`**

---

background
`private java.awt.Color` **`background`**

---

composite
`private java.awt.Composite` **`composite`**

---

paint

```
private java.awt.Paint paint
```

---

stroke

```
private java.awt.Stroke stroke
```

---

transform

```
private java.awt.geom.AffineTransform transform
```

Constructor Detail

TiledImageGraphics

```
public TiledImageGraphics(TiledImage im)
```

Construct a `TiledImageGraphics` object that draws onto a particular `TiledImage`. The `TiledImage` parameter must be of integral data type or an `UnsupportedOperationException` will be thrown. Likewise, if no appropriate `ColorModel` can be derived an `UnsupportedOperationException` will be thrown.

Parameters:
`im` - The `TiledImage` which will serve as the graphics canvas.

Throws:
`UnsupportedOperationException` - if no appropriate `ColorModel` can be derived.

Method Detail

getBoundingBox

```
private static final java.awt.Rectangle getBoundingBox(int[] xPoints,
                                                       int[] yPoints,
                                                       int nPoints)
```

Determine the bounding box of the points represented by the supplied arrays of X and Y coordinates.

Parameters:
`xPoints` - An array of *x* points.
`yPoints` - An array of *y* points.
`nPoints` - The total number of points.

---

copyState

```
private void copyState(java.awt.Graphics2D g2d)
```

Copy the graphics state of the current object to a `Graphics2D` object.

Parameters:
`g2d` - The target `Graphics2D` object.

---

getBogusGraphics2D

```
private java.awt.Graphics2D getBogusGraphics2D(boolean shouldCopyState)
```

Creates a bogus `Graphics2D` object to be used to retrieve information dependent on system aspects which are image-independent.

The `dispose()` method of the `Graphics2D` object returned should be called to free the associated resources as\ soon as possible.

Parameters:
`shouldCopyState` - Whether the state of the returned `Graphics2D` should be initialized to that of the current `TiledImageGraphics` object.

Returns:
A `Graphics2D` object.

---

getColorModel

```
private static java.awt.image.ColorModel getColorModel(TiledImage ti)
```

Derive an approriate `ColorModel` for use with the underlying `BufferedImage` canvas. If an appropriate `ColorModel` cannot be derived an `UnsupportedOperationException` will be thrown.

Returns:

An appropriate `ColorModel`.

Throws:

`UnsupportedOperationException` - if no appropriate `ColorModel` can be derived.

---

doGraphicsOp

```
private boolean doGraphicsOp(int x,
                             int y,
                             int width,
                             int height,
                             java.lang.String name,
                             java.lang.Class[] argTypes,
                             java.lang.Object[] args)
```

Effect a graphics operation on the `TiledImage` by creating a `BufferedImage` for each tile in the affected region and using the corresponding `Graphics2D` to perform the equivalent operation on the tile.

Parameters:

`x` - The *x* coordinate of the upper left corner.
`y` - The *y* coordinate of the upper left corner.
`width` - The width of the region.
`height` - The height of the region.
`argTypes` - An array of the `Classes` of the arguments of the specified operation.
`args` - The arguments of the operation as an array of `Objects`.

---

doGraphicsOp

```
private boolean doGraphicsOp(java.awt.Shape s,
                             java.lang.String name,
                             java.lang.Class[] argTypes,
                             java.lang.Object[] args)
```

Effect a graphics operation on the `TiledImage` by creating a `BufferedImage` for each tile in the affected region and using the corresponding `Graphics2D` to perform the equivalent operation on the tile.

Parameters:

`s` - The encompassing `Shape`.
`argTypes` - An array of the `Classes` of the arguments of the specified operation.
`args` - The arguments of the operation as an array of `Objects`.

---

create

```
public java.awt.Graphics create()
```

Overrides:

create in class java.awt.Graphics

---

getColor

```
public java.awt.Color getColor()
```

Overrides:

getColor in class java.awt.Graphics

---

setColor

```
public void setColor(java.awt.Color c)
```

Overrides:

setColor in class java.awt.Graphics

---

setPaintMode

```
public void setPaintMode()
```

Overrides:
setPaintMode in class java.awt.Graphics

---

setXORMode

```
public void setXORMode(java.awt.Color c1)
```

Overrides:
setXORMode in class java.awt.Graphics

---

getFont

```
public java.awt.Font getFont()
```

Overrides:
getFont in class java.awt.Graphics

---

setFont

```
public void setFont(java.awt.Font font)
```

Overrides:
setFont in class java.awt.Graphics

---

getFontMetrics

```
public java.awt.FontMetrics getFontMetrics(java.awt.Font f)
```

Overrides:
getFontMetrics in class java.awt.Graphics

---

getClipBounds

```
public java.awt.Rectangle getClipBounds()
```

Overrides:
getClipBounds in class java.awt.Graphics

---

clipRect

```
public void clipRect(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
clipRect in class java.awt.Graphics

---

setClip

```
public void setClip(int x,
                    int y,
                    int width,
                    int height)
```

Overrides:
setClip in class java.awt.Graphics

---

getClip

```
public java.awt.Shape getClip()
```

Overrides:
getClip in class java.awt.Graphics

---

setClip

```
public void setClip(java.awt.Shape clip)
```

Overrides:
setClip in class java.awt.Graphics

---

copyArea

```
public void copyArea(int x,
                     int y,
                     int width,
                     int height,
                     int dx,
                     int dy)
```

Overrides:
copyArea in class java.awt.Graphics

---

drawLine

```
public void drawLine(int x1,
                     int y1,
                     int x2,
                     int y2)
```

Overrides:
drawLine in class java.awt.Graphics

---

fillRect

```
public void fillRect(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
fillRect in class java.awt.Graphics

---

clearRect

```
public void clearRect(int x,
                      int y,
                      int width,
                      int height)
```

Overrides:
clearRect in class java.awt.Graphics

---

drawRoundRect

```
public void drawRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

Overrides:
drawRoundRect in class java.awt.Graphics

---

fillRoundRect

```
public void fillRoundRect(int x,
                          int y,
                          int width,
                          int height,
                          int arcWidth,
                          int arcHeight)
```

Overrides:
fillRoundRect in class java.awt.Graphics

---

draw3DRect

```
public void draw3DRect(int x,
                       int y,
                       int width,
                       int height,
                       boolean raised)
```

Overrides:
draw3DRect in class java.awt.Graphics2D

---

fill3DRect

```
public void fill3DRect(int x,
                       int y,
                       int width,
                       int height,
                       boolean raised)
```

Overrides:
fill3DRect in class java.awt.Graphics2D

---

drawOval

```
public void drawOval(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
drawOval in class java.awt.Graphics

---

fillOval

```
public void fillOval(int x,
                     int y,
                     int width,
                     int height)
```

Overrides:
fillOval in class java.awt.Graphics

---

drawArc

```
public void drawArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

Overrides:
drawArc in class java.awt.Graphics

---

fillArc

```
public void fillArc(int x,
                    int y,
                    int width,
                    int height,
                    int startAngle,
                    int arcAngle)
```

Overrides:
fillArc in class java.awt.Graphics

---

drawPolyline

```
public void drawPolyline(int[] xPoints,
                         int[] yPoints,
                         int nPoints)
```

Overrides:
drawPolyline in class java.awt.Graphics

---

drawPolygon

```
public void drawPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

Overrides:
drawPolygon in class java.awt.Graphics

---

fillPolygon

```
public void fillPolygon(int[] xPoints,
                        int[] yPoints,
                        int nPoints)
```

Overrides:
fillPolygon in class java.awt.Graphics

---

drawString

```
public void drawString(java.lang.String str,
                       int x,
                       int y)
```

Overrides:
drawString in class java.awt.Graphics2D

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawRenderedImage

```
public void drawRenderedImage(java.awt.image.RenderedImage im,
                              java.awt.geom.AffineTransform transform)
```

Overrides:
drawRenderedImage in class java.awt.Graphics2D

---

drawRenderableImage

```
public void drawRenderableImage(java.awt.image.renderable.RenderableImage img,
                                java.awt.geom.AffineTransform xform)
```

Overrides:
drawRenderableImage in class java.awt.Graphics2D

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int x,
                         int y,
                         int width,
                         int height,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         int dx1,
                         int dy1,
                         int dx2,
                         int dy2,
                         int sx1,
                         int sy1,
                         int sx2,
                         int sy2,
                         java.awt.Color bgcolor,
                         java.awt.image.ImageObserver observer)
```

Overrides:
drawImage in class java.awt.Graphics

---

dispose

```
public void dispose()
```

Overrides:
dispose in class java.awt.Graphics

---

addRenderingHints

```
public void addRenderingHints(java.util.Map hints)
```

Overrides:
addRenderingHints in class java.awt.Graphics2D

---

draw

```
public void draw(java.awt.Shape s)
```

Overrides:
draw in class java.awt.Graphics2D

---

drawImage

```
public boolean drawImage(java.awt.Image img,
                         java.awt.geom.AffineTransform xform,
                         java.awt.image.ImageObserver obs)
```

Overrides:
drawImage in class java.awt.Graphics2D

---

drawImage

```
public void drawImage(java.awt.image.BufferedImage img,
                      java.awt.image.BufferedImageOp op,
                      int x,
                      int y)
```

Overrides:
drawImage in class java.awt.Graphics2D

---

drawString

```
public void drawString(java.lang.String s,
                       float x,
                       float y)
```

Overrides:
drawString in class java.awt.Graphics2D

---

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       int x,
                       int y)
```

Overrides:
drawString in class java.awt.Graphics2D

---

drawString

```
public void drawString(java.text.AttributedCharacterIterator iterator,
                       float x,
                       float y)
```

Overrides:
drawString in class java.awt.Graphics2D

---

drawGlyphVector

```
public void drawGlyphVector(java.awt.font.GlyphVector g,
                            float x,
                            float y)
```

Overrides:
drawGlyphVector in class java.awt.Graphics2D

---

fill

```
public void fill(java.awt.Shape s)
```

Overrides:
fill in class java.awt.Graphics2D

---

hit

```
public boolean hit(java.awt.Rectangle rect,
                   java.awt.Shape s,
                   boolean onStroke)
```

Overrides:
hit in class java.awt.Graphics2D

---

getDeviceConfiguration

```
public java.awt.GraphicsConfiguration getDeviceConfiguration()
```

Overrides:
getDeviceConfiguration in class java.awt.Graphics2D

---

setComposite

```
public void setComposite(java.awt.Composite comp)
```

Overrides:
setComposite in class java.awt.Graphics2D

---

setPaint

```
public void setPaint(java.awt.Paint paint)
```

Overrides:
setPaint in class java.awt.Graphics2D

---

setStroke

```
public void setStroke(java.awt.Stroke s)
```

Overrides:
setStroke in class java.awt.Graphics2D

---

setRenderingHint

```
public void setRenderingHint(java.awt.RenderingHints.Key hintKey,
                             java.lang.Object hintValue)
```

Overrides:
setRenderingHint in class java.awt.Graphics2D

---

getRenderingHint

```
public java.lang.Object getRenderingHint(java.awt.RenderingHints.Key hintKey)
```

Overrides:
getRenderingHint in class java.awt.Graphics2D

---

setRenderingHints

```
public void setRenderingHints(java.util.Map hints)
```

Overrides:
setRenderingHints in class java.awt.Graphics2D

---

getRenderingHints

```
public java.awt.RenderingHints getRenderingHints()
```

Overrides:
getRenderingHints in class java.awt.Graphics2D

---

translate

```
public void translate(int x,
                      int y)
```

Overrides:
translate in class java.awt.Graphics2D

---

translate

```
public void translate(double x,
                      double y)
```

Overrides:
translate in class java.awt.Graphics2D

---

rotate

```
public void rotate(double theta)
```

Overrides:
rotate in class java.awt.Graphics2D

---

rotate

```
public void rotate(double theta,
                   double x,
                   double y)
```

Overrides:
rotate in class java.awt.Graphics2D

---

scale

```
public void scale(double sx,
                  double sy)
```

Overrides:
scale in class java.awt.Graphics2D

---

shear

```
public void shear(double shx,
                  double shy)
```

Overrides:
shear in class java.awt.Graphics2D

---

transform

```
public void transform(java.awt.geom.AffineTransform Tx)
```

Overrides:
transform in class java.awt.Graphics2D

---

setTransform

```
public void setTransform(java.awt.geom.AffineTransform Tx)
```

Overrides:
setTransform in class java.awt.Graphics2D

---

getTransform

`public java.awt.geom.AffineTransform getTransform()`

Overrides:

getTransform in class java.awt.Graphics2D

---

getPaint

`public java.awt.Paint getPaint()`

Overrides:

getPaint in class java.awt.Graphics2D

---

getComposite

`public java.awt.Composite getComposite()`

Overrides:

getComposite in class java.awt.Graphics2D

---

setBackground

`public void setBackground(java.awt.Color color)`

Overrides:

setBackground in class java.awt.Graphics2D

---

getBackground

`public java.awt.Color getBackground()`

Overrides:

getBackground in class java.awt.Graphics2D

---

getStroke

`public java.awt.Stroke getStroke()`

Overrides:

getStroke in class java.awt.Graphics2D

---

clip

`public void clip(java.awt.Shape s)`

Overrides:

clip in class java.awt.Graphics2D

---

getFontRenderContext

`public java.awt.font.FontRenderContext getFontRenderContext()`

Overrides:

getFontRenderContext in class java.awt.Graphics2D

javax.media.jai
Class UntiledOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.UntiledOpImage
```

public abstract class **UntiledOpImage**
extends OpImage

A general class for single-source operations in which the values of all pixels in the source image contribute to the value of each pixel in the destination image.

The output image will have a single tile, regardless of the `ImageLayout` settings passed to the constructor.

Subclasses should implement the `computeImage` method which requests computation of the entire image at once.

See Also:
`OpImage`

Field Detail

tileDependencies

```
private java.awt.Point[] tileDependencies
```

The tile dependency array: needs to be computed only once.

Constructor Detail

UntiledOpImage

```
public UntiledOpImage(java.awt.image.RenderedImage source,
                      TileCache cache,
                      ImageLayout layout)
```

Constructs an `UntiledOpImage`. The image layout is copied from the source image. The tile grid layout, `SampleModel, and ColorModel may optionally be specified by an ImageLayout object. Cobbling will be performed on the source image as needed.`

Parameters:
`source - a RenderedImage.`
`cache - a TileCache object to store tiles from this OpImage, or null. If null, a default cache will be used.`
`layout - an ImageLayout optionally containing the SampleModel, and ColorModel. The tile grid layout information will be overridden in order to ensure that the image has a single tile.`

Method Detail

layoutHelper

```
private static ImageLayout layoutHelper(ImageLayout layout,
                                        java.awt.image.RenderedImage source)
```

Creates the `ImageLayout` for the image. If the layout parameter is null, create a new `ImageLayout` using as a fallback equivalent to which the `RenderedImage` would have. Also, force the tile grid offset to equal the image origin and the tile width and height to be equal to the image width and height, respectively, thereby forcing the image to have a single tile.

Parameters:
`layout` - The `ImageLayout` to be cloned; may be null.
`source` - The `RenderedImage` the attributes of which are to be used as fallbacks in creating a new `ImageLayout`.

Returns:
The `ImageLayout` to be used.

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:

`sourceRect` - the `Rectangle` in source coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `sourceRect` is `null`.

Overrides:

mapSourceRect in class OpImage

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Returns a conservative estimate of the region of a specified source that is required in order to compute the pixels of a given destination rectangle.

Parameters:

`destRect` - the Rectangle in destination coordinates.
`sourceIndex` - the index of the source image.

Returns:

a `Rectangle` indicating the required source region.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class OpImage

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. The entire source is cobbled together and `computeImage` is called to produce the single output tile.

Parameters:

`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Overrides:

computeTile in class OpImage

---

computeImage

```
protected abstract void computeImage(java.awt.image.Raster source,
                                     java.awt.image.WritableRaster dest,
                                     java.awt.Rectangle destRect)
```

Calculate the destination image from the source image.

Parameters:

`source` - The source Raster; should be the whole image.
`dest` - The destination WritableRaster; should be the whole image.
`destRect` - The destination Rectangle; should equal the destination image bounds.

---

getTileDependencies

```
public java.awt.Point[] getTileDependencies(int tileX,
                                            int tileY,
                                            int sourceIndex)
```

Returns an array of points indicating the tile dependencies which in this case is the set of all tiles in the source image.

Overrides:

getTileDependencies in class OpImage

javax.media.jai
Class Warp

```
java.lang.Object
  |
  +--javax.media.jai.Warp
```

Direct Known Subclasses:
WarpGrid, WarpPerspective, WarpPolynomial

---

public abstract class **Warp**
extends java.lang.Object
implements java.io.Serializable

A description of an image warp.

The central method of a `Warp` is `warpSparseRect()`, which returns the source pixel positions for a specified (subdivided) rectangular region of the output.

As in the `Interpolation` class, pixel positions are represented using scaled integer coordinates, yielding subpixel accuracy but still allowing the use of integer arithmetic. The degree of precision is set by means of the `getSubSampleBitsH()` and `getSubSampleBitsV` parameters to the `warpRect()` method.

See Also:
`Interpolation`, `WarpAffine`, `WarpGrid`, `WarpPerspective`, `WarpPolynomial`, `WarpQuadratic`, `WarpOpImage`

Constructor Detail

Warp

```
protected Warp()
```

Default constructor.

Method Detail

warpRect

```
public int[] warpRect(int x,
                      int y,
                      int width,
                      int height,
                      int subsampleBitsH,
                      int subsampleBitsV,
                      int[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in fixed point, subpixel coordinates using the `subsampleBitsH` and `subsampleBitsV` parameters.

The integral destination rectangle coordinates should be considered pixel indices. The actual real (non-discrete) plane of pixels locates each pixel index at a half-pixel location. For example, destination pixel (0,0) is located at the real location (0.5, 0.5). Thus pixels are considered to have a dimension of (1.0 x 1.0) with their "energy" concentrated in a "delta function" at relative coordinates (0.5, 0.5).

Destination to source mappings must keep this (0.5, 0.5) pixel center in mind when formulating transformation functions. Given integral destination pixel indices as an input, the fractional source location, as calculated by functions X(xDst,yDst), Y(xDst,yDst) is given by:

```
Xsrc = X(xDst+0.5, yDst+0.5) - 0.5
Ysrc = Y(xDst+0.5, yDst+0.5) - 0.5
```

The subtraction of 0.5 in the above formula produces the source pixel indices (in fractional form) needed to implement the various types of interpolation algorithms.

All of the Sun-supplied warp mapping functions perform the above final subtraction, since they have no knowledge of what interpolation algorithm will be used by a WarpOpImage implementation.

As a convenience, an implementation is provided for this method that calls `warpSparseRect()`. Subclasses may wish to provide their own implementations for better performance.

Parameters:
`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.

`height` - The height of the destination region.
`subsampleBitsH` - The desired fixed-point precision of the output X coordinates.
`subsampleBitsV` - The desired fixed-point precision of the output Y coordinates.
`destRect` - An int array containing at least `2*width*height` elements, or `null`. If `null`, a new array will be constructed.

Returns:
A reference to the destRect parameter if it is non-`null`, or a new `int` array of length `2*width*height` otherwise.

---

warpRect

```
public float[] warpRect(int x,
                        int y,
                        int width,
                        int height,
                        float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in floating point.

As a convenience, an implementation is provided for this method that calls `warpSparseRect()`. Subclasses may wish to provide their own implementations for better performance.

Parameters:
`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`destRect` - A `float` array containing at least `2*width*height` elements, or `null`. If `null`, a new array will be constructed.

Returns:
A reference to the `destRect` parameter if it is non-`null`, or a new `float` array of length `2*width*height` otherwise.

Throws:
java.lang.IllegalArgumentException - if destRect is too small.

---

warpPoint

```
public int[] warpPoint(int x,
                       int y,
                       int subsampleBitsH,
                       int subsampleBitsV,
                       int[] destRect)
```

Computes the source subpixel position for a given destination pixel. The destination pixel is specified using normal integral (full pixel) coordinates. The source position returned by the method is specified in fixed point, subpixel coordinates using the `subsampleBitsH` and `subsampleBitsV` parameters.

As a convenience, an implementation is provided for this method that calls `warpSparseRect()`. Subclasses may wish to provide their own implementations for better performance.

Parameters:
`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`subsampleBitsH` - The desired fixed-point precision of the output X coordinates.
`subsampleBitsV` - The desired fixed-point precision of the output Y coordinates.
`destRect` - An `int` array containing at least 2 elements, or `null`. If `null`, a new array will be constructed.

Returns:
A reference to the destRect parameter if it is non-`null`, or a new `int` array of length 2 otherwise.

Throws:
java.lang.IllegalArgumentException - if destRect is too small.

---

warpPoint

```
public float[] warpPoint(int x,
                         int y,
                         float[] destRect)
```

Computes the source subpixel position for a given destination pixel. The destination pixel is specified using normal integral (full pixel) coordinates. The source position returned by the method is specified in floating point.

As a convenience, an implementation is provided for this method that calls `warpSparseRect()`. Subclasses may wish to provide their own implementations for better performance.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`destRect` - A `float` array containing at least 2 elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `float` array of length 2 otherwise.

Throws:

java.lang.IllegalArgumentException - if destRect is too small.

---

warpSparseRect

```
public int[] warpSparseRect(int x,
                            int y,
                            int width,
                            int height,
                            int periodX,
                            int periodY,
                            int subsampleBitsH,
                            int subsampleBitsV,
                            int[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in fixed point, subpixel coordinates using the `subsampleBitsH` and `subsampleBitsV` parameters.

As a convenience, an implementation is provided for this method that calls `warpSparseRect()` with a `float destRect` parameter. Subclasses may wish to provide their own implementations for better performance.

Parameters:

`x` - the minimum X coordinate of the destination region.
`y` - the minimum Y coordinate of the destination region.
`width` - the width of the destination region.
`height` - the height of the destination region.
`periodX` - the horizontal sampling period.
`periodY` - the horizontal sampling period.
`subsampleBitsH` - The desired fixed-point precision of the output X coordinates.
`subsampleBitsV` - The desired fixed-point precision of the output Y coordinates.
`destRect` - An int array containing at least 2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY) elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `int` array otherwise.

Throws:

java.lang.IllegalArgumentException - if destRect is too small.

---

warpSparseRect

```
public abstract float[] warpSparseRect(int x,
                                       int y,
                                       int width,
                                       int height,
                                       int periodX,
                                       int periodY,
                                       float[] destRect)
```

This method is abstract in this class and must be provided in concrete subclasses.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - A `float` array containing at least `2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY)` elements, or `null`. If `null`, a new array will be constructed.

Returns:

a reference to the `destRect` parameter if it is non-`null`, or a new `float` array otherwise.

---

mapSourceRect

`public java.awt.Rectangle **mapSourceRect**(java.awt.Rectangle sourceRect)`

Computes a rectangle that is guaranteed to enclose the region of the destination that can potentially be affected by the pixels of a rectangle of a given source. Unlike the corresponding `WarpOpImage` method, this routine may return `null` if it is infeasible to compute such a bounding box.

The default implementation in this class returns `null`.

Parameters:

`sourceRect` - The Rectangle in source coordinates.

Returns:

A `Rectangle` in the destination coordinate system that enclose the region that can potentially be affected by the pixels of a rectangle of a given source, or `null`.

---

mapDestRect

`public java.awt.Rectangle **mapDestRect**(java.awt.Rectangle destRect)`

Computes a rectangle that is guaranteed to enclose the region of the source that is required in order to produce a given rectangular output region. Unlike the corresponding `WarpOpImage` method, this routine may return `null` if it is infeasible to compute such a bounding box.

The default implementation in this class returns `null`.

Parameters:

`destRect` - The Rectangle in destination coordinates.

Returns:

A `Rectangle` in the source coordinate system that is guaranteed to contain all pixels referenced by the output of `warpRect()` on the destination region, or `null`.

javax.media.jai
Class WarpAffine

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPolynomial
              |
              +--javax.media.jai.WarpAffine
```

public final class **WarpAffine**
extends WarpPolynomial

A description of an Affine warp.

The transform is specified as a mapping from destination space to source space. In other words, it is the inverse of the normal specification of an affine image transformation.

The source position (x’, y’) of a point (x, y) is given by the quadratic bivariate polynomials:

```
 x’ = p(x, y) = c1 + c2*x + c3*y
 y’ = q(x, y) = c4 + c5*x + c6*y
```

`WarpAffine` is marked final so that it may be more easily inlined.

Field Detail

c1

`private float c1`

c2

`private float c2`

c3

`private float c3`

c4

`private float c4`

c5

`private float c5`

c6

`private float c6`

transform

`private java.awt.geom.AffineTransform transform`

Constructor Detail

WarpAffine

```
public WarpAffine(float[] xCoeffs,
                  float[] yCoeffs,
                  float preScaleX,
                  float preScaleY,
                  float postScaleX,
                  float postScaleY)
```

Constructs a `WarpAffine` with a given transform mapping destination pixels into source space. The transform is given by:

```
 x' = xCoeffs[0] + xCoeffs[1]*x + xCoeffs[2]*y;
 y' = yCoeffs[0] + yCoeffs[1]*x + yCoeffs[2]*y;
```

where `x', y'` are the source image coordinates and `x, y` are the destination image coordinates.

Parameters:

`xCoeffs` - The 3 destination to source transform coefficients for the X coordinate.
`yCoeffs` - The 3 destination to source transform coefficients for the Y coordinate.
`preScaleX` - The scale factor to apply to input (dest) X positions.
`preScaleY` - The scale factor to apply to input (dest) Y positions.
`postScaleX` - The scale factor to apply to the evaluated x transform
`postScaleY` - The scale factor to apply to the evaluated y transform

Throws:

java.lang.IllegalArgumentException - if array `xCoeffs` or `yCoeffs` does not have length of 3.

WarpAffine

```
public WarpAffine(float[] xCoeffs,
                  float[] yCoeffs)
```

Constructs a `WarpAffine` with pre- and post-scale factors of 1.

Parameters:

`xCoeffs` - The 3 destination to source transform coefficients for the X coordinate.
`yCoeffs` - The 3 destination to source transform coefficients for the Y coordinate.

WarpAffine

```
public WarpAffine(java.awt.geom.AffineTransform transform,
                  float preScaleX,
                  float preScaleY,
                  float postScaleX,
                  float postScaleY)
```

Constructs a `WarpAffine` with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of affine mapping of an image.

Parameters:

`transform` - The destination to source transform.
`preScaleX` - The scale factor to apply to source X positions.
`preScaleY` - The scale factor to apply to source Y positions.
`postScaleX` - The scale factor to apply to destination X positions.
`postScaleY` - The scale factor to apply to destination Y positions.

WarpAffine

```
public WarpAffine(java.awt.geom.AffineTransform transform)
```

Constructs a `WarpAffine` with pre- and post-scale factors of 1.

Parameters:

`transform` - An `AffineTransform` mapping dest to source coordinates.

Method Detail

xCoeffsHelper

```
private static final float[] xCoeffsHelper(java.awt.geom.AffineTransform transform)
```

Parameters:

`transform` -

Returns:

An array of `float`s.

---

yCoeffsHelper

```
private static final float[] yCoeffsHelper(java.awt.geom.AffineTransform transform)
```

---

getTransform

```
public java.awt.geom.AffineTransform getTransform()
```

Returns a clone of the `AffineTransform` associated with this `WarpAffine` object.

Returns:

An `AffineTransform`.

---

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in floating point.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - A `float` array containing at least `2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY)` elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `float` array otherwise.

Overrides:

warpSparseRect in class Warp

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect)
```

Computes a Rectangle that is guaranteed to enclose the region of the source that is required in order to produce a given rectangular output region.

Parameters:

`destRect` - The Rectangle in destination coordinates.

Returns:

A `Rectangle` in the source coordinate system that is guaranteed to contain all pixels referenced by the output of `warpRect()` on the destination region, or `null`.

Throws:

NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class WarpPolynomial

---

mapDestPoint

```
private float[] mapDestPoint(int x,
                             int y)
```

javax.media.jai
Class WarpCubic

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPolynomial
              |
              +--javax.media.jai.WarpCubic
```

public final class **WarpCubic**
extends WarpPolynomial

A cubic-based description of an image warp.

The source position (x’, y’) of a point (x, y) is given by the cubic polynomial:

```
 x’ = p(x, y) = c1 + c2*x + c3*y + c4*x^2 + c5*x*y + c6*y^2 +
               c7*x^3 + c8*x^2*y + c9*x*y^2 + c10*y^3
 y’ = q(x, y) = c11 + c12*x + c13*y + c14*x^2 + c15*x*y + c16*y^2 +
               c17*x^3 + c18*x^2*y + c19*x*y^2 + c20*y^3
```

`WarpCubic` is marked final so that it may be more easily inlined.

See Also:
`WarpPolynomial`

Field Detail

c1
`private float c1`

c2
`private float c2`

c3
`private float c3`

c4
`private float c4`

c5
`private float c5`

c6
`private float c6`

c7
`private float c7`

c8
`private float c8`

c9

`private float c9`

---

c10

`private float c10`

---

c11

`private float c11`

---

c12

`private float c12`

---

c13

`private float c13`

---

c14

`private float c14`

---

c15

`private float c15`

---

c16

`private float c16`

---

c17

`private float c17`

---

c18

`private float c18`

---

c19

`private float c19`

---

c20

`private float c20`

Constructor Detail

WarpCubic

```
public WarpCubic(float[] xCoeffs,
                 float[] yCoeffs,
                 float preScaleX,
                 float preScaleY,
                 float postScaleX,
                 float postScaleY)
```

Constructs a `WarpCubic` with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of the mapping of an image. The coeffs arrays must each contain 10 floats corresponding to the coefficients c1, c2, etc. as shown in the class comment.

Parameters:

`xCoeffs` - The 10 destination to source transform coefficients for the X coordinate.
`yCoeffs` - The 10 destination to source transform coefficients for the Y coordinate.
`preScaleX` - The scale factor to apply to input (dest) X positions.
`preScaleY` - The scale factor to apply to input (dest) Y positions.
`postScaleX` - The scale factor to apply to the result of the X polynomial evaluation
`postScaleY` - The scale factor to apply to the result of the Y polynomial evaluation

Throws:

java.lang.IllegalArgumentException - if the length of the xCoeffs and yCoeffs arrays are not both 10.

WarpCubic

```
public WarpCubic(float[] xCoeffs,
                 float[] yCoeffs)
```

Constructs a `WarpCubic` with pre- and post-scale factors of 1.

Parameters:

`xCoeffs` - The 10 destination to source transform coefficients for the X coordinate.
`yCoeffs` - The 10 destination to source transform coefficients for the Y coordinate.

Throws:

java.lang.IllegalArgumentException - if the length of the xCoeffs and yCoeffs arrays are not both 10.

Method Detail

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in floating point.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - A `float` array containing at least `2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY)` elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `float` array otherwise.

Throws:

ArrayBoundsException - if destRect is too small

Overrides:

warpSparseRect in class Warp

javax.media.jai

Class WarpGeneralPolynomial

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPolynomial
              |
              +--javax.media.jai.WarpGeneralPolynomial
```

public final class **WarpGeneralPolynomial**
extends WarpPolynomial

A general polynomial-based description of an image warp.

The mapping is defined by two bivariate polynomial functions X(x, y) and Y(x, y) that define the source X and Y positions that map to a given destination (x, y) pixel coordinate.

The functions X(x, y) and Y(x, y) have the form:

```
 SUM{i = 0 to n} {SUM{j = 0 to i}{a_ij*x^(i - j)*y^j}}
```

See Also:
`WarpPolynomial`

Constructor Detail

WarpGeneralPolynomial

```
public WarpGeneralPolynomial(float[] xCoeffs,
                             float[] yCoeffs,
                             float preScaleX,
                             float preScaleY,
                             float postScaleX,
                             float postScaleY)
```

Constructs a WarpGeneralPolynomial with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of the mapping of an image.

The `xCoeffs` and `yCoeffs` parameters must contain the same number of coefficients of the form `(n + 1)(n + 2)/2` for some `n`, where `n` is the non-negative degree power of the polynomial. The coefficients, in order, are associated with the terms:

```
 1, x, y, x^2, x*y, y^2, ..., x^n, x^(n - 1)*y, ..., x*y^(n - 1), y^n
```

and coefficients of value 0 cannot be omitted.

The destination pixel coordinates (the arguments to the X() and Y() functions) are given in normal integral pixel coordinates, while the output of the functions is given in fixed-point, subpixel coordinates with a number of fractional bits specified by the subsampleBitsH and subsampleBitsV parameters.

Parameters:
`xCoeffs` - The destination to source transform coefficients for the X coordinate.
`yCoeffs` - The destination to source transform coefficients for the Y coordinate.
`preScaleX` - The scale factor to apply to input (dst) X positions.
`preScaleY` - The scale factor to apply to input (dst) Y positions.
`postScaleX` - The scale factor to apply to output (src) X positions.
`postScaleY` - The scale factor to apply to output (src) Y positions.

Throws:
java.lang.IllegalArgumentException - if arrays xCoeffs and yCoeffs do not have the correct number of entries.

WarpGeneralPolynomial

```
public WarpGeneralPolynomial(float[] xCoeffs,
                             float[] yCoeffs)
```

Constructs a WarpGeneralPolynomial with pre- and post-scale factors of 1.

Parameters:
`xCoeffs` - The destination to source transform coefficients for the X coordinate.
`yCoeffs` - The destination to source transform coefficients for the Y coordinate.

Throws:
java.lang.IllegalArgumentException - if arrays xCoeffs and yCoeffs do not have the correct number of entries.

Method Detail

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - An int array containing at least 2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY) elements, or `null`. If `null`, a new array will be constructed.

Returns:

a reference to the destRect parameter if it is non-`null`, or a new int array of length 2*width*height otherwise.

Throws:

ArrayBoundsException - if destRect array is too small

Overrides:

warpSparseRect in class Warp

javax.media.jai
Class WarpGrid

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpGrid
```

public final class **WarpGrid**
extends Warp

A regular grid-based description of an image warp.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

```
 xfrac = (x - x0)/(x1 - x0)
 yfrac = (y - y0)/(y1 - y0)

 s = sx0 + (sx1 - sx0)*xfrac
 t = sy0 + (sy1 - sy0)*xfrac

 u = sx2 + (sx3 - sx2)*xfrac
 v = sy2 + (sy3 - sy2)*xfrac

 sx = s + (u - s)*yfrac
 sy = t + (v - t)*yfrac
```

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:

```
 (x0, y0) ->           (x1, y0) ->
   (sx0, sy0)            (sx1, sy1)
    +------------+---------+
    |            |\        |
    |            | (s, t)  |
    |            |         |
    |            |         |
    |            |         |
    |            |         |
    | (x, y) ->  |         |
    |  (sx, sy)--+         |
    |            |         |
    |            |         |
    |            | (u, v)  |
    |            |/        |
    +------------+---------+
 (x0, y1) ->         (x1, y1) ->
   (sx2, sy2)          (sx3, sy3)
```

WarpGrid is marked final so that it may be more easily inlined.

Field Detail

xStart

`private int xStart`

yStart

`private int yStart`

xEnd

```
private int xEnd
```

---

yEnd

```
private int yEnd
```

---

xStep

```
private int xStep
```

---

yStep

```
private int yStep
```

---

xNumCells

```
private int xNumCells
```

---

yNumCells

```
private int yNumCells
```

---

xWarpPos

```
private float[] xWarpPos
```

---

yWarpPos

```
private float[] yWarpPos
```

Constructor Detail

WarpGrid

```
public WarpGrid(int xStart,
                int xStep,
                int xNumCells,
                int yStart,
                int yStep,
                int yNumCells,
                float[] warpPositions)
```

Constructs a WarpGrid with a given grid-based transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of the mapping of an image.

The grid is defined by a set of equal-sized cells. The grid starts at (xStart, yStart). Each cell has width equal to xStep and height equal to yStep, and there are xNumCells cells horizontally and yNumCells cells vertically.

The degree of warping within each cell is defined by the values in the table parameter. This parameter must contain 2*(xNumCells + 1)*(yNumCells + 1) values, which alternately contain the source X and Y coordinates to which each destination grid intersection point maps. The cells are enumerated in row-major order, that is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

As an example, suppose xNumCells is equal to 2 and yNumCells is equal 1. Then the order of the data in table would be:

```
 x00, y00, x10, y10, x20, y20, x01, y01, x11, y11, x21, y21
```

for a total of 2*(2 + 1)*(1 + 1) = 12 elements.

Parameters:

`xStart` - the minimum X coordinate of the grid.
`xStep` - the horizontal spacing between grid cells.
`xNumCells` - the number of grid cell columns.
`yStart` - the minimum Y coordinate of the grid.
`yStep` - the vertical spacing between grid cells.
`yNumCells` - the number of grid cell rows.
`warpPositions` - a float array of length 2*(xNumCells + 1)* (yNumCells + 1) containing the warp positions at the grid points, in row-major order.

Throws:
java.lang.IllegalArgumentException - if the length of warpPositions is incorrect

---

WarpGrid

```
public WarpGrid(Warp master,
                int xStart,
                int xStep,
                int xNumCells,
                int yStart,
                int yStep,
                int yNumCells)
```

Constructs a WarpGrid object by sampling the displacements given by another Warp object of any kind.

The grid is defined by a set of equal-sized cells. The grid starts at (xStart, yStart). Each cell has width equal to xStep and height equal to yStep, and there are xNumCells cells horizontally and yNumCells cells vertically.

Parameters:
`master` - the Warp object used to initialize the grid displacements.
`xStart` - the minimum X coordinate of the grid.
`xStep` - the horizontal spacing between grid cells.
`xNumCells` - the number of grid cell columns.
`yStart` - the minimum Y coordinate of the grid.
`yStep` - the vertical spacing between grid cells.
`yNumCells` - the number of grid cell rows.

Method Detail

initialize

```
private void initialize(int xStart,
                        int xStep,
                        int xNumCells,
                        int yStart,
                        int yStep,
                        int yNumCells,
                        float[] warpPositions)
```

Parameters:
`xStart` -
`xStep` -
`xNumCells` -
`yStart` -
`yStep` -
`yNumCells` -
`warpPositions` -

---

getXStart

```
public int getXStart()
```

Returns the minimum X coordinate of the grid.

---

getYStart

```
public int getYStart()
```

Returns the minimum Y coordinate of the grid.

---

getXStep

```
public int getXStep()
```

Returns the horizontal spacing between grid cells.

---

getYStep

```
public int getYStep()
```

Returns the horizontal spacing between grid cells.

---

getXNumCells

```
public int getXNumCells()
```

Returns the number of grid cell columns.

---

getYNumCells

```
public int getYNumCells()
```

Returns the number of grid cell columns.

---

getXWarpPos

```
public float[] getXWarpPos()
```

Returns the horizontal warp positions at the grid points.

---

getYWarpPos

```
public float[] getYWarpPos()
```

Returns the horizontal warp positions at the grid points.

---

noWarpSparseRect

```
private float[] noWarpSparseRect(int x1,
                                 int x2,
                                 int y1,
                                 int y2,
                                 int periodX,
                                 int periodY,
                                 int offset,
                                 int stride,
                                 float[] destRect)
```

Copies source to destination, no warpping.

Parameters:

`x1` -
`x2` -
`y1` -
`y2` -
`periodX` -
`periodY` -
`offset` -
`stride` -
`destRect` -

Returns:

An array of `float`s.

Throws:

NullPointerException - if destRect is null
ArrayBoundsException - if destRect is too small

---

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - An int array containing at least 2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY) elements, or `null`. If `null`, a new array will be constructed.

Returns:

a reference to the destRect parameter if it is non-`null`, or a new int array of length 2*width*height otherwise.

Throws:

NullPointerException - if destRect is null
ArrayBoundsException - if destRect is too small

Overrides:

warpSparseRect in class Warp

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect)
```

Computes a Rectangle that is guaranteed to enclose the region of the source that is required in order to produce a given rectangular output region.

Parameters:

`destRect` - The Rectangle in destination coordinates.

Returns:

A `Rectangle` in the source coordinate system that is guaranteed to contain all pixels referenced by the output of `warpRect()` on the destination region, or `null`.

Throws:

NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class Warp

javax.media.jai

Class WarpOpImage

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.OpImage
              |
              +--javax.media.jai.WarpOpImage
```

Direct Known Subclasses:
ScaleOpImage

public abstract class **WarpOpImage**
extends OpImage

A general implementation of image warping, and a superclass for other geometric image operations.

The image warp is specified by a `Warp` object and an `Interpolation` object.

Subclasses of `WarpOpImage` may choose whether they wish to implement the cobbled or non-cobbled variant of `computeRect` by means of the `cobbleSources` constructor parameter. The class comments for `OpImage` provide more information about how to override `computeRect`.

See Also:
`OpImage`, `ScaleOpImage`, `Warp`, `Interpolation`

Field Detail

warp

`protected Warp warp`

The `Warp` object describing the backwards pixel map.

interp

`protected Interpolation interp`

The `Interpolation` object describing the subpixel interpolation method.

writableBounds

`protected java.awt.Rectangle writableBounds`

The writable boundary of this image. By default, this is determined based on the boundary of the source image, the type of the border extender, and the interpolation method. Subclasses should set this variable based on individual cases.

Constructor Detail

WarpOpImage

```
public WarpOpImage(java.awt.image.RenderedImage source,
                   BorderExtender extender,
                   TileCache cache,
                   ImageLayout layout,
                   Warp warp,
                   Interpolation interp,
                   boolean cobbleSources)
```

Constructs a `WarpOpImage`. The output minX, minY, width, and height are derived from the source image unless overridden by the `layout` parameter. The `SampleModel` and `ColorModel` of the output are set in the standard way by the `OpImage` constructor.

Additional control over the image bounds, tile grid layout, `SampleModel`, and `ColorModel` may be obtained by specifying an `ImageLayout` parameter. This parameter will be passed to the superclass constructor unchanged.

Parameters:
`source` - A RenderedImage.
`extender` - A BorderExtender, or null.
`cache` - A TileCache object to store tiles from this OpImage, or null. If null, a default cache will be used.
`layout` - An `ImageLayout` optionally containing the tile grid layout, `SampleModel`, and `ColorModel`.
`warp` - The `Warp` object describing the warp.

`interp` - The `Interpolation` object describing the interpolation method.
`cobbleSources` - A `boolean` indicating whether `computeRect()` expects contiguous sources. To use the default implementation of warping contained in this class, set `cobbleSources` to `false`.

Throws:

java.lang.IllegalArgumentException - if combining the source bounds with the layout parameter results in negative output width or height.

Method Detail

getLeftPadding

```
public int getLeftPadding()
```

Returns the number of samples required to the left of the center.

Returns:

The left padding factor.

---

getRightPadding

```
public int getRightPadding()
```

Returns the number of samples required to the right of the center.

Returns:

The right padding factor.

---

getTopPadding

```
public int getTopPadding()
```

Returns the number of samples required above the center.

Returns:

The top padding factor.

---

getBottomPadding

```
public int getBottomPadding()
```

Returns the number of samples required below the center.

Returns:

The bottom padding factor.

---

mapSourceRect

```
public java.awt.Rectangle mapSourceRect(java.awt.Rectangle sourceRect,
                                        int sourceIndex)
```

Returns a conservative estimate of the destination region that can potentially be affected by the pixels of a rectangle of a given source.

Parameters:

`sourceRect` - The `Rectangle` in source coordinates.
`sourceIndex` - The index of the source image.

Returns:

a `Rectangle` indicating the potentially affected destination region, or `null` if the region is unknown.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `sourceRect` is `null`.

Overrides:

mapSourceRect in class OpImage

---

mapDestRect

```
public java.awt.Rectangle mapDestRect(java.awt.Rectangle destRect,
                                      int sourceIndex)
```

Returns a conservative estimate of the region of a specified source that is required in order to compute the pixels of a given destination rectangle.

Parameters:

`destRect` - The `Rectangle` in destination coordinates.
`sourceIndex` - The index of the source image.

Returns:

a `Rectangle` indicating the required source region.

Throws:

java.lang.IllegalArgumentException - if `sourceIndex` is negative or greater than the index of the last source.
NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class OpImage

---

computeTile

```
public java.awt.image.Raster computeTile(int tileX,
                                         int tileY)
```

Computes a tile. A new `WritableRaster` is created to represent the requested tile. Its width and height equals to this image's tile width and tile height respectively. If the requested tile lies outside of the image's boundary, the created raster is returned with all of its pixels set to 0.

Whether or not this method performs source cobbling is determined by the `cobbleSources` variable set at construction time. If `cobbleSources` is `true`, cobbling is performed on the source for areas that intersect multiple tiles, and `computeRect(Raster[], WritableRaster, Rectangle)` is called to perform the actual computation. Otherwise, `computeRect(PlanarImage[], WritableRaster, Rectangle)` is called to perform the actual computation.

Parameters:

`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

Returns:

The tile as a `Raster`.

Overrides:

computeTile in class OpImage

javax.media.jai
Class WarpPerspective

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPerspective
```

public final class **WarpPerspective**
extends Warp

A description of a perspective (projective) warp.

The transform is specified as a mapping from destination space to source space. In other words, it is the inverse of the normal specification of a perspective image transformation.

WarpPerpsective is marked final so that it may be more easily inlined.

Field Detail

transform

`private PerspectiveTransform transform`

Constructor Detail

WarpPerspective

`public WarpPerspective(PerspectiveTransform transform)`

Constructs a `WarpPerspective` with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of perspective mapping of an image.

Parameters:

`transform` - The destination to source transform.

Throws:

java.lang.IllegalArgumentException - if transform is null

Method Detail

getTransform

`public PerspectiveTransform getTransform()`

Returns a clone of the `PerspectiveTransform` associated with this `WarpPerspective` object.

Returns:

An instance of `PerspectiveTransform`.

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in floating point.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The horizontal sampling period.
`destRect` - A `float` array containing at least `2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY)` elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `float` array otherwise.

Overrides:

warpSparseRect in class Warp

---

mapDestRect

`public java.awt.Rectangle` **`mapDestRect`**`(java.awt.Rectangle destRect)`

Computes a Rectangle that is guaranteed to enclose the region of the source that is required in order to produce a given rectangular output region.

Parameters:

`destRect` - the `Rectangle` in destination coordinates.

Returns:

A `Rectangle` in the source coordinate system that is guaranteed to contain all pixels referenced by the output of `warpRect()` on the destination region.

Overrides:

mapDestRect in class Warp

javax.media.jai
Class WarpPolynomial

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPolynomial
```

Direct Known Subclasses:
WarpAffine, WarpCubic, WarpGeneralPolynomial, WarpQuadratic

public abstract class **WarpPolynomial**
extends Warp

A polynomial-based description of an image warp.

The mapping is defined by two bivariate polynomial functions X(x, y) and Y(x, y) that map destination (x, y) coordinates to source X and Y positions respectively

The functions X(x, y) and Y(x, y) have the form:

```
 SUM{i = 0 to n} {SUM{j = 0 to i}{a_ij*x^(i - j)*y^j}}
```

WarpAffine, WarpQuadratic, and WarpCubic are special cases of WarpPolynomial for n equal to 1, 2, and 3 respectively. WarpGeneralPolynomial provides a concrete implementation for polynomials of higher degree.

See Also:
`WarpAffine`, `WarpQuadratic`, `WarpCubic`, `WarpGeneralPolynomial`

Field Detail

xCoeffs

`protected float[] xCoeffs`

An array of coefficients that maps a destination point to the source's X coordinate.

yCoeffs

`protected float[] yCoeffs`

An array of coefficients that maps a destination point to the source's Y coordinate.

preScaleX

`protected float preScaleX`

A scaling factor applied to input (dest) x coordinates to which may improve computational accuracy.

preScaleY

`protected float preScaleY`

A scaling factor applied to input (dest) y coordinates to which may improve computational accuracy.

postScaleX

`protected float postScaleX`

A scaling factor applied to the result of the X polynomial evaluation which compensates for the input scaling, so that the correctly scaled result is achieved.

postScaleY

`protected float postScaleY`

A scaling factor applied to the result of the Y polynomial evaluation which compensates for the input scaling, so that the correctly scaled result is achieved.

degree

```
protected int degree
```

The degree of the polynomial, determined by the number of coefficients supplied via the X and Y coefficients arrays.

Constructor Detail

WarpPolynomial

```
public WarpPolynomial(float[] xCoeffs,
                      float[] yCoeffs,
                      float preScaleX,
                      float preScaleY,
                      float postScaleX,
                      float postScaleY)
```

Constructs a WarpPolynomial with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of the mapping of an image.

The `xCoeffs` and `yCoeffs` parameters must contain the same number of coefficients of the form `(n + 1)(n + 2)/2` for some `n`, where `n` is the non-negative degree power of the polynomial. The coefficients, in order, are associated with the terms:

```
 1, x, y, x^2, x*y, y^2, ..., x^n, x^(n - 1)*y, ..., x*y^(n - 1), y^n
```

and coefficients of value 0 cannot be omitted.

The source (x, y) coordinate is pre-scaled by the factors preScaleX and preScaleY prior to the evaluation of the polynomial. The result of the polynomial evaluations are scaled by postScaleX and postScaleY to produce the destination pixel coordinates. This process allows for better precision of the results.

Parameters:

`xCoeffs` - The destination to source transform coefficients for the X coordinate.
`yCoeffs` - The destination to source transform coefficients for the Y coordinate.
`preScaleX` - The scale factor to apply to input (dest) X positions.
`preScaleY` - The scale factor to apply to input (dest) Y positions.
`postScaleX` - The scale factor to apply to the X polynomial output.
`postScaleY` - The scale factor to apply to the Y polynomial output.

Throws:

java.lang.IllegalArgumentException - if xCoeff or yCoeff have an illegal number of entries.

WarpPolynomial

```
public WarpPolynomial(float[] xCoeffs,
                      float[] yCoeffs)
```

Constructs a WarpPolynomial with pre- and post-scale factors of 1.

Parameters:

`xCoeffs` - The destination to source transform coefficients for the X coordinate.
`yCoeffs` - The destination to source transform coefficients for the Y coordinate.

Method Detail

getXCoeffs

```
public float[] getXCoeffs()
```

Returns the raw coefficients array for the X coordinate.

Returns:

A cloned array of `float`s giving the polynomial coefficients for the X coordinate.

getYCoeffs

```
public float[] getYCoeffs()
```

Returns the raw coefficients array for the Y coordinate.

Returns:

A cloned array of `float`s giving the polynomial coefficients for the Y coordinate.

getCoeffs

```
public float[][] getCoeffs()
```

Returns the raw coefficients array for both the X and Y coordinates.

Returns:

A cloned two-dimensional array of `float`s giving the polynomial coefficients for the X and Y coordinate.

---

getPreScaleX

```
public float getPreScaleX()
```

Returns the scaling factor applied to input (dest) X coordinates.

---

getPreScaleY

```
public float getPreScaleY()
```

Returns the scaling factor applied to input (dest) Y coordinates.

---

getPostScaleX

```
public float getPostScaleX()
```

Returns the scaling factor applied to the result of the X polynomial.

---

getPostScaleY

```
public float getPostScaleY()
```

Returns the scaling factor applied to the result of the Y polynomial.

---

getDegree

```
public int getDegree()
```

Returns the degree of the warp polynomials.

Returns:

The degree as an `int`.

---

createWarp

```
public static WarpPolynomial createWarp(float[] sourceCoords,
                                        int sourceOffset,
                                        float[] destCoords,
                                        int destOffset,
                                        int numCoords,
                                        float preScaleX,
                                        float preScaleY,
                                        float postScaleX,
                                        float postScaleY,
                                        int degree)
```

Returns an instance of `WarpPolynomial` or its subclasses that approximately maps the given scaled destination image coordinates into the given scaled source image coordinates. The mapping is given by:

```
 x' = postScaleX*(xpoly(x*preScaleX, y*preScaleY));
 x' = postScaleY*(ypoly(x*preScaleX, y*preScaleY));
```

Typically, it is useful to set `preScaleX` to `1.0F/destImage.getWidth()` and `postScaleX` to `srcImage.getWidth()` so that the input and output of the polynomials lie between 0 and 1.

The degree of the polynomial is supplied as an argument.

Parameters:

`sourceCoords` - An array of `float`s containing the source coordinates with X and Y alternating.
`sourceOffset` - the initial entry of `sourceCoords` to be used.
`destCoords` - An array of `float`s containing the destination coordinates with X and Y alternating.
`destOffset` - The initial entry of `destCoords` to be used.
`numCoords` - The number of coordinates from `sourceCoords` and `destCoords` to be used.
`preScaleX` - The scale factor to apply to input (dest) X positions.
`preScaleY` - The scale factor to apply to input (dest) Y positions.
`postScaleX` - The scale factor to apply to X polynomial output.
`postScaleY` - The scale factor to apply to the Y polynomial output.
`degree` - The desired degree of the warp polynomials.

Returns:

An instance of `WarpPolynomial`.

Throws:

java.lang.IllegalArgumentException - if arrays sourceCoords or destCoords are too small

---

mapDestRect

`public java.awt.Rectangle **mapDestRect**(java.awt.Rectangle destRect)`

Computes a Rectangle that is guaranteed to enclose the region of the source that is required in order to produce a given rectangular output region.

Parameters:

`destRect` - The Rectangle in destination coordinates.

Returns:

A `Rectangle` in the source coordinate system that is guaranteed to contain all pixels referenced by the output of `warpRect()` on the destination region, or `null`.

Throws:

NullPointerException - if `destRect` is `null`.

Overrides:

mapDestRect in class Warp

javax.media.jai

Class WarpQuadratic

```
java.lang.Object
  |
  +--javax.media.jai.Warp
        |
        +--javax.media.jai.WarpPolynomial
              |
              +--javax.media.jai.WarpQuadratic
```

public final class **WarpQuadratic**
extends WarpPolynomial

A quadratic-based description of an image warp.

The source position (x', y') of a point (x, y) is given by the quadratic bivariate polynomials:

```
 x' = p(x, y) = c1 + c2*x + c3*y + c4*x^2 + c5*x*y + c6*y^2
 y' = q(x, y) = c7 + c8*x + c9*y + c10*x^2 + c11*x*y + c12*y^2
```

`WarpQuadratic` is marked final so that it may be more easily inlined.

See Also:
`WarpPolynomial`

Field Detail

c1

`private float c1`

c2

`private float c2`

c3

`private float c3`

c4

`private float c4`

c5

`private float c5`

c6

`private float c6`

c7

`private float c7`

c8

`private float c8`

c9

```
private float c9
```

---

c10

```
private float c10
```

---

c11

```
private float c11
```

---

c12

```
private float c12
```

Constructor Detail

WarpQuadratic

```
public WarpQuadratic(float[] xCoeffs,
                     float[] yCoeffs,
                     float preScaleX,
                     float preScaleY,
                     float postScaleX,
                     float postScaleY)
```

Constructs a `WarpQuadratic` with a given transform mapping destination pixels into source space. Note that this is the inverse of the customary specification of the mapping of an image. The coeffs arrays must each contain 6 floats corresponding to the coefficients c1, c2, etc. as shown in the class comment.

Parameters:

`xCoeffs` - The six destination to source transform coefficients for the X coordinate.
`yCoeffs` - The six destination to source transform coefficients for the Y coordinate.
`preScaleX` - The scale factor to apply to input (dest) X positions.
`preScaleY` - The scale factor to apply to input (dest) Y positions.
`postScaleX` - The scale factor to apply to the result of the X polynomial evaluation
`postScaleY` - The scale factor to apply to the result of the Y polynomial evaluation

Throws:

java.lang.IllegalArgumentException - if the xCoeff and yCoeff arrays do not each have size entries.

WarpQuadratic

```
public WarpQuadratic(float[] xCoeffs,
                     float[] yCoeffs)
```

Constructs a `WarpQuadratic` with pre- and post-scale factors of 1.

Parameters:

`xCoeffs` - The 6 destination to source transform coefficients for the X coordinate.
`yCoeffs` - The 6 destination to source transform coefficients for the Y coordinate.

Throws:

java.lang.IllegalArgumentException - if the xCoeff and yCoeff arrays do not each have size entries.

Method Detail

warpSparseRect

```
public float[] warpSparseRect(int x,
                              int y,
                              int width,
                              int height,
                              int periodX,
                              int periodY,
                              float[] destRect)
```

Computes the source subpixel positions for a given rectangular destination region, subsampled with an integral period. The destination region is specified using normal integral (full pixel) coordinates. The source positions returned by the method are specified in floating point.

Parameters:

`x` - The minimum X coordinate of the destination region.
`y` - The minimum Y coordinate of the destination region.
`width` - The width of the destination region.
`height` - The height of the destination region.
`periodX` - The horizontal sampling period.
`periodY` - The vertical sampling period.
`destRect` - A `float` array containing at least `2*((width+periodX-1)/periodX)*((height+periodY-1)/periodY)` elements, or `null`. If `null`, a new array will be constructed.

Returns:

A reference to the `destRect` parameter if it is non-`null`, or a new `float` array otherwise.

Throws:

ArrayBoundsException - if destRect is too small

Overrides:

warpSparseRect in class Warp

javax.media.jai

Class WritableRasterJAI

```
java.lang.Object
  |
  +--java.awt.image.Raster
        |
        +--java.awt.image.WritableRaster
              |
              +--javax.media.jai.WritableRasterJAI
```

class **WritableRasterJAI**
extends java.awt.image.WritableRaster

Constructor Detail

WritableRasterJAI

```
protected WritableRasterJAI(java.awt.image.SampleModel sampleModel,
                            java.awt.image.DataBuffer dataBuffer,
                            java.awt.Rectangle aRegion,
                            java.awt.Point sampleModelTranslate,
                            java.awt.image.WritableRaster parent)
```

javax.media.jai

Class WritableRenderedImageAdapter

```
java.lang.Object
  |
  +--javax.media.jai.PlanarImage
        |
        +--javax.media.jai.RenderedImageAdapter
              |
              +--javax.media.jai.WritableRenderedImageAdapter
```

public final class **WritableRenderedImageAdapter**
extends RenderedImageAdapter
implements java.awt.image.WritableRenderedImage

A `PlanarImage` wrapper for a `WritableRenderedImage`. The tile layout, sample model, and so forth are preserved. Calls to `getTile()` and so forth are forwarded.

From JAI's point of view, this image is a `PlanarImage` of unknown type, with no sources, and additionally an implementer of the `WritableRenderedImage` interface. The image's pixel data appear to be variable.

The class and all its methods are marked `final` in order to allow dynamic inlining to take place. This should eliminate any performance penalty associated with the use of an adapter class.

See Also:
`PlanarImage`, `RenderedImage`, `RenderedImageAdapter`, `WritableRenderedImage`

Field Detail

theImage

`private java.awt.image.WritableRenderedImage` **`theImage`**

The WritableRenderedImage being adapted.

Constructor Detail

WritableRenderedImageAdapter

`public` **`WritableRenderedImageAdapter`**`(java.awt.image.WritableRenderedImage im)`

Constructs a WritableRenderedImageAdapter.

Parameters:
`im` - A WritableRenderedImage to be 'wrapped' as a PlanarImage.

Throws:
java.lang.IllegalArgumentException - if `im` is `null`.

Method Detail

addTileObserver

`public final void` **`addTileObserver`**`(java.awt.image.TileObserver tileObserver)`

Add an observer. If the observer is already present, it will receive multiple notifications.

Specified by:
addTileObserver in interface java.awt.image.WritableRenderedImage

Parameters:
`tileObserver` - The `TileObserver` to be added.

Throws:
java.lang.IllegalArgumentException - if `im` is `null`.

removeTileObserver

`public final void` **`removeTileObserver`**`(java.awt.image.TileObserver tileObserver)`

Remove an observer. If the observer was not registered, nothing happens. If the observer was registered for multiple notifications, it will now be registered for one fewer.

Specified by:
removeTileObserver in interface java.awt.image.WritableRenderedImage

Parameters:
`tileObserver` - The `TileObserver` to be removed.
Throws:
java.lang.IllegalArgumentException - if `im` is `null`.

---

getWritableTile

```
public final java.awt.image.WritableRaster getWritableTile(int tileX,
                                                          int tileY)
```

Check out a tile for writing.

The `WritableRenderedImage` is responsible for notifying all of its `TileObservers` when a tile goes from having no writers to having one writer.

Specified by:
getWritableTile in interface java.awt.image.WritableRenderedImage
Parameters:
`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.
Returns:
The tile as a `WritableRaster`.

---

releaseWritableTile

```
public final void releaseWritableTile(int tileX,
                                      int tileY)
```

Relinquish the right to write to a tile. If the caller continues to write to the tile, the results are undefined. Calls to this method should only appear in matching pairs with calls to `getWritableTile()`; any other use will lead to undefined results.

The `WritableRenderedImage` is responsible for notifying all of its `TileObserver`s when a tile goes from having one writer to having no writers.

Specified by:
releaseWritableTile in interface java.awt.image.WritableRenderedImage
Parameters:
`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.

---

isTileWritable

```
public final boolean isTileWritable(int tileX,
                                    int tileY)
```

Return whether a tile is currently checked out for writing.

Specified by:
isTileWritable in interface java.awt.image.WritableRenderedImage
Parameters:
`tileX` - The X index of the tile.
`tileY` - The Y index of the tile.
Returns:
`true` if the tile currently has writers.

---

getWritableTileIndices

```
public final java.awt.Point[] getWritableTileIndices()
```

Return an array of `Point` objects indicating which tiles are checked out for writing.

Specified by:
getWritableTileIndices in interface java.awt.image.WritableRenderedImage
Returns:
an array of `Point`s.

---

hasTileWriters

```
public final boolean hasTileWriters()
```

Return whether any tile is checked out for writing. Semantically equivalent to (getWritableTiles().size() != 0).

Specified by:
hasTileWriters in interface java.awt.image.WritableRenderedImage

Returns:

`true` if any tile currently has writers.

---

setData

```
public final void setData(java.awt.image.Raster raster)
```

Set a rectangular region of the image to the contents of `raster`.

Specified by:

setData in interface java.awt.image.WritableRenderedImage

Parameters:

`raster` - A `Raster`.

Throws:

java.lang.IllegalArgumentException - if `im` is `null`.

Package javax.media.jai.iterator

Interface Summary	
RandomIter	An iterator that allows random read-only access to any sample within its bounding rectangle.
RectIter	An iterator for traversing a read-only image in top-to-bottom, left-to-right order.
RookIter	An iterator for traversing a read-only image using arbitrary up-down and left-right moves.
WritableRandomIter	An iterator that allows random read/write access to any sample within its bounding rectangle.
WritableRectIter	An iterator for traversing a read/write image in top-to-bottom, left-to-right order.
WritableRookIter	An iterator for traversing a read/write image using arbitrary up-down and left-right moves.

Class Summary	
JaiI18N	
RandomIterFactory	A factory class to instantiate instances of the RandomIter and WritableRandomIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.
RectIterFactory	A factory class to instantiate instances of the RectIter and WritableRectIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.
RookIterFactory	A factory class to instantiate instances of the RookIter and WritableRookIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.

javax.media.jai.iterator
Class JaiI18N

```
java.lang.Object
  |
  +--javax.media.jai.iterator.JaiI18N
```

class **JaiI18N**
extends java.lang.Object

Field Detail

packageName

`static java.lang.String packageName`

Constructor Detail

JaiI18N

`JaiI18N()`

Method Detail

getString

`public static java.lang.String getString(java.lang.String key)`

javax.media.jai.iterator

Interface RandomIter

All Known Subinterfaces:
WritableRandomIter

public abstract interface **RandomIter**

An iterator that allows random read-only access to any sample within its bounding rectangle. This flexibility will generally exact a corresponding price in speed and setup overhead.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself.

The getSample(), getSampleFloat(), and getSampleDouble() methods are provided to allow read-only access to the source data. The getPixel() methods allow retrieval of all bands simultaneously.

An instance of RandomIter may be obtained by means of the RandomIterFactory.create() method, which returns an opaque object implementing this interface.

See Also:
`WritableRandomIter`, `RandomIterFactory`

Method Detail

getSample

```
public int getSample(int x,
                     int y,
                     int b)
```

Returns the specified sample from the image.

Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`b` - the band to retrieve.

getSampleFloat

```
public float getSampleFloat(int x,
                            int y,
                            int b)
```

Returns the specified sample from the image as a float.

Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`b` - the band to retrieve.

getSampleDouble

```
public double getSampleDouble(int x,
                              int y,
                              int b)
```

Returns the specified sample from the image as a double.

Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`b` - the band to retrieve.

getPixel

```
public int[] getPixel(int x,
                      int y,
                      int[] iArray)
```

Returns the samples of the specified pixel from the image in an array of int.

Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`iArray` - An optionally preallocated int array.
Returns:
the contents of the pixel as an int array.

---

getPixel

```
public float[] getPixel(int x,
                        int y,
                        float[] fArray)
```

Returns the samples of the specified pixel from the image in an array of float.
Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`fArray` - An optionally preallocated float array.
Returns:
the contents of the pixel as a float array.

---

getPixel

```
public double[] getPixel(int x,
                         int y,
                         double[] dArray)
```

Returns the samples of the specified pixel from the image in an array of double.
Parameters:
`x` - the X coordinate of the desired pixel.
`y` - the Y coordinate of the desired pixel.
`dArray` - An optionally preallocated double array.
Returns:
the contents of the pixel as a double array.

---

done

```
public void done()
```

Informs the iterator that it may discard its internal data structures. This method should be called when the iterator will no longer be used.

javax.media.jai.iterator

Class RandomIterFactory

```
java.lang.Object
  |
  +--javax.media.jai.iterator.RandomIterFactory
```

public class **RandomIterFactory**
extends java.lang.Object

A factory class to instantiate instances of the RandomIter and WritableRandomIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.

See Also:
`RandomIter`, `WritableRandomIter`

Constructor Detail

RandomIterFactory

```
private RandomIterFactory()
```

Prevent this class from ever being instantiated.

Method Detail

create

```
public static RandomIter create(java.awt.image.RenderedImage im,
                                java.awt.Rectangle bounds)
```

Constructs and returns an instance of RandomIter suitable for iterating over the given bounding rectangle within the given RenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a read-only RenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RandomIter allowing read-only access to the source.

create

```
public static RandomIter create(java.awt.image.Raster ras,
                                java.awt.Rectangle bounds)
```

Constructs and returns an instance of RandomIter suitable for iterating over the given bounding rectangle within the given Raster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:
`ras` - a read-only Raster source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RandomIter allowing read-only access to the source.

createWritable

```
public static WritableRandomIter createWritable(java.awt.image.WritableRenderedImage im,
                                                java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRandomIter suitable for iterating over the given bounding rectangle within the given WritableRenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a WritableRenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a WritableRandomIter allowing read/write access to the source.

createWritable

```
public static WritableRandomIter createWritable(java.awt.image.WritableRaster ras,
                                                java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRandomIter suitable for iterating over the given bounding rectangle within the given WritableRaster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:

`ras` - a WritableRaster source.

`bounds` - the bounding Rectangle for the iterator, or null.

Returns:

a WritableRandomIter allowing read/write access to the source.

javax.media.jai.iterator
Interface RectIter

All Known Subinterfaces:
RookIter, WritableRectIter, WritableRookIter

public abstract interface **RectIter**

An iterator for traversing a read-only image in top-to-bottom, left-to-right order. This will generally be the fastest style of iterator, since it does not need to perform bounds checks against the top or left edges of tiles.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself. Once initialized, the iterator may be reset to its initial state by means of the startLine(), startPixels(), and startBands() methods. Its position may be advanced using the nextLine(), jumpLines(), nextPixel(), jumpPixels(), and nextBand() methods.

The iterator's position may be tested against the bounding rectangle by means of the finishedLines(), finishedPixels(), and finishedBands() methods, as well as the hybrid methods nextLineDone(), nextPixelDone(), and nextBandDone().

The getSample(), getSampleFloat(), and getSampleDouble() methods are provided to allow read-only access to the source data. The various source bands may also be accessed in random fashion using the variants that accept a band index. The getPixel() methods allow retrieval of all bands simultaneously.

An instance of RectIter may be obtained by means of the RectIterFactory.create() method, which returns an opaque object implementing this interface.

See Also:
`WritableRectIter`, `RectIterFactory`

Method Detail

startLines

`public void **startLines**()`

Sets the iterator to the first line of its bounding rectangle. The pixel and band offsets are unchanged.

nextLine

`public void **nextLine**()`

Sets the iterator to the next line of the image. The pixel and band offsets are unchanged. If the iterator passes the bottom line of the rectangles, calls to get() methods are not valid.

nextLineDone

`public boolean **nextLineDone**()`

Sets the iterator to the next line in the image, and returns true if the bottom row of the bounding rectangle has been passed.

jumpLines

`public void **jumpLines**(int num)`

Jumps downward num lines from the current position. Num may be negative. The pixel and band offsets are unchanged. If the position after the jump is outside of the iterator's bounding box, an `IndexOutOfBoundsException` will be thrown and the position will be unchanged.

Throws:
java.lang.IndexOutOfBoundsException - if the position goes outside of the iterator's bounding box.

finishedLines

`public boolean **finishedLines**()`

Returns true if the bottom row of the bounding rectangle has been passed.

startPixels

`public void **startPixels**()`

Sets the iterator to the leftmost pixel of its bounding rectangle. The line and band offsets are unchanged.

---

nextPixel

`public void **nextPixel**()`

Sets the iterator to the next pixel in image (that is, move rightward). The line and band offsets are unchanged.

---

nextPixelDone

`public boolean **nextPixelDone**()`

Sets the iterator to the next pixel in the image (that is, move rightward). Returns true if the right edge of the bounding rectangle has been passed. The line and band offsets are unchanged.

---

jumpPixels

`public void **jumpPixels**(int num)`

Jumps rightward num pixels from the current position. Num may be negative. The line and band offsets are unchanged. If the position after the jump is outside of the iterator's bounding box, an `IndexOutOfBoundsException` will be thrown and the position will be unchanged.

Throws:

java.lang.IndexOutOfBoundsException - if the position goes outside of the iterator's bounding box.

---

finishedPixels

`public boolean **finishedPixels**()`

Returns true if the right edge of the bounding rectangle has been passed.

---

startBands

`public void **startBands**()`

Sets the iterator to the first band of the image. The pixel column and line are unchanged.

---

nextBand

`public void **nextBand**()`

Sets the iterator to the next band in the image. The pixel column and line are unchanged.

---

nextBandDone

`public boolean **nextBandDone**()`

Sets the iterator to the next band in the image, and returns true if the max band has been exceeded. The pixel column and line are unchanged.

---

finishedBands

`public boolean **finishedBands**()`

Returns true if the max band in the image has been exceeded.

---

getSample

`public int **getSample**()`

Returns the current sample as an integer.

---

getSample

`public int **getSample**(int b)`

Returns the specified sample of the current pixel as an integer.

Parameters:

`b` - the band index of the desired sample.

---

getSampleFloat

`public float **getSampleFloat**()`

Returns the current sample as a float.

---

getSampleFloat

`public float **getSampleFloat**(int b)`

Returns the specified sample of the current pixel as a float.

Parameters:

`b` - the band index of the desired sample.

---

getSampleDouble

`public double **getSampleDouble**()`

Returns the current sample as a double.

---

getSampleDouble

`public double **getSampleDouble**(int b)`

Returns the specified sample of the current pixel as a double.

Parameters:

`b` - the band index of the desired sample.

---

getPixel

`public int[] **getPixel**(int[] iArray)`

Returns the samples of the current pixel from the image in an array of int.

Parameters:

`iArray` - An optionally preallocated int array.

Returns:

the contents of the pixel as an int array.

---

getPixel

`public float[] **getPixel**(float[] fArray)`

Returns the samples of the current pixel from the image in an array of float.

Parameters:

`fArray` - An optionally preallocated float array.

Returns:

the contents of the pixel as a float array.

---

getPixel

`public double[] **getPixel**(double[] dArray)`

Returns the samples of the current pixel from the image in an array of double.

Parameters:

`dArray` - An optionally preallocated double array.

Returns:

the contents of the pixel as a double array.

javax.media.jai.iterator

Class RectIterFactory

```
java.lang.Object
  |
  +--javax.media.jai.iterator.RectIterFactory
```

---

public class **RectIterFactory**
extends java.lang.Object

A factory class to instantiate instances of the RectIter and WritableRectIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.

See Also:
`RectIter`, `WritableRectIter`

Constructor Detail

RectIterFactory

```
private RectIterFactory()
```

Prevent this class from ever being instantiated.

Method Detail

create

```
public static RectIter create(java.awt.image.RenderedImage im,
                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of RectIter suitable for iterating over the given bounding rectangle within the given RenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a read-only RenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RectIter allowing read-only access to the source.

---

create

```
public static RectIter create(java.awt.image.Raster ras,
                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of RectIter suitable for iterating over the given bounding rectangle within the given Raster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:
`ras` - a read-only Raster source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RectIter allowing read-only access to the source.

---

createWritable

```
public static WritableRectIter createWritable(java.awt.image.WritableRenderedImage im,
                                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRectIter suitable for iterating over the given bounding rectangle within the given WritableRenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a WritableRenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a WritableRectIter allowing read/write access to the source.

---

createWritable

```
public static WritableRectIter createWritable(java.awt.image.WritableRaster ras,
                                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRectIter suitable for iterating over the given bounding rectangle within the given WritableRaster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:

`ras` - a WritableRaster source.

`bounds` - the bounding Rectangle for the iterator, or null.

Returns:

a WritableRectIter allowing read/write access to the source.

javax.media.jai.iterator
Interface RookIter

All Known Subinterfaces:
WritableRookIter

public abstract interface **RookIter**
extends RectIter

An iterator for traversing a read-only image using arbitrary up-down and left-right moves. This will generally be somewhat slower than a corresponding instance of RectIter, since it must perform bounds checks against the top and left edges of tiles in addition to their bottom and right edges.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself. Once initialized, the iterator may be reset to its initial state by means of the startLine(), startPixels(), and startBands() methods. As with RectIter, its position may be advanced using the nextLine(), jumpLines(), nextPixel(), jumpPixels(), and nextBand() methods.

In addition, prevLine(), prevPixel(), and prevBand() methods exist to move in the upwards and leftwards directions and to access smaller band indices. The iterator may be set to the far edges of the bounding rectangle by means of the endLines(), endPixels(), and endBands() methods.

The iterator's position may be tested against the bounding rectangle by means of the finishedLines(), finishedPixels(), and finishedBands() methods, as well as the hybrid methods nextLineDone(), prevLineDone(), nextPixelDone(), prevPixelDone(), nextBandDone(), and prevBandDone().

The getSample(), getSampleFloat(), and getSampleDouble() methods are provided to allow read-only access to the source data. The various source bands may also be accessed in random fashion using the variants that accept a band index. The getPixel() methods allow retrieval of all bands simultaneously.

An instance of RookIter may be obtained by means of the RookIterFactory.create() method, which returns an opaque object implementing this interface.

See Also:
`RectIter`, `RookIterFactory`

Method Detail

prevLine

`public void` **`prevLine`**`()`

Sets the iterator to the previous line of the image. The pixel and band offsets are unchanged. If the iterator passes the top line of the rectangle, calls to get() methods are not valid.

prevLineDone

`public boolean` **`prevLineDone`**`()`

Sets the iterator to the previous line in the image, and returns true if the top row of the bounding rectangle has been passed.

endLines

`public void` **`endLines`**`()`

Sets the iterator to the last line of its bounding rectangle. The pixel and band offsets are unchanged.

prevPixel

`public void` **`prevPixel`**`()`

Sets the iterator to the previous pixel in the image (that is, move leftward). The line and band offsets are unchanged.

prevPixelDone

`public boolean` **`prevPixelDone`**`()`

Sets the iterator to the previous pixel in the image (that is, move leftward). Returns true if the left edge of the bounding rectangle has been passed. The line and band offsets are unchanged.

endPixels

```
public void endPixels()
```

Sets the iterator to the rightmost pixel of its bounding rectangle. The line and band offsets are unchanged.

prevBand

```
public void prevBand()
```

Sets the iterator to the previous band in the image. The pixel column and line are unchanged.

prevBandDone

```
public boolean prevBandDone()
```

Sets the iterator to the previous band in the image, and returns true if the min band has been exceeded. The pixel column and line are unchanged.

endBands

```
public void endBands()
```

Sets the iterator to the last band of the image. The pixel column and line are unchanged.

javax.media.jai.iterator
Class RookIterFactory

```
java.lang.Object
  |
  +--javax.media.jai.iterator.RookIterFactory
```

public class **RookIterFactory**
extends java.lang.Object

A factory class to instantiate instances of the RookIter and WritableRookIter interfaces on sources of type Raster, RenderedImage, and WritableRenderedImage.

See Also:
`RookIter`, `WritableRookIter`

Constructor Detail

RookIterFactory

```
private RookIterFactory()
```

Prevent this class from ever being instantiated.

Method Detail

create

```
public static RookIter create(java.awt.image.RenderedImage im,
                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of RookIter suitable for iterating over the given bounding rectangle within the given RenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a read-only RenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RookIter allowing read-only access to the source.

create

```
public static RookIter create(java.awt.image.Raster ras,
                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of RookIter suitable for iterating over the given bounding rectangle within the given Raster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:
`ras` - a read-only Raster source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a RookIter allowing read-only access to the source.

createWritable

```
public static WritableRookIter createWritable(java.awt.image.WritableRenderedImage im,
                                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRookIter suitable for iterating over the given bounding rectangle within the given WritableRenderedImage source. If the bounds parameter is null, the entire image will be used.

Parameters:
`im` - a WritableRenderedImage source.
`bounds` - the bounding Rectangle for the iterator, or null.

Returns:
a WritableRookIter allowing read/write access to the source.

createWritable

```
public static WritableRookIter createWritable(java.awt.image.WritableRaster ras,
                                              java.awt.Rectangle bounds)
```

Constructs and returns an instance of WritableRookIter suitable for iterating over the given bounding rectangle within the given WritableRaster source. If the bounds parameter is null, the entire Raster will be used.

Parameters:

`ras` - a WritableRaster source.

`bounds` - the bounding Rectangle for the iterator, or null.

Returns:

a WritableRookIter allowing read/write access to the source.

javax.media.jai.iterator
Interface WritableRandomIter

public abstract interface **WritableRandomIter**
extends RandomIter

An iterator that allows random read/write access to any sample within its bounding rectangle. This flexibility will generally exact a corresponding price in speed and setup overhead.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself.

The setSample() and setPixel() methods allow individual source samples and whole pixels to be written.

An instance of RandomIter may be obtained by means of the RandomIterFactory.createWritable() method, which returns an opaque object implementing this interface.

See Also:
`RandomIter`, `RandomIterFactory`

Method Detail

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      int s)
```

Sets the specified sample of the image to an integral value.

Parameters:
`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`b` - the band to be set.
`s` - the sample’s new integral value.

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      float s)
```

Sets the specified sample of the image to a float value.

Parameters:
`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`b` - the band to be set.
`s` - the sample’s new float value.

setSample

```
public void setSample(int x,
                      int y,
                      int b,
                      double s)
```

Sets the specified sample of the image to a double value.

Parameters:
`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`b` - the band to be set.
`s` - the sample’s new double value.

setPixel

```
public void setPixel(int x,
                     int y,
                     int[] iArray)
```

Sets a pixel in the image using an int array of samples for input.

Parameters:

`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`iArray` - the input samples in an int array.

---

setPixel

```
public void setPixel(int x,
                     int y,
                     float[] fArray)
```

Sets a pixel in the image using a float array of samples for input.

Parameters:

`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`iArray` - the input samples in a float array.

---

setPixel

```
public void setPixel(int x,
                     int y,
                     double[] dArray)
```

Sets a pixel in the image using a float array of samples for input.

Parameters:

`x` - the X coordinate of the pixel.
`y` - the Y coordinate of the pixel.
`dArray` - the input samples in a double array.

javax.media.jai.iterator
Interface WritableRectIter

All Known Subinterfaces:
WritableRookIter

public abstract interface **WritableRectIter**
extends RectIter

An iterator for traversing a read/write image in top-to-bottom, left-to-right order. This will generally be the fastest style of iterator, since it does not need to perform bounds checks against the top or left edges of tiles.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself. Once initialized, the iterator may be reset to its initial state by means of the startLine(), startPixels(), and startBands() methods. Its position may be advanced using the nextLine(), jumpLines(), nextPixel(), jumpPixels(), and nextBand() methods.

The iterator's position may be tested against the bounding rectangle by means of the finishedLines(), finishedPixels(), and finishedBands() methods, as well as the hybrid methods nextLineDone(), nextPixelDone(), and nextBandDone().

The getSample(), getSampleFloat(), and getSampleDouble() methods are provided to allow read-only access to the source data. The various source bands may also be accessed in random fashion using the variants that accept a band index. The getPixel() methods allow retrieval of all bands simultaneously.

WritableRookIter adds the ability to alter the source pixel values using the various setSample() and setPixel() methods.

An instance of WritableRectIter may be obtained by means of the RectIterFactory.createWritable() method, which returns an opaque object implementing this interface.

See Also:
`RectIter`, `RectIterFactory`

Method Detail

setSample

```
public void setSample(int s)
```

Sets the current sample to an integral value.

setSample

```
public void setSample(int b,
                      int s)
```

Sets the specified sample of the current pixel to an integral value.

setSample

```
public void setSample(float s)
```

Sets the current sample to a float value.

setSample

```
public void setSample(int b,
                      float s)
```

Sets the specified sample of the current pixel to a float value.

setSample

```
public void setSample(double x)
```

Sets the current sample to a double value.

setSample

```
public void setSample(int b,
                      double s)
```

Sets the specified sample of the current pixel to a double value.

---

setPixel

```
public void setPixel(int[] iArray)
```

Sets all samples of the current pixel to a set of int values.
Parameters:
`iArray` - an int array containing a value for each band.

---

setPixel

```
public void setPixel(float[] fArray)
```

Sets all samples of the current pixel to a set of float values.
Parameters:
`fArray` - a float array containing a value for each band.

---

setPixel

```
public void setPixel(double[] dArray)
```

Sets all samples of the current pixel to a set of double values.
Parameters:
`dArray` - a double array containing a value for each band.

javax.media.jai.iterator
Interface WritableRookIter

public abstract interface **WritableRookIter**
extends RookIter, WritableRectIter

An iterator for traversing a read/write image using arbitrary up-down and left-right moves. This will generally be somewhat slower than a corresponding instance of RectIter, since it must perform bounds checks against the top and left edges of tiles in addition to their botton and right edges.

The iterator is initialized with a particular rectangle as its bounds, which it is illegal to exceed. This initialization takes place in a factory method and is not a part of the iterator interface itself. Once initialized, the iterator may be reset to its initial state by means of the startLine(), startPixels(), and startBands() methods. As with RectIter, its position may be advanced using the nextLine(), jumpLines(), nextPixel(), jumpPixels(), and nextBand() methods.

In addition, prevLine(), prevPixel(), and prevBand() methods exist to move in the upwards and leftwards directions and to access smaller band indices. The iterator may be set to the far edges of the bounding rectangle by means of the endLines(), endPixels(), and endBands() methods.

The iterator's position may be tested against the bounding rectangle by means of the finishedLines(), finishedPixels(), and finishedBands() methods, as well as the hybrid methods nextLineDone(), prevLineDone(), nextPixelDone(), prevPixelDone(), nextBandDone(), and prevBandDone().

The getSample(), getSampleFloat(), and getSampleDouble() methods are provided to allow read-only access to the source data. The various source bands may also be accessed in random fashion using the variants that accept a band index. The getPixel() methods allow retrieval of all bands simultaneously.

WritableRookIter adds the ability to alter the source pixel values using the various setSample() and setPixel() methods. These methods are inherited from the WritableRectIter interface unchanged.

An instance of WritableRookIter may be obtained by means of the RookIterFactory.createWritable() method, which returns an opaque object implementing this interface.

Note that a WritableRookIter inherits multiply from RookIter and WritableRectIter, and so may be passed into code expecting either interface. WritableRookIter in fact adds no methods not found in one of its parent interfaces.

See Also:
`RookIter`, `WritableRectIter`, `RookIterFactory`

Package javax.media.jai.operator

Class Summary	
AbsoluteDescriptor	An `OperationDescriptor` describing the "Absolute" operation.
AddCollectionDescriptor	An `OperationDescriptor` describing the "AddCollection" operation.
AddConstDescriptor	An `OperationDescriptor` describing the "AddConst" operation.
AddConstToCollectionDescriptor	An `OperationDescriptor` describing the "AddConstToCollection" operation.
AddDescriptor	An `OperationDescriptor` describing the "Add" operation.
AffineDescriptor	An `OperationDescriptor` describing the "Affine" operation.
AffinePropertyGenerator	This property generator computes the properties for the operation "Affine" dynamically.
AndConstDescriptor	An `OperationDescriptor` describing the "AndConst" operation.
AndDescriptor	An `OperationDescriptor` describing the "And" operation.
AWTImageDescriptor	An `OperationDescriptor` describing the "AWTImage" operation.
BandCombineDescriptor	An `OperationDescriptor` describing the "BandCombine" operation.
BandSelectDescriptor	An `OperationDescriptor` describing the "BandSelect" operation.
BMPDescriptor	An `OperationDescriptor` describing the "BMP" operation.
BorderDescriptor	An `OperationDescriptor` describing the "Border" operation.
BoxFilterDescriptor	An `OperationDescriptor` describing the "BoxFilter" operation.
ClampDescriptor	An `OperationDescriptor` describing the "Clamp" operation.
ColorConvertDescriptor	An `OperationDescriptor` describing the "ColorConvert" operation.
CompositeDescriptor	An `OperationDescriptor` describing the "Composite" operation.
ConjugateDescriptor	An `OperationDescriptor` describing the "Conjugate" operation.
ConjugatePropertyGenerator	This property generator computes the properties for the operation "Conjugate" dynamically.
ConstantDescriptor	An `OperationDescriptor` describing the "Constant" operation.
ConvolveDescriptor	An `OperationDescriptor` describing the "Convolve" operation.
CropDescriptor	An `OperationDescriptor` describing the "Crop" operation.
DCTDescriptor	An `OperationDescriptor` describing the "DCT" operation.
DFTDescriptor	An `OperationDescriptor` describing the "DFT" operation.
DFTPropertyGenerator	This property generator computes the properties for the operation "DFT" dynamically.
DivideByConstDescriptor	An `OperationDescriptor` describing the "DivideByConst" operation.
DivideComplexDescriptor	An `OperationDescriptor` describing the "DivideComplex" operation.
DivideComplexPropertyGenerator	This property generator computes the properties for the operation "DivideComplex" dynamically.
DivideDescriptor	An `OperationDescriptor` describing the "Divide" operation.

DivideIntoConstDescriptor	An `OperationDescriptor` describing the "DivideIntoConst" operation.
EncodeDescriptor	An `OperationDescriptor` describing the "Encode" operation.
ErrorDiffusionDescriptor	An `OperationDescriptor` describing the "ErrorDiffusion" operation.
ExpDescriptor	An `OperationDescriptor` describing the "Exp" operation.
ExtremaDescriptor	An `OperationDescriptor` describing the "Extrema" operation.
FileLoadDescriptor	An `OperationDescriptor` describing the "FileLoad" operation.
FileStoreDescriptor	An `OperationDescriptor` describing the "FileStore" operation.
FormatDescriptor	An `OperationDescriptor` describing the "Format" operation.
FPXDescriptor	An `OperationDescriptor` describing the "FPX" operation.
GIFDescriptor	An `OperationDescriptor` describing the "GIF" operation.
GradientMagnitudeDescriptor	An `OperationDescriptor` describing the "GradientMagnitude" operation.
HistogramDescriptor	An `OperationDescriptor` describing the "Histogram" operation.
IDCTDescriptor	An `OperationDescriptor` describing the "IDCT" operation.
IDFTDescriptor	An `OperationDescriptor` describing the "IDFT" operation.
IDFTPropertyGenerator	This property generator computes the properties for the operation "IDFT" dynamically.
IIPDescriptor	An `OperationDescriptor` describing the "IIP" operation.
IIPResolutionDescriptor	An `OperationDescriptor` describing the "IIPResolution" operation.
ImageFunctionDescriptor	An `OperationDescriptor` describing the "ImageFunction" operation.
ImageFunctionPropertyGenerator	This property generator computes the properties for the operation "ImageFunction" dynamically.
InvertDescriptor	An `OperationDescriptor` describing the "Invert" operation.
JaiI18N	
JPEGDescriptor	An `OperationDescriptor` describing the "JPEG" operation.
LogDescriptor	An `OperationDescriptor` describing the "Log" operation.
LookupDescriptor	An `OperationDescriptor` describing the "Lookup" operation.
MagnitudeDescriptor	An `OperationDescriptor` describing the "Magnitude" operation.
MagnitudePropertyGenerator	This property generator computes the properties for the operation "Magnitude" dynamically.
MagnitudeSquaredDescriptor	An `OperationDescriptor` describing the "MagnitudeSquared" operation.
MagnitudeSquaredPropertyGenerator	This property generator computes the properties for the operation "MagnitudeSquared" dynamically.
MatchCDFDescriptor	An `OperationDescriptor` describing the "MatchCDF" operation.
MaxDescriptor	An `OperationDescriptor` describing the "Max" operation.
MeanDescriptor	An `OperationDescriptor` describing the "Mean" operation.
MedianFilterDescriptor	An `OperationDescriptor` describing the "MedianFilter" operation.
MinDescriptor	An `OperationDescriptor` describing the "Min" operation.

MultiplyComplexDescriptor	An `OperationDescriptor` describing the "MultiplyComplex" operation.
MultiplyComplexPropertyGenerator	This property generator computes the properties for the operation "MultiplyComplex" dynamically.
MultiplyConstDescriptor	An `OperationDescriptor` describing the "MultiplyConst" operation.
MultiplyDescriptor	An `OperationDescriptor` describing the "Multiply" operation.
NotDescriptor	An `OperationDescriptor` describing the "Not" operation.
OrConstDescriptor	An `OperationDescriptor` describing the "OrConst" operation.
OrderedDitherDescriptor	An `OperationDescriptor` describing the "OrderedDither" operation.
OrDescriptor	An `OperationDescriptor` describing the "Or" operation.
OverlayDescriptor	An `OperationDescriptor` describing the "Overlay" operation.
PatternDescriptor	An `OperationDescriptor` describing the "Pattern" operation.
PeriodicShiftDescriptor	An `OperationDescriptor` describing the "PeriodicShift" operation.
PhaseDescriptor	An `OperationDescriptor` describing the "Phase" operation.
PhasePropertyGenerator	This property generator computes the properties for the operation "Phase" dynamically.
PiecewiseDescriptor	An `OperationDescriptor` describing the "Piecewise" operation.
PNGDescriptor	An `OperationDescriptor` describing the "PNG" operation.
PNMDescriptor	An `OperationDescriptor` describing the "PNM" operation.
PolarToComplexDescriptor	An `OperationDescriptor` describing the "PolarToComplex" operation.
PolarToComplexPropertyGenerator	This property generator computes the properties for the operation "PolarToComplex" dynamically.
RenderableDescriptor	An `OperationDescriptor` describing the "Renderable" operation.
RescaleDescriptor	An `OperationDescriptor` describing the "Rescale" operation.
RotateDescriptor	An `OperationDescriptor` describing the "Rotate" operation.
RotatePropertyGenerator	This property generator computes the properties for the operation "Rotate" dynamically.
ScaleDescriptor	An `OperationDescriptor` describing the "Scale" operation.
ScalePropertyGenerator	This property generator computes the properties for the operation "Scale" dynamically.
ShearDescriptor	An `OperationDescriptor` describing the "Shear" operation.
ShearPropertyGenerator	This property generator computes the properties for the operation "Shear" dynamically.
StreamDescriptor	An `OperationDescriptor` describing the "Stream" operation.
SubtractConstDescriptor	An `OperationDescriptor` describing the "SubtractConst" operation.
SubtractDescriptor	An `OperationDescriptor` describing the "Subtract" operation.
SubtractFromConstDescriptor	An `OperationDescriptor` describing the "SubtractFromConst" operation.
ThresholdDescriptor	An `OperationDescriptor` describing the "Threshold" operation.
TIFFDescriptor	An `OperationDescriptor` describing the "TIFF" operation.

TranslateDescriptor	An `OperationDescriptor` describing the "Translate" operation.
TranslatePropertyGenerator	This property generator computes the properties for the operation "Translate" dynamically.
TransposeDescriptor	An `OperationDescriptor` describing the "Transpose" operation.
TransposePropertyGenerator	This property generator computes the properties for the operation "Transpose" dynamically.
URLDescriptor	An `OperationDescriptor` describing the "URL" operation.
WarpDescriptor	An `OperationDescriptor` describing the "Warp" operation.
WarpPropertyGenerator	This property generator computes the properties for the operation "Warp" dynamically.
XorConstDescriptor	An `OperationDescriptor` describing the "XorConst" operation.
XorDescriptor	An `OperationDescriptor` describing the "Xor" operation.

javax.media.jai.operator
Class AWTImageDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AWTImageDescriptor
```

public class **AWTImageDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "AWTImage" operation.

The AWTImage operation converts a standard `java.awt.Image` into a rendered image. By default, the width and height of the image are the same as the original AWT image. The sample model and color model are set according to the AWT image data.

Resource List

Name	Value
GlobalName	AWTImage
LocalName	AWTImage
Vendor	com.sun.media.jai
Description	Converts a `java.awt.Image` into a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AWTImageDescriptor.html
Version	1.0
arg0Desc	The AWT image to be converted.

Parameter List

Name	Class Type	Default Value
awtImage	java.awt.Image	NO_PARAMETER_DEFAULT

See Also:
`Image`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

AWTImageDescriptor

```
public AWTImageDescriptor()
```

Constructor.

javax.media.jai.operator
Class AbsoluteDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AbsoluteDescriptor
```

public class **AbsoluteDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Absolute" operation.

The "Absolute" operation takes a single rendered or renderable source image, and computes the mathematical absolute value of each pixel:

```
 if (src[x][y][b] < 0) {
     dst[x][y][b] = -src[x][y][b];
 } else {
     dst[x][y][b] =  src[x][y][b];
 }
```

For signed integral data types, the smallest value of the data type does not have a positive counterpart; such values will be left unchanged. This behavior parallels that of the Java unary minus operator (see *The Java Language Specification*, section 15.14.4).

Resource List

Name	Value
GlobalName	Absolute
LocalName	Absolute
Vendor	com.sun.media.jai
Description	Replaces the pixel values of an image by their absolute values.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AbsoluteDescriptor.html
Version	1.0

No parameters are needed for the "Absolute" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

AbsoluteDescriptor

`public AbsoluteDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class AddCollectionDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AddCollectionDescriptor
```

public class **AddCollectionDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "AddCollection" operation.

The AddCollection operation takes a collection of rendered source images, and adds every set of pixels, one from each source image of the corresponding position and band. No additional parameters are required.

There is no restriction on the actual class type used to represent the source collection, but each element of the collection must be an instance of `RenderedImage`. The number of images in the collection may vary from 2 to n. The source images may have different numbers of bands and data types.

By default, the destination image bounds are the intersection of all of the source image bounds. If any of the two sources are completely disjoint, the destination will have a width and a height of 0. The number of bands of the destination image is equal to the minimum number of bands of all the sources, and the data type is the biggest data type of all the sources. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
dst[x][y][b] = 0;
for (int i = 0; i < numSources; i++) {
    dst[x][y][b] += srcs[i][x][y][b];
}
```

Resource List

Name	Value
GlobalName	AddCollection
LocalName	AddCollection
Vendor	com.sun.media.jai
Description	Adds a collection of rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AddCollectionDescriptor.html
Version	1.0

See Also:
`RenderedImage`, `Collection`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

sourceClasses

`private static final java.lang.Class[] sourceClasses`

The source class list for this operation.

Constructor Detail

AddCollectionDescriptor

```
public AddCollectionDescriptor()
```

Constructor.

Method Detail

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates input source collection.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator

Class AddConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AddConstDescriptor
```

public class **AddConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "AddConst" operation.

The AddConst operation takes one rendered or renderable source image and an array of double constants, and adds a constant to every pixel of its corresponding band of the source. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

By default, the destination image bound, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
 if (constants.length < dstNumBands) {
     dst[x][y][b] = src[x][y][b] + constants[0];
 } else {
     dst[x][y][b] = src[x][y][b] + constants[b];
 }
```

Resource List

Name	Value
GlobalName	AddConst
LocalName	AddConst
Vendor	com.sun.media.jai
Description	Adds constants to a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AddConstDescriptor.html
Version	1.0
arg0Desc	The constants to be added.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

AddConstDescriptor

`public **AddConstDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameter.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class AddConstToCollectionDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AddConstToCollectionDescriptor
```

public class **AddConstToCollectionDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "AddConstToCollection" operation.

The AddConstToCollection operation takes a collection of rendered images and an array of double constants, and for each rendered image in the collection adds a constant to every pixel of its corresponding band. If the number of constants supplied is less than the number of bands of a source image then the same constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

The operation will attempt to store the result images in the same collection class as that of the source images. If a new instance of the source collection class can not be created, then the operation will store the result images in a java.util.Vector. There will be the same number of images in the output collection as in the source collection.

Resource List

Name	Value
GlobalName	AddConstToCollection
LocalName	AddConstToCollection
Vendor	com.sun.media.jai
Description	Adds constants to a collection of rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AddConstToCollectionDescriptor.html
Version	1.0
arg0Desc	The constants to be added.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`CollectionImage`, `Collection`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

sourceClasses

`private static final java.lang.Class[] `**`sourceClasses`**

The source class list for this operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

AddConstToCollectionDescriptor

```
public AddConstToCollectionDescriptor()
```

Constructor.

Method Detail

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates input source and parameter.

Overrides:

validateArguments in class OperationDescriptorImpl

---

getDestClass

```
public java.lang.Class getDestClass()
```

Returns the destination's class type of this operation.

Overrides:

getDestClass in class OperationDescriptorImpl

javax.media.jai.operator
Class AddDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AddDescriptor
```

public class **AddDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Add" operation.

The Add operation takes two rendered or renderable source images, and adds every pair of pixels, one from each source image of the corresponding position and band. No additional parameters are required.

The two source images may have different numbers of bands and data types. By default, the destination image bounds are the intersection of the two source image bounds. If the sources don’t intersect, the destination will have a width and height of 0.

The default number of bands of the destination image is equal to the smallest number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types (not necessarily the range of their sums).

As a special case, if one of the source images has N bands (N > 1), the other source has 1 band, and an `ImageLayout` hint is provided containing a destination `SampleModel` with K bands (1 < K <= N), then the single band of the 1-banded source is added to each of the first K bands of the N-band source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
dst[x][y][dstBand] = clamp(srcs[0][x][y][src0Band] +
                           srcs[1][x][y][src1Band]);
```

Resource List

Name	Value
GlobalName	Add
LocalName	Add
Vendor	com.sun.media.jai
Description	Adds two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AddDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

AddDescriptor

`public AddDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator

Class AffineDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AffineDescriptor
```

public class **AffineDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Affine" operation.

The Affine operation performs (possibly filtered) affine mapping on a rendered or renderable source image.

The relationship between the source and the destination pixels is defined as follows. For each pixel (x, y) of the destination, the source value at the fractional subpixel position (x', y') is constructed by means of an Interpolation object and written to the destination. The mapping between the destination pixel (x, y) and the source position (x', y') is given by:

```
    x' = m[0][0] * x + m[0][1] * y + m[0][2]
    y' = m[1][0] * x + m[1][1] * y + m[1][2]
```

where m is a 3x2 transform matrix that inverts the matrix supplied as the "transform" argument.

When interpolations which require padding the source such as Bilinear or Bicubic interpolation are specified, the source needs to be extended such that it has the extra pixels needed to compute all the destination pixels. This extension is performed via the `BorderExtender` class. The type of Border Extension can be specified as a `RenderingHint` to the `JAI.create` method.

If no BorderExtender is specified (is null), the source will not be extended. The transformed image size is still the same as if the source had been extended. However, since there is insufficient source to compute all the destination pixels, only that subset of the destination image's pixels which can be computed will be written in the destination. The rest of the destination will be set to zeros.

"Affine" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	Affine
LocalName	Affine
Vendor	com.sun.media.jai
Description	Performs interpolated affine transform on an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AffineDescriptor.html
Version	1.0
arg0Desc	The affine transform matrix.
arg1Desc	The interpolation method.

Parameter List

Name	Class Type	Default Value
transform	java.awt.geom.AffineTransform	NO_PARAMETER_DEFAULT
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`AffineTransform`, `Interpolation`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

AffineDescriptor

`public **AffineDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Affine" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "transform" is invertible.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class AffinePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.AffinePropertyGenerator
```

---

class **AffinePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Affine" dynamically.

Constructor Detail

AffinePropertyGenerator

```
public AffinePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Affine".
Specified by:
getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

javax.media.jai.operator

Class AndConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AndConstDescriptor
```

public class **AndConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "AndConst" operation.

This operation takes one rendered or renderable image and an array of integer constants, and performs a bit-wise logical "and" between every pixel in the same band of the source and the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

The source image must have an integral data type. By default, the destination image bound, data type, and number of bands are the same as the source image.

The following matrix defines the logical "and" operation.

Logical "and"

src	const	Result
0	0	0
0	1	0
1	0	0
1	1	1

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = srcs[x][y][b] & constants[0];
} else {
    dst[x][y][b] = srcs[x][y][b] & constants[b];
}
```

Resource List

Name	Value
GlobalName	AndConst
LocalName	AndConst
Vendor	com.sun.media.jai
Description	Logically "ands" a rendered image with constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AndConstDescriptor.html
Version	1.0
arg0Desc	The constants to logically "and" with.

Parameter List

Name	Class Type	Default Value
constants	int[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

AndConstDescriptor

```
public AndConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.
Overrides:
isRenderableSupported in class OperationDescriptorImpl

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer message)
```

Validates the input source and parameter.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an integral data type and that "constants" has length at least 1.
Overrides:
validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class AndDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.AndDescriptor
```

public class **AndDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "And" operation.

The And operation takes two source images, and performs a bit-wise logical "and" on every pair of pixels, one from each source image, of the corresponding position and band. No additional parameters are required.

Both source images must have integral data types. The two data types may be different.

Unless altered by an `ImageLayout` hint, the destination image bound is the intersection of the two source image bounds. If the two sources don’t intersect, the destination will have a width and height of 0. The number of bands of the destination image is equal to the lesser number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types.

The following matrix defines the logical "and" operation.

Logical "and"

src1	src2	Result
1	1	1
1	0	0
0	1	0
0	0	0

The destination pixel values are defined by the pseudocode:

```
dst[x][y][b] = srcs[0][x][y][b] & srcs[1][x][y][b];
```

Resource List

Name	Value
GlobalName	And
LocalName	And
Vendor	com.sun.media.jai
Description	Logically "ands" two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/AndDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

AndDescriptor

```
public AndDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source images are of integral data type.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator
Class BMPDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.BMPDescriptor
```

public class **BMPDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "BMP" operation.

The "BMP" operation reads a standard BMP input stream. The "BMP" operation currently reads Version2, Version3 and some of the Version 4 images, as defined in the Microsoft Windows BMP file format.

Version 4 of the BMP format allows for the specification of alpha values, gamma values and CIE colorspaces. These are not currently handled, but the relevant properties are emitted, if they are available from the BMP image file.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	BMP
LocalName	BMP
Vendor	com.sun.media.jai
Description	Reads an image from a BMP stream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/BMPDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT

See Also:
`SeekableStream`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "BMP" operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter names for the "BMP" operation.

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class types for the "BMP" operation.

---

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default values for the "BMP" operation.

Constructor Detail

BMPDescriptor

`public `**`BMPDescriptor`**`()`

Constructor.

javax.media.jai.operator

Class BandCombineDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.BandCombineDescriptor
```

public class **BandCombineDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "BandCombine" operation.

The BandCombing operation computes a set of arbitrary linear combinations of the bands of a rendered or renderable source image, using a specified matrix. The matrix must a number of rows equal to the number of desired destination bands and a number of columns equal to the number of source bands plus one. In other words, the array may be constructed using the syntax:

```
 double[][] matrix = new double[destBands][sourceBands + 1];
```

The number of source bands used to determine the matrix dimensions is given by `OpImage.getExpandedNumBands(source.getSampleModel(), source.getColorModel()). In particular, if the source image has an IndexColorModel, the number of bands is given by the ColorModel's number of output components.`

`If the result of the computation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.`

Resource List

Name	Value
GlobalName	BandCombine
LocalName	BandCombine
Vendor	com.sun.media.jai
Description	Performs arbitrary interband linear combination using a specified matrix.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/BandCombineDescriptor.html
Version	1.0
arg0Desc	The matrix specifying the band combination.

Parameter List

Name	Class Type	Default Value
matrix	double[][]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

BandCombineDescriptor

```
public BandCombineDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer message)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that "matrix" has at least 1 row and (source bands + 1) columns.

The number of source bands is considered to be equal to `OpImage.getExpandedNumBands(source0.getSampleModel(), source0.getColorModel())`.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class BandSelectDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.BandSelectDescriptor
```

public class **BandSelectDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "BandSelect" operation.

The BandSelect operation chooses `N` bands from a rendered or renderable source image and copies the pixel data of these bands to the destination image in the order specified. The `bandIndices` parameter specifies the source band indices, and its size (`bandIndices.length`) determines the number of bands of the destination image. The destination image may have ay number of bands, and a particular band of the source image may be repeated in the destination image by specifying it multiple times in the `bandIndices` parameter.

Each of the `bandIndices` value should be a valid band index number of the source image. For example, if the source only has two bands, then 1 is a valid band index, but 3 is not. The first band is numbered 0.

The destination pixel values are defined by the pseudocode:

```
 dst[x][y][b] = src[x][y][bandIndices[b]];
```

Resource List

Name	Value
GlobalName	BandSelect
LocalName	BandSelect
Vendor	com.sun.media.jai
Description	Selects n number of bands from a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/BandSelectDescriptor.html
Version	1.0
arg0Desc	The indices of the selected bands.

Parameter List

Name	Class Type	Default Value
bandIndices	int[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

BandSelectDescriptor

```
public BandSelectDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer message)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that "bandIndices" has a length of at least 1 and does not contain any values less than 0 or greater than the number of source bands minus 1.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class BorderDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.BorderDescriptor
```

public class **BorderDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Border" operation.

The Border operation adds a border around a rendered image. The size of the border is specified in pixels by the left, right, top, and bottom padding parameters, corresponding to the four sides of the source image. These paddings may not be less than 0.

The pixel values of the added border area may be set in the following ways using the constants defined in this class:

- it may be extended with zeros (BORDER_ZERO_FILL);
- it may be extended with a constant set of values (BORDER_CONST_FILL);
- it may be created by reflection about the edges of the image (BORDER_REFLECT); or,
- it may be extended by "wrapping" the image plane toroidally, that is, joining opposite edges of the image (BORDER_WRAP).

When choosing the `BORDER_CONST_FILL` option, an array of constants must be supplied. The array must have at least one element, in which case this same constant is applied to all destination image bands. Alternatively, it may have a different constant entry for each corresponding band. For all other border types, this `constants` parameter may be `null`.

Resource List

Name	Value
GlobalName	Border
LocalName	Border
Vendor	com.sun.media.jai
Description	Adds a border around an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/BorderDescriptor.html
Version	1.0
arg0Desc	The image’s left padding.
arg1Desc	The image’s right padding.
arg2Desc	The image’s top padding.
arg3Desc	The image’s bottom padding.
arg4Desc	The border type.
arg5Desc	The constants used by the BORDER_CONST_FILL.

Parameter List

Name	Class Type	Default Value
leftPad	java.lang.Integer	0
rightPad	java.lang.Integer	0
topPad	java.lang.Integer	0
bottomPad	java.lang.Integer	0
type	java.lang.Integer	BORDER_ZERO_FILL
constants	double[]	null

See Also:
`OperationDescriptor`

Field Detail

BORDER_ZERO_FILL

```
public static final int BORDER_ZERO_FILL
```

BORDER_CONST_FILL

```
public static final int BORDER_CONST_FILL
```

BORDER_EXTEND

```
public static final int BORDER_EXTEND
```

BORDER_REFLECT

```
public static final int BORDER_REFLECT
```

BORDER_WRAP

```
public static final int BORDER_WRAP
```

---

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

BorderDescriptor

```
public BorderDescriptor()
```

Constructor.

Method Detail

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

getParamMaxValue

```
public java.lang.Number getParamMaxValue(int index)
```

Returns the maximum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates input parameters.

In addition to the standard checks performed by the superclass method, this method checks that if "type" is equal to `BORDER_CONST_FILL`, "constants" must not a non-`null` instance of `double[]` of length at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class BoxFilterDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.BoxFilterDescriptor
```

public class **BoxFilterDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "BoxFilter" operation.

The "BoxFilter" operation determines the intensity of a pixel in an image by averaging the source pixels within a rectangular area around the pixel. This is a special case of the convolution operation, in which each source pixel contributes the same weight to the destination pixel. The pixel values of the destination image are defined by the pseudocode:

```
     int count = width * height; // # of pixels in the box
     for (int b = 0; b < numBands; b++) {
         int total = 0;
         for (int j = -yKey; j < -yKey + height; j++) {
             for (int i = -xKey; i < -xKey + width; i++) {
                 total += src[x+i][y+j][b];
             }
         }
         dst[x][y][b] = (total + count/2) / count; // round
     }
```

Convolution, like any neighborhood operation, leaves a band of pixels around the edges undefined. For example, for a 3x3 kernel only four kernel elements and four source pixels contribute to the convolution pixel at the corners of the source image. Pixels that do not allow the full kernel to be applied to the source are not included in the destination image. A "Border" operation may be used to add an appropriate border to the source image in order to avoid shrinkage of the image boundaries.

The kernel may not be bigger in any dimension than the image data.

Resource List

Name	Value
GlobalName	BoxFilter
LocalName	BoxFilter
Vendor	com.sun.media.jai
Description	Performs special case convolution where each source pixel contributes equally to the intensity of the destination pixel.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/BoxFilterDescriptor.html
Version	1.0
arg0Desc	The width of the box.
arg1Desc	The height of the box.
arg2Desc	The X position of the key element.
arg3Desc	The Y position of the key element.

Parameter List

Name	Class Type	Default Value
width	java.lang.Integer	NO_PARAMETER_DEFAULT
height	java.lang.Integer	width
xKey	java.lang.Integer	width/2

yKey	java.lang.Integer	height/2

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

BoxFilterDescriptor

`public **BoxFilterDescriptor**()`

Constructor.

Method Detail

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.
Overrides:
getParamMinValue in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Overrides:
validateParameters in class OperationDescriptorImpl

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of

javax.media.jai.operator
Class ClampDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ClampDescriptor
```

public class **ClampDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Clamp" operation.

The Clamp operation takes one rendered or renderable source image, and sets all the pixels whose value is below a "low" value to that low value and all the pixels whose value is above a "high" value to that high value. The pixels whose value is between the "low" value and the "high" value are left unchanged.

A different set of "low" and "high" values may be applied to each band of the source image, or the same set of "low" and "high" values may be applied to all bands of the source. If the number of "low" and "high" values supplied is less than the number of bands of the source, then the values from entry 0 are applied to all the bands. Each "low" value must be less than or equal to its corresponding "high" value.

The destination pixel values are defined by the pseudocode:

```
lowVal = (low.length < dstNumBands) ?
         low[0] : low[b];
highVal = (high.length < dstNumBands) ?
          high[0] : high[b];

if (src[x][y][b] < lowVal) {
    dst[x][y][b] = lowVal;
} else if (src[x][y][b] > highVal) {
    dst[x][y][b] = highVal;
} else {
    dst[x][y][b] = src[x][y][b];
}
```

Resource List

Name	Value
GlobalName	Clamp
LocalName	Clamp
Vendor	com.sun.media.jai
Description	Clamps the pixel values of a rendered image to a specified range.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ClampDescriptor.html
Version	1.0
arg0Desc	The lower boundary for each band.
arg1Desc	The upper boundary for each band.

Parameter List

Name	Class Type	Default Value
low	double[]	NO_PARAMETER_DEFAULT
high	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

ClampDescriptor

`public **ClampDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "low" and "high" have length at least 1 and that each "low" value is less than or equal to the corresponding "high" value.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class ColorConvertDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ColorConvertDescriptor
```

public class **ColorConvertDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "ColorConvert" operation.

The ColorConvert operation performs a pixel-by-pixel color conversion of the data in a rendered or renderable source image.

The data are treated as having no alpha channel, i.e., all bands are color bands. The color space of the source image is specified by the ColorSpace object of the source image ColorModel which must not be null.

Integral data are assumed to occupy the full range of the respective data type; floating point data are assumed to be normalized to the range [0.0,1.0].

By default, the destination image bounds, data type, and number of bands are the same as those of the source image.

Resource List

Name	Value
GlobalName	ColorConvert
LocalName	ColorConvert
Vendor	com.sun.media.jai
Description	Convert the color space of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ColorConvertDescriptor.html
Version	1.0
arg0Desc	The destination color space.

Parameter List

Name	Class Type	Default Value
colorSpace	java.awt.color.ColorSpace	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`, `ColorSpace`, `ColorModel`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

ColorConvertDescriptor

`public ColorConvertDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator

Class CompositeDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.CompositeDescriptor
```

public class **CompositeDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Composite" operation.

The "Composite" operation combines two images based on their alpha values at each pixel. It is done on a per-band basis, and the two source images are expected to have the same number of bands and the same data type. The destination image has the same data type as the two sources.

The `destAlpha` parameter indicates if the destination image should have an extra alpha channel. If this parameter is set to `NO_DESTINATION_ALPHA`, then the destination image does not include an alpha band, and it should have the same number of bands as the two source images. If it is set to `DESTINATION_ALPHA_FIRST`, then the destination image has one extra band than the source images, which represents the result alpha channel, and this band is the first band (band 0) of the destination. If it is set to `DESTINATION_ALPHA_LAST`, then the destination image also has the extra alpha channel, but this band is the last band of the destination.

The destination pixel values may be viewed as representing a fractional pixel coverage or transparency factor. Specifically, Composite implements the Porter-Duff "over" rule (see *Computer Graphics*, July 1984 pp. 253-259), in which the output color of a pixel with source value/alpha tuples `(A, a)` and `(B, b)` is given by `a*A + (1 - a)*(b*B)`. The output alpha value is given by `a + (1 - a)*b`. For premultiplied sources tuples `(a*A, a)` and `(b*B, b)`, the premultiplied output value is simply `(a*A) + (1 - a)*(b*B)`.

The color channels of the two source images are supplied via `source1` and `source2`. The two sources must be either both pre-multiplied by alpha or not. Alpha channel should not be included in `source1` and `source2`.

The alpha channel of the first source images must be supplied via the `source1Alpha` parameter. This parameter may not be null. The alpha channel of the second source image may be supplied via the `source2Alpha` parameter. This parameter may be null, in which case the second source is considered completely opaque. The alpha images should be single-banded, and have the same data type as well as dimensions as their corresponding source images.

The `alphaPremultiplied` parameter indicates whether or not the supplied alpha image is premultiplied to both the source images. It also indicates whether the destination image color channels have the alpha values multiplied to the pixel color values.

The destination image is the combination of the two source images. It has the color channels, and if specified, one additional alpha channel (the band index depends on the value of the `destAlpha` parameter). Whether alpha value is pre-multiplied to the color channels also depend on the value of `alphaPremultiplied` (pre-multiplied if true).

Resource List

Name	Value
GlobalName	composite
LocallName	composite
Vendor	com.sun.media.jai
Description	Composites two images based on an alpha mask.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jaiapi/javax.media.jai.operator.CompositeDescriptor.html
Version	1.0
arg0Desc	The alpha image for the first source.
arg1Desc	The alpha image for the second source.
arg2Desc	True if alpha has been premultiplied to both sources and the destination.
arg3Desc	Indicates if the destination image should include an extra alpha channel, and if so, should it be the first or last band.

Parameter List

Name	Class Type	Default Value
source1Alpha	javax.media.jai.PlanarImage	NO_PARAMETER_DEFAULT
source2Alpha	javax.media.jai.PlanarImage	null
alphaPremultiplied	java.lang.Boolean	false
destAlpha	java.lang.Integer	NO_DESTINATION_ALPHA

See Also:
`ColorModel`, `OperationDescriptor`, `PlanarImage`

Field Detail

NO_DESTINATION_ALPHA

`public static final int` **`NO_DESTINATION_ALPHA`**

The destination image does not have the alpha channel.

DESTINATION_ALPHA_FIRST

`public static final int` **`DESTINATION_ALPHA_FIRST`**

The destination image has the channel, and it is the first band.

DESTINATION_ALPHA_LAST

`public static final int` **`DESTINATION_ALPHA_LAST`**

The destination image has the channel, and it is the last band.

---

resources

`protected static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[]` **`paramClasses`**

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[]` **`paramNames`**

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[]` **`paramDefaults`**

The parameter default value list for this operation.

Constructor Detail

CompositeDescriptor

```
public CompositeDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input sources and parameters.

In addition to the standard checks performed by the superclass method, this method checks that the source image `samplemodel`s have the same number of bands and transfer type, and that the alpha images have the same bounds as the corresponding sources and the correct transfer type.

Overrides:

validateArguments in class OperationDescriptorImpl

---

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

getParamMaxValue

```
public java.lang.Number getParamMaxValue(int index)
```

Returns the maximum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

javax.media.jai.operator

Class ConjugateDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ConjugateDescriptor
```

public class **ConjugateDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Conjugate" operation.

The Conjugate operation negates the imaginary components of pixel values of a rendered or renderable source image containing complex data. The source image must contain an even number of bands with the even-indexed bands (0, 2, ...) representing the real and the odd-indexed bands (1, 3, ...) the imaginary parts of each pixel. The destination image similarly contains an even number of bands with the same interpretation and with contents defined by:

```
 dst[x][y][2*k]   =  src[x][y][2*k];
 dst[x][y][2*k+1] = -src[x][y][2*k+1];
```

where the index *k* varies from zero to one less than the number of complex components in the destination image.

"Conjugate" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.TRUE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	Conjugate
LocalName	Conjugate
Vendor	com.sun.media.jai
Description	Computes the complex conjugate of a complex image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ConjugateDescriptor.html
Version	1.0

No parameters are needed for the "Conjugate" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

ConjugateDescriptor

`public ConjugateDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "Conjugate" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator
Class ConjugatePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.ConjugatePropertyGenerator
```

class **ConjugatePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Conjugate" dynamically.

Constructor Detail

ConjugatePropertyGenerator

```
public ConjugatePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Conjugate".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator

Class ConstantDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ConstantDescriptor
```

public class **ConstantDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Constant" operation.

The Constant operation creates a multi-banded, tiled rendered image, where all the pixels from the same band have a constant value. The width and height of the image must be specified and greater than 0. At least one constant must be supplied. The number of bands of the image is determined by the number of constant pixel values supplied in the "bandValues" parameter. The data type is determined by the type of the constants; this means all elements of the `bandValues` array must be of the same type.

If the `bandValues` array is a `Short` array, then `TYPE_USHORT` is used if all values are non-negative; otherwise `TYPE_SHORT` is used.

Resource List

Name	Value
GlobalName	Constant
LocalName	Constant
Vendor	com.sun.media.jai
Description	Creates a rendered image with constant pixel values.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ConstantDescriptor.html
Version	1.0
arg0Desc	Image width in pixels.
arg1Desc	Image height in pixels.
arg2Desc	The constant pixel band values.

Parameter List

Name	Class Type	Default Value
width	java.lang.Float	NO_PARAMETER_DEFAULT
height	java.lang.Float	NO_PARAMETER_DEFAULT
bandValues	java.lang.Number[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

ConstantDescriptor

```
public ConstantDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "width" and "height" are greater than 0 and that "bandValues" has length at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class ConvolveDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ConvolveDescriptor
```

public class **ConvolveDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Convolve" operation.

Convolution is a spatial operation that computes each output sample by multiplying elements of a kernel with the samples surrounding a particular source sample.

For each destination sample, the kernel is rotated 180 degrees and its "key element," or origin, is placed over the source pixel corresponding with the destination pixel. The kernel elements are multiplied with the source pixels beneath them, and the resulting products are summed together to produce the destination sample value.

Pseudocode for the convolution operation on a single sample dst[x][y] is as follows, assuming the kernel is of size width x height and has already been rotated through 180 degrees. The kernel's Origin element is located at position (xOrigin, yOrigin):

```
dst[x][y] = 0;
for (int i = -xOrigin; i < -xOrigin + width; i++) {
    for (int j = -yOrigin; j < -yOrigin + height; j++) {
        dst[x][y] += src[x + i][y + j]*kernel[xOrigin + i][yOrigin + j];
    }
}
```

Convolution, like any neighborhood operation, leaves a band of pixels around the edges undefined. For example, for a 3x3 kernel only four kernel elements and four source pixels contribute to the convolution pixel at the corners of the source image. Pixels that do not allow the full kernel to be applied to the source are not included in the destination image. A "Border" operation may be used to add an appropriate border to the source image in order to avoid shrinkage of the image boundaries.

The kernel may not be bigger in any dimension than the image data.

Resource List

Name	Value
GlobalName	Convolve
LocalName	Convolve
Vendor	com.sun.media.jai
Description	Performs kernel-based convolution on an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ConvolveDescriptor.html
Version	1.0
arg0Desc	The convolution kernel.

Parameter List

Name	Class Type	Default Value
kernel	javax.media.jai.KernelJAI	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`, `KernelJAI`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for a Convolve operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter names for the Convolve operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the Convolve operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the Convolve operation.

Constructor Detail

ConvolveDescriptor

```
public ConvolveDescriptor()
```

Constructor.

Method Detail

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of

javax.media.jai.operator

Class CropDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.CropDescriptor
```

public class **CropDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Crop" operation.

The Crop operation takes one rendered or renderable image and crops the image to a specified rectangular area. The rectangular area must not be empty, and must be fully contained with the source image bounds.

For rendered images the supplied origin and dimensions are used to determine the smallest rectangle with integral origin and dimensions which encloses the rectangular area requested.

For renderable images the rectangular area is specified in rendering-independent coordinates. When the image is rendered this area will be mapped to rendered image coordinates using the affine transform supplied for the rendering. The crop bounds in rendered coordinates are defined to be the minimum bounding box of the rectangular area mapped to rendered image coordinates.

Resource List

Name	Value
GlobalName	Crop
LocalName	Crop
Vendor	com.sun.media.jai
Description	Crops the pixel values of a rendered image to a specified rectangle.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/CropDescriptor.html
Version	1.0
arg0Desc	The x origin for each band.
arg1Desc	The y origin for each band.
arg2Desc	The width for each band.
arg3Desc	The height for each band.

Parameter List

Name	Class Type	Default Value
x	Float	NO_PARAMETER_DEFAULT
y	Float	NO_PARAMETER_DEFAULT
width	Float	NO_PARAMETER_DEFAULT
height	Float	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

CropDescriptor

```
public CropDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters in the rendered mode.

In addition to the standard checks performed by the superclass method, this method checks that "x", "y", "width", and "height" form a rectangle that is not empty and that is fully contained within the bounds of the source image.

Overrides:

validateArguments in class OperationDescriptorImpl

---

validateRenderableArguments

```
public boolean validateRenderableArguments(java.awt.image.renderable.ParameterBlock args,
                                           java.lang.StringBuffer msg)
```

Validates the input source and parameters in the renderable mode.

In addition to the standard checks performed by the superclass method, this method checks that "x", "y", "width", and "height" form a rectangle that is not empty and that is fully contained within the bounds of the source image.

Overrides:

validateRenderableArguments in class OperationDescriptorImpl

javax.media.jai.operator

Class DCTDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DCTDescriptor
```

---

public class **DCTDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "DCT" operation.

The "DCT" operation computes the even discrete cosine transform (DCT) of an image. Each band of the destination image is derived by performing a two-dimensional DCT on the corresponding band of the source image.

Resource List

Name	Value
GlobalName	DCT
LocalName	DCT
Vendor	com.sun.media.jai
Description	Computes the discrete cosine transform of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DCTDescriptor.html
Version	1.0

No parameters are needed for the "DCT" operation.

See Also:
    `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

DCTDescriptor

`public DCTDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class DFTDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DFTDescriptor
```

public class **DFTDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "DFT" operation.

The "DFT" operation computes the discrete Fourier transform of an image. A negative exponential is used as the basis function for the transform. The operation supports real-to-complex, complex-to-complex, and complex-to-real transforms. A complex image must have an even number of bands, with the even bands (0, 2, ...) representing the real parts and the odd bands (1, 3, ...) the imaginary parts of each complex pixel.

The nature of the source and destination data is specified by the "dataNature" operation parameter. If the source data are complex then the number of bands in the source image must be a multiple of 2. The number of bands in the destination must match that which would be expected given the number of bands in the source image and the specified nature of the source and destination data. If the source image is real then the number of bands in the destination will be twice that in the source. If the destination image is real than the number of bands in the destination will be half that in the source. Otherwise the number of bands in the source and destination must be equal.

If an underlying fast Fourier transform (FFT) implementation is used which requires that the image dimensions be powers of 2, then the width and height may each be increased to the power of 2 greater than or equal to the original width and height, respectively.

"DFT" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.FALSE` if the "dataNature" operation parameter is equal to COMPLEX_TO_REAL and to `java.lang.Boolean.TRUE` otherwise. The value of this property may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	DFT
LocalName	DFT
Vendor	com.sun.media.jai
Description	Computes the discrete Fourier transform of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DFTDescriptor.html
Version	1.0
arg0Desc	The type of scaling to be used.
arg1Desc	The nature of the data.

Parameter List

Name	Class Type	Default Value
scalingType	Integer	DFTDescriptor.SCALING_NONE
dataNature	Integer	DFTDescriptor.REAL_TO_COMPLEX

See Also:
`OperationDescriptor`

Field Detail

SCALING_NONE

`public static final java.lang.Integer` **`SCALING_NONE`**

A flag indicating that the transform is not to be scaled.

SCALING_UNITARY

`public static final java.lang.Integer` **`SCALING_UNITARY`**

A flag indicating that the transform is to be scaled by the square root of the product of its dimensions.

SCALING_DIMENSIONS

`public static final java.lang.Integer` **`SCALING_DIMENSIONS`**

A flag indicating that the transform is to be scaled by the product of its dimensions.

REAL_TO_COMPLEX

`public static final java.lang.Integer` **`REAL_TO_COMPLEX`**

A flag indicating that the source data are real and the destination data complex.

COMPLEX_TO_COMPLEX

`public static final java.lang.Integer` **`COMPLEX_TO_COMPLEX`**

A flag indicating that the source and destination data are both complex.

COMPLEX_TO_REAL

`public static final java.lang.Integer` **`COMPLEX_TO_REAL`**

A flag indicating that the source data are complex and the destination data real.

---

resources

`private static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[]` **`paramClasses`**

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[]` **`paramNames`**

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[]` **`paramDefaults`**

The parameter default value list for this operation.

Constructor Detail

DFTDescriptor

```
public DFTDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that "scalingType" is one of `SCALING_NONE`, `SCALING_UNITARY`, or `SCALING_DIMENSIONS`, and that "dataNature" is one of `REAL_TO_COMPLEX`, `COMPLEX_TO_COMPLEX`, or `COMPLEX_TO_REAL`. Also, if "dataNature" is `COMPLEX_TO_COMPLEX` or `COMPLEX_TO_REAL` the number of source bands must be even.

Overrides:

validateArguments in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "DFT" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class DFTPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.DFTPropertyGenerator
```

class **DFTPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "DFT" dynamically.

Constructor Detail

DFTPropertyGenerator

```
public DFTPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "DFT".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator
Class DivideByConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DivideByConstDescriptor
```

public class **DivideByConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "DivideByConst" operation.

The DivideByConst operation takes one rendered or renderable source image and an array of double constants, and divides every pixel of the same band of the source by the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

In case of division by 0, if the numerator is 0, then the result is set to 0; otherwise, the result is set to the maximum value supported by the destination data type.

By default, the destination image bound, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = srcs[x][y][b]/constants[0];
} else {
    dst[x][y][b] = srcs[x][y][b]/constants[b];
}
```

Resource List

Name	Value
GlobalName	DivideByConst
LocalName	DivideByConst
Vendor	com.sun.media.jai
Description	Divides a rendered image by constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DivideByConstDescriptor.html
Version	1.0
arg0Desc	The constants to be divided by.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

DivideByConstDescriptor

```
public DivideByConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class DivideComplexDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DivideComplexDescriptor
```

public class **DivideComplexDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "DivideComplex" operation.

The "DivideComplex" operation divides two images representing complex data. The source images must each contain an even number of bands with the even-indexed bands (0, 2, ...) representing the real and the odd-indexed bands (1, 3, ...) the imaginary parts of each pixel. The destination image similarly contains an even number of bands with the same interpretation and with contents defined by:

```
 a = src0[x][y][2*k];
 b = src0[x][y][2*k+1];
 c = src1[x][y][2*k];
 d = src1[x][y][2*k+1];

 dst[x][y][2*k] = (a*c + b*d)/(c^2 + d^2)
 dst[x][y][2*k+1] = (b*c - a*d)/(c^2 + d^2)
```

where 0 <= *k* < numBands/2. With one exception, the number of bands of the destination image is the same as the minimum of the number of bands of the two sources, and the data type is the biggest data type of the sources. The exception occurs when one of the source images has two bands, the other source image has N = 2*K bands where K > 1, and an `ImageLayout` hint is provided containing a destination `SampleModel` which specifies M = 2*L bands for the destination image where L > 1 and L <= K. In this special case if the first source has 2 bands its single complex component will be divided by each of the first L complex components of the second source; if the second source has 2 bands its single complex component will divide each of the L complex components of the first source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

"DivideComplex" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.TRUE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	DivideComplex
LocalName	DivideComplex
Vendor	com.sun.media.jai
Description	Compute the complex quotient of two images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DivideComplexDescriptor.html
Version	1.0

No parameters are needed for the "DivideComplex" operation.

See Also:
    `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

DivideComplexDescriptor

```
public DivideComplexDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that both sources have an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "DivideComplex" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class DivideComplexPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.DivideComplexPropertyGenerator
```

class **DivideComplexPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "DivideComplex" dynamically.

Constructor Detail

DivideComplexPropertyGenerator

```
public DivideComplexPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "DivideComplex".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator
Class DivideDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DivideDescriptor
```

public class **DivideDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Divide" operation.

The Divide operation takes two rendered or renderable images, and for every pair of pixels, one from each source image of the corresponding position and band, divides the pixel from the first source by the pixel from the second source. No additional parameters are required for this operation.

In case of division by 0, if the numerator is 0, then the result is set to 0; otherwise, the result is set to the maximum value supported by the destination data type.

The two source images may have different number of bands and data types. By default, the destination image bound is the intersection of the two source image bounds. If the two sources don't intersect, the destination will have a width and a height of 0.

The default number of bands of the destination image is the same as the least number of bands of the sources, and the data type is the biggest data type of the sources.

As a special case, if one of the source images has N bands (N > 1), the other source has 1 band, and an `ImageLayout` hint is provided containing a destination `SampleModel` with K bands (1 < K <= N), then the single band of the 1-banded source will be divided by or into to each of the first K bands of the N-band source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
 dst[x][y][dstBand] = srcs[0][x][y][src0Band]/srcs[1][x][y][src1Band];
```

Resource List

Name	Value
GlobalName	divide
LocalName	divide
Vendor	com.sun.media.jai
Description	Dividies one rendered image by another rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DivideDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

DivideDescriptor

`public` **`DivideDescriptor`**`()`

Constructor.

Method Detail

isRenderableSupported

`public boolean` **`isRenderableSupported`**`()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class DivideIntoConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.DivideIntoConstDescriptor
```

public class **DivideIntoConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "DivideIntoConst" operation.

The DivideIntoConst operation takes one rendered or renderable image and an array of double constants, and divides every pixel of the same band of the source into the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

In case of division by 0, if the numerator is 0, then the result is set to 0; otherwise, the result is set to the maximum value supported by the destination data type.

By default, the destination image bound, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = constants[0]/src[x][y][b];
} else {
    dst[x][y][b] = constants[b]/src[x][y][b];
}
```

Resource List

Name	Value
GlobalName	DivideIntoConst
LocalName	DivideIntoConst
Vendor	com.sun.media.jai
Description	Divides a rendered image into constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/DivideIntoConstDescriptor.html
Version	1.0
arg0Desc	The constants to be divided into.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

DivideIntoConstDescriptor

```
public DivideIntoConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class EncodeDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.EncodeDescriptor
```

public class **EncodeDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Encode" operation. The "Encode" operation writes an image to a given `OutputStream` in a specified format using the supplied encoding parameters.

The third parameter contains an instance of `ImageEncodeParam` to be used during the decoding. It may be set to `null` in order to perform default encoding, or equivalently may be omitted. If non-`null`, it must be of the correct class type for the selected format.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	encode
LocalName	encode
Vendor	com.sun.media.jai
Description	Stores an image to an OutputStream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/EncodeDescriptor.html
Version	1.0
arg0Desc	The OutputStream to write to.
arg1Desc	The format of the created file.
arg2Desc	The encoding parameters.

Parameter List

Name	Class Type	Default Value
stream	java.io.OutputStream	NO_PARAMETER_DEFAULT
format	java.lang.String	"tiff"
param	com.sun.media.jai.codec.ImageEncodeParam	null

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "Encode" operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter names for the "Encode" operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the "Encode" operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the "Encode" operation.

Constructor Detail

EncodeDescriptor

```
public EncodeDescriptor()
```

Constructor.

Method Detail

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that the format name is recognized and is capable of encoding the source image using the encoding parameter "param", if non-`null`.

Overrides:

validateArguments in class OperationDescriptorImpl

isImmediate

```
public boolean isImmediate()
```

Returns true indicating that the operation should be rendered immediately during a call to `JAI.create()`.

Overrides:

isImmediate in class OperationDescriptorImpl

See Also:

`OperationDescriptor`

javax.media.jai.operator
Class ErrorDiffusionDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ErrorDiffusionDescriptor
```

public class **ErrorDiffusionDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "ErrorDiffusion" operation.

The "ErrorDiffusion" operation performs color quantization by finding the nearest color to each pixel in a supplied color map and "diffusing" the color quantization error below and to the right of the pixel.

Resource List

Name	Value
GlobalName	ErrorDiffusion
LocalName	ErrorDiffusion
Vendor	com.sun.media.jai
Description	Performs error diffusion color quantization using a specified color map and error filter.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ErrorDiffusionDescriptor.html
Version	1.0
arg0Desc	The color map.
arg1Desc	The error filter kernel.

Parameter List

Name	Class Type	Default Value
colorMap	javax.media.jai.LookupTableJAI	NO_PARAMETER_DEFAULT
errorKernel	javax.media.jai.KernelJAI	javax.media.jai.KernelJAI.ERROR_FILTER_FLOYD_STEINBERG

See Also:
`LookupTableJAI`, `KernelJAI`, `ColorCube`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "ErrorDiffusion" operation.

paramNames

`private static final java.lang.String[]` **`paramNames`**

The parameter names for the "ErrorDiffusion" operation.

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class types for the "ErrorDiffusion" operation.

---

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default values for the "ErrorDiffusion" operation.

Constructor Detail

ErrorDiffusionDescriptor

`public `**`ErrorDiffusionDescriptor`**`()`

Constructor.

javax.media.jai.operator

Class ExpDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ExpDescriptor
```

public class **ExpDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Exp" operation.

The "Exp" operation takes the exponential of the pixel values of an image. The pixel values of the destination image are defined by the pseudocode:

```
dst[x][y][b] = java.lang.Math.exp(src[x][y][b])
```

For integral image datatypes, the result will be rounded and clamped as needed.

Resource List

Name	Value
GlobalName	Exp
LocalName	Exp
Vendor	com.sun.media.jai
Description	Computes the exponential of the pixel values of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ExpDescriptor.html
Version	1.0

No parameters are needed for the "Exp" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

ExpDescriptor

`public ExpDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class ExtremaDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ExtremaDescriptor
```

public class **ExtremaDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Extrema" operation.

The Extrema operation scans a specific region of a rendered image and finds the maximum and minimum pixel values for each band within that region of the image. The image data pass through this operation unchanged.

The region-wise maximum and minimum pixel values may be obtained as properties. Calling the `getProperty` method on this operation with "extrema" as the property name retrieves both the region-wise maximum and minimum pixel values. Calling it with "maximum" as the property name retrieves the region-wise maximum pixel value, and with "minimum" as the property name retrieves the region-wise minimum pixel value. The return value for "extrema" has type `double[2][#bands]`, and those for "maximum" and "minimum" have type `double[#bands]`.

The region of interest (ROI) does not have to be a rectangle. It may be `null`, in which case the entire image is scanned to find the image-wise maximum and minimum pixel values for each band.

The set of pixels scanned may be further reduced by specifying the "xPeriod" and "yPeriod" parameters that represent the sampling rate along each axis. These variables may not be less than 1. However, they may be `null`, in which case the sampling rate is set to 1; that is, every pixel in the ROI is processed.

Resource List

Name	Value
GlobalName	Extrema
LocalName	Extrema
Vendor	com.sun.media.jai
Description	Finds the maximum and minimum pixel value in each band of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ExtremaDescriptor.html
Version	1.0
arg0Desc	The region of the image to scan.
arg1Desc	The horizontal sampling rate, may not be less than 1.
arg2Desc	The vertical sampling rate, may not be less than 1.

Parameter List

Name	Class Type	Default Value
roi	javax.media.jai.ROI	null
xPeriod	java.lang.Integer	1
yPeriod	java.lang.Integer	1

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

ExtremaDescriptor

`public **ExtremaDescriptor**()`

Constructor.

Method Detail

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

javax.media.jai.operator

Class FPXDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.FPXDescriptor
```

public class **FPXDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "FPX" operation. The "FPX" operation reads an image from a FlashPix stream.

The second parameter contains an instance of `FPXDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	FPX
LocalName	FPX
Vendor	com.sun.media.jai
Description	Reads an image from a FlashPix stream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/FPXDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.
arg1Desc	The FPXDecodeParam to use.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.FPXDecodeParam	null

See Also:
`SeekableStream`, `OperationDescriptor`

Field Detail

MAX_RESOLUTION

`public static final java.lang.Integer` **`MAX_RESOLUTION`**

Convenience name for the Max Resolution of an FPX image

resources

`private static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "FPX" operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "FPX" operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "FPX" operation.

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "FPX" operation.

Constructor Detail

FPXDescriptor

`public **FPXDescriptor**()`

Constructor.

javax.media.jai.operator
Class FileLoadDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.FileLoadDescriptor
```

public class **FileLoadDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "FileLoad" operation.

The `validateArguments()` method checks that the named file exists and is readable. If not, it will return `false`, causing `JAI.createNS()` to throw an `IllegalArgumentException`.

The allowable formats are those registered with the `com.sun.media.jai.codec.ImageCodec` class.

The second parameter contains an instance of `ImageDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	fileload
LocalName	fileload
Vendor	com.sun.media.jai
Description	Reads an image from a file.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/FileLoadDescriptor.html
Version	1.0
arg0Desc	The path of the file to read from.
arg1Desc	The ImageDecodeParam to use.

Parameter List

Name	Class Type	Default Value
filename	java.lang.String	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.ImageDecodeParam	null

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "FileLoad" operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter names for the "FileLoad" operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the "FileLoad" operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the "FileLoad" operation.

Constructor Detail

FileLoadDescriptor

```
public FileLoadDescriptor()
```

Constructor.

Method Detail

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that the source file exists and is readable.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class FileStoreDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.FileStoreDescriptor
```

public class **FileStoreDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "FileStore" operation. The "FileStore" operation writes an image to a given file in a specified format using the supplied encoding parameters.

The third parameter contains an instance of `ImageEncodeParam` to be used during the decoding. It may be set to `null` in order to perform default encoding, or equivalently may be omitted. If non-`null`, it must be of the correct class type for the selected format.

The requested file path must be writable.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	filestore
LocalName	filestore
Vendor	com.sun.media.jai
Description	Stores an image to a file.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/FileStoreDescriptor.html
Version	1.0
arg0Desc	The path of the file to write to.
arg1Desc	The format of the file.
arg2Desc	The encoding parameters.

Parameter List

Name	Class Type	Default Value
filename	java.lang.String	NO_PARAMETER_DEFAULT
format	java.lang.String	"tiff"
param	com.sun.media.jai.codec.ImageEncodeParam	null

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for the "FileStore" operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "FileStore" operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "FileStore" operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "FileStore" operation.

Constructor Detail

FileStoreDescriptor

`public **FileStoreDescriptor**()`

Constructor.

Method Detail

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that the format name is recognized and is capable of encoding the source image using the encoding parameter "param", if non-`null`, ans that the output file path "filename" is writable.

Overrides:

validateArguments in class OperationDescriptorImpl

---

isImmediate

`public boolean **isImmediate**()`

Returns true indicating that the operation should be rendered immediately during a call to `JAI.create()`.

Overrides:

isImmediate in class OperationDescriptorImpl

See Also:

`OperationDescriptor`

javax.media.jai.operator
Class FormatDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.FormatDescriptor
```

public class **FormatDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Format" operation.

The "Format" operation performs reformatting on an image. It is capable of casting the pixel values of an image to a given data type, replacing the SampleModel and ColorModel of an image, and restructuring the image's tile grid layout. The pixel values of the destination image are defined by the pseudocode:

```
dst[x][y][b] = cast(src[x][y][b], dataType)
```

where "dataType" is one of the constants TYPE_BYTE, TYPE_SHORT, TYPE_USHORT, TYPE_INT, TYPE_FLOAT, or TYPE_DOUBLE from `java.awt.image.DataBuffer`.

The output SampleModel, ColorModel and tile grid layout are specified by passing an ImageLayout object as a RenderingHint named "ImageLayout". The output image will have a SampleModel compatible with the one specified in the layout hint wherever possible; however, for output data types of `float` and double a `ComponentSampleModel` will be used regardless of the value of the hint parameter.

The ImageLayout may also specify a tile grid origin and size which will be respected.

The typecasting performed by the `Format` function is defined by the following set of expressions, dependent on the data types of the source and destination. Casting an image to its current data type is a no-op. See The Java Language Specification for the definition of type conversions between primitive types.

In most cases, it is not necessary to explictly perform widening typecasts since they will be performed automatically by image operators when handed source images having different datatypes.

Source Type	Destination Type	Action
BYTE	SHORT	(short)(x & 0xff)
BYTE	USHORT	(short)(x & 0xff)
BYTE	INT	(int)(x & 0xff)
BYTE	FLOAT	(float)(x & 0xff)
BYTE	DOUBLE	(double)(x & 0xff)
SHORT	BYTE	(byte)clamp((int)x, 0, 255)
SHORT	USHORT	(short)clamp((int)x, 0, 32767)
SHORT	INT	(int)x
SHORT	FLOAT	(float)x
SHORT	DOUBLE	(double)x
USHORT	BYTE	(byte)clamp((int)x & 0xffff, 0, 255)
USHORT	SHORT	(short)clamp((int)x & 0xffff, 0, 32767)
USHORT	INT	(int)(x & 0xffff)
USHORT	FLOAT	(float)(x & 0xffff)
USHORT	DOUBLE	(double)(x & 0xffff)
INT	BYTE	(byte)clamp(x, 0, 255)
INT	SHORT	(short)clamp(x, -32768, 32767)
INT	USHORT	(short)clamp(x, 0, 65535)
INT	FLOAT	(float)x
INT	DOUBLE	(double)x
FLOAT	BYTE	(byte)clamp((int)x, 0, 255)
FLOAT	SHORT	(short)clamp((int)x, -32768, 32767)
FLOAT	USHORT	(short)clamp((int)x, 0, 65535)
FLOAT	INT	(int)x
FLOAT	DOUBLE	(double)x
DOUBLE	BYTE	(byte)clamp((int)x, 0, 255)
DOUBLE	SHORT	(short)clamp((int)x, -32768, 32767)
DOUBLE	USHORT	(short)clamp((int)x, 0, 65535)
DOUBLE	INT	(int)x
DOUBLE	FLOAT	(float)x

The `clamp` function may be defined as:

```
int clamp(int x, int low, int high) {
    return (x < low) ? low : ((x > high) ? high : x);
}
```

Resource List

Name	Value
GlobalName	Format
LocalName	Format
Vendor	com.sun.media.jai
Description	Reformats an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/FormatDescriptor.html
Version	1.0
arg0Desc	The output data type (from java.awt.image.DataBuffer).

Parameter List

Name	Class Type	Default Value
dataType	java.lang.Integer	DataBuffer.TYPE_BYTE

See Also:
`DataBuffer`, `ImageLayout`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

FormatDescriptor

`public **FormatDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

getParamMaxValue

`public java.lang.Number **getParamMaxValue**(int index)`

Returns the maximum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

javax.media.jai.operator
Class GIFDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.GIFDescriptor
```

public class **GIFDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "GIF" operation.

The "GIF" operation reads an image from a GIF stream.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	GIF
LocalName	GIF
Vendor	com.sun.media.jai
Description	Reads an image from a GIF stream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/GIFDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT

See Also:
`SeekableStream`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "GIF" operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter names for the "GIF" operation.

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class types for the "GIF" operation.

---

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default values for the "GIF" operation.

Constructor Detail

GIFDescriptor

`public `**`GIFDescriptor`**`()`

Constructor.

javax.media.jai.operator

Class GradientMagnitudeDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.GradientMagnitudeDescriptor
```

public class **GradientMagnitudeDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "GradientMagnitude" operation.

The "GradientMagnitude" operation is an edge detector which computes the magnitude of the image gradient vector in two orthogonal directions.

The result of the "GradientMagnitude" operation may be defined as:

```
 dst[x][y][b] = ((SH(x,y,b))^2 + (SV(x,y,b))^2 )^0.5
```

where SH(x,y,b) and SV(x,y,b) are the horizontal and vertical gradient images generated from band *b* of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks. Origins set on the kernels will be ignored. The origins are assumed to be width/2 & height/2.

Resource List

Name	Value
GlobalName	GradientMagnitude
LocallName	GradientMagnitude
Vendor	com.sun.media.jai
Description	Performs gradient magnitude edge detection on an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jaiapi/javax.media.jai.operator.GradientMagnitudeDescriptor.html
Version	1.0
arg0Desc	A gradient mask
arg1Desc	A gradient mask orthogonal to the first one.

Parameter List

Name	Class Type	Default Value
mask1	javax.media.jai.KernelJAI	KernalJAI.GRADIENT_MASK_SOBEL_HORIZONTAL
mask2	javax.media.jai.KernelJAI	KernalJAI.GRADIENT_MASK_SOBEL_VERTICAL

See Also:
`OperationDescriptor`, `KernelJAI`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for the GradientMagnitude operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter names for the GradientMagnitude operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the GradientMagnitude operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the GradientMagnitude operation.

Constructor Detail

GradientMagnitudeDescriptor

```
public GradientMagnitudeDescriptor()
```

Constructor for the GradientMagnitudeDescriptor.

Method Detail

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "mask1" and "mask2" have the same dimensions.

Overrides:

validateParameters in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of

javax.media.jai.operator
Class HistogramDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.HistogramDescriptor
```

public class **HistogramDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Histogram" operation.

The Histogram operation scans a specific region of a rendered image and generates a histogram based on the pixel values within that region of the image. The histogram data is stored in the user supplied `javax.media.jai.Histogram` object, and may be retrieved by calling the `getProperty` method on this operation with "histogram" as the property name. The return value will be of type `javax.media.jai.Histogram`. The image data pass through this operation unchanged.

The region of interest (ROI) does not have to be a rectangle. It may be `null`, in which case the entire image is scanned to generate the histogram.

The set of pixels scanned may be further reduced by specifying the "xPeriod" and "yPeriod" parameters that represent the sampling rate along each axis. These variables may not be less than 1. However, they may be `null`, in which case the sampling rate is set to 1; that is, every pixel in the ROI is processed.

Resource List

Name	Value
GlobalName	Histogram
LocalName	Histogram
Vendor	com.sun.media.jai
Description	Generates a histogram based on the pixel values within a specified region of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/HistogramDescriptor.html
Version	1.0
arg0Desc	The specification for the type of histogram to be generated.
arg1Desc	The region of the image to scan.
arg2Desc	The horizontal sampling rate, may not be less than 1.
arg3Desc	The vertical sampling rate, may not be less than 1.

Parameter List

Name	Class Type	Default Value
histogram	javax.media.jai.Histogram	NO_PARAMETER_DEFAULT
roi	javax.media.jai.ROI	null
xPeriod	java.lang.Integer	1
yPeriod	java.lang.Integer	1

See Also:
`Histogram`, `ROI`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

HistogramDescriptor

`public **HistogramDescriptor**()`

Constructor.

Method Detail

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

javax.media.jai.operator
Class IDCTDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.IDCTDescriptor
```

public class **IDCTDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "IDCT" operation.

The "IDCT" operation computes the inverse even discrete cosine transform (DCT) of an image. Each band of the destination image is derived by performing a two-dimensional inverse DCT on the corresponding band of the source image.

Resource List

Name	Value
GlobalName	IDCT
LocalName	IDCT
Vendor	com.sun.media.jai
Description	Computes the inverse discrete cosine transform of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/IDCTDescriptor.html
Version	1.0

No parameters are needed for the "IDCT" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

IDCTDescriptor

`public IDCTDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class IDFTDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.IDFTDescriptor
```

public class **IDFTDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "IDFT" operation.

The "IDFT" operation computes the inverse discrete Fourier transform of an image. A positive exponential is used as the basis function for the transform. The operation supports real-to-complex, complex-to-complex, and complex-to-real transforms. A complex image must have an even number of bands, with the even bands (0, 2, ...) representing the real parts and the odd bands (1, 3, ...) the imaginary parts of each complex pixel.

The nature of the source and destination data is specified by the "dataNature" operation parameter. If the source data are complex then the number of bands in the source image must be a multiple of 2. The number of bands in the destination must match that which would be expected given the number of bands in the source image and the specified nature of the source and destination data. If the source image is real then the number of bands in the destination will be twice that in the source. If the destination image is real than the number of bands in the destination will be half that in the source. Otherwise the number of bands in the source and destination must be equal.

If an underlying fast Fourier transform (FFT) implementation is used which requires that the image dimensions be powers of 2, then the width and height may each be increased to the power of 2 greater than or equal to the original width and height, respectively.

"IDFT" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.FALSE` if the "dataNature" operation parameter is equal to DFTDescriptor.COMPLEX_TO_REAL and to `java.lang.Boolean.TRUE` otherwise. The value of this property may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	IDFT
LocalName	IDFT
Vendor	com.sun.media.jai
Description	Computes the discrete Fourier transform of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/IDFTDescriptor.html
Version	1.0
arg0Desc	The type of scaling to be used.
arg1Desc	The nature of the data.

Parameter List

Name	Class Type	Default Value
scalingType	Integer	DFTDescriptor.SCALING_DIMENSIONS
dataNature	Integer	DFTDescriptor.COMPLEX_TO_REAL

See Also:
`OperationDescriptor`, `DFTDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

IDFTDescriptor

`public IDFTDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that "scalingType" is one of `SCALING_NONE`, `SCALING_UNITARY`, or `SCALING_DIMENSIONS`, and that "dataNature" is one of `REAL_TO_COMPLEX`, `COMPLEX_TO_COMPLEX`, or `COMPLEX_TO_REAL`. Also, if "dataNature" is `COMPLEX_TO_COMPLEX` or `COMPLEX_TO_REAL` the number of source bands must be even.

Overrides:

validateArguments in class OperationDescriptorImpl

---

getPropertyGenerators

`public PropertyGenerator[] getPropertyGenerators()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "IDFT" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class IDFTPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.IDFTPropertyGenerator
```

class **IDFTPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "IDFT" dynamically.

Constructor Detail

IDFTPropertyGenerator

```
public IDFTPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "IDFT".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator

Class IIPDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.IIPDescriptor
```

public class **IIPDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "IIP" operation.

This operation provides client-side support of the Internet Imaging Protocol (IIP) in both the rendered and renderable modes. It creates a `java.awt.image.RenderedImage` or a `java.awt.image.renderable.RenderableImage` based on the data received from the IIP server, and optionally applies a sequence of operations to the created image.

The operations that may be applied and the order in which they are applied are defined in section 2.2.1.1 of the Internet Imaging Protocol Specification version 1.0.5. Some or all of the requested operations may be executed on the IIP server if it is determined that the server supports such operations. Any of the requested operations not supported by the server will be executed on the host on which the operation chain is rendered.

The processing sequence for the supplied operations is as follows:

- filtering (blur or sharpen);
- tone and color correction ("color twist");
- contrast adjustment;
- selection of source rectangle of interest;
- spatial orientation (rendering-independent affine transformation);
- selection of destination rectangle of interest;
- rendering transformation (renderable mode only);
- transposition (rotation and/or mirroring).

As indicated, the rendering transformation is performed only in renderable mode processing. This transformation is derived from the `AffineTransform` supplied in the `RenderContext` when rendering actually occurs. Rendered mode processing creates a `RenderedImage` which is the default rendering of the `RenderableImage` created in renderable mode processing.

The "URL" parameter specifies the URL of the IIP image as a `java.lang.String`. It must represent a valid URL, and include any required FIF or SDS commands. It cannot be `null`.

The "subImages" parameter optionally indicates the sub-images to be used by the server to get the images at each resolution level. The values in this `int` array cannot be negative. If this parameter is not specified, or if the array is too short (length is 0), or if a negative value is specified, then this operation will use the zeroth sub-image of the resolution level actually processed.

The "filter" parameter specifies a blur or sharpen operation: a positive value indicates sharpen and a negative value blur. A unit step should produce a perceptible change in the image. The default value is 0 which signifies that no filtering will occur.

The "colorTwist" parameter represents a 4x4 matrix stored in row-major order and should have an array length of at least 16. If an array of length greater than 16 is specified, all elements from index 16 and beyond are ignored. Elements 12, 13 and 14 must be 0. This matrix will be applied to the (possibly padded) data in an intermediate normalized PhotoYCC color space with a premultiplied alpha channel. This operation will force an alpha channel to be added to the image if the last column of the last row of the color twist matrix is not 1.0F. Also, if the image originally has a grayscale color space it will be cast up to RGB if casting the data back to grayscale after applying the color twist matrix would result in any loss of data.

The "contrast" parameter specifies a contrast enhancement operation with increasing contrast for larger value. It must be greater than or equal to 1.0F. A value of 1.0F indicates no contrast adjustment.

The "sourceROI" parameter specifies the rectangle of interest in the source image in rendering-independent coordinates. The intersection of this rectangle with the rendering-independent bounds of the source image must equal itself. The rendering-independent bounds of the source image are defined to be (0.0F, 0.0F, r, 1.0F) where *r* is the aspect ratio (width/height) of the source image. Note that the source image will not in fact be cropped to these limits but values outside of this rectangle will be suppressed.

The "transform" parameter represents an affine backward mapping to be applied in rendering-independent coordinates. Note that the direction of transformation is opposite to that of the `AffineTransform` supplied in the `RenderContext` which is a forward mapping. The default value of this transform is the identity mapping. The supplied `AffineTransform` must be invertible.

The "aspectRatio" parameter specifies the rendering-independent width of the destination image and must be positive. The rendering-independent bounds of the destination image are (0.0F, 0.0F, aspectRatio, 1.0F). If this parameter is not provided the destination aspect ratio defaults to that of the source.

The "destROI" parameter specifies the rectangle of interest in the destination image in rendering-independent coordinates. This rectangle must have a non-empty intersection with the rendering-independent bounds of the destination image but is not constrained to the destination image bounds.

A counterclockwise rotation may be applied to the destination image. However, the angle is limited to 0, 90, 180, or 270 degrees. By default, the destination image is not rotated.

The "mirrorAxis" parameter may be `null`, in which case no flipping is applied, or a `String` of "x", "X", "y", or "Y".

The "ICCProfile" parameter may only be used with client-side processing or with server-side processing if the connection protocol supports the ability to transfer a profile.

The "JPEGQuality" and "JPEGTable" parameters are only used with server-side processing. If provided, JPEGQuality must be in the range [0,100] and JPEGTable in [1,255].

There is no source image associated with this operation.

Resource List

Name	Value
GlobalName	IIP
LocalName	IIP
Vendor	com.sun.media.jai
Description	Provides client support of the Internet Imaging Protocol in the rendered and renderable modes.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/IIPDescriptor.html
Version	1.0
arg0Desc	The URL of the IIP image.
arg1Desc	The sub-images to be used by the server for images at each resolution level.
arg2Desc	The filtering value.
arg3Desc	The color twist matrix.
arg4Desc	The contrast value.
arg5Desc	The source rectangle of interest in rendering-independent coordinates.
arg6Desc	The rendering-independent spatial orientation transform.
arg7Desc	The aspect ratio of the destination image.
arg8Desc	The destination rectangle of interest in rendering-independent coordinates.
arg9Desc	The counterclockwise rotation angle to be applied to the destination.
arg10Desc	The mirror axis.
arg11Desc	The ICC profile used to represent the color space of the source image.
arg12Desc	The JPEG quality factor.
arg13Desc	The JPEG compression group index number.

Parameter List

Name	Class Type	Default Value
URL	java.lang.String	NO_PARAMETER_DEFAULT
subImages	int[]	{ 0 }
filter	java.lang.Float	0.0F
colorTwist	float[]	null
contrast	java.lang.Float	1.0F
sourceROI	java.awt.geom.Rectangle2D.Float	null
transform	java.awt.geom.AffineTransform	identity transform
aspectRatio	java.lang.Float	null
destROI	java.awt.geom.Rectangle2D.Float	null
rotation	java.lang.Integer	0
mirrorAxis	java.lang.String	null
ICCProfile	java.awt.color.ICC_Profile	null
JPEGQuality	java.lang.Integer	null
JPEGTable	java.lang.Integer	null

Since:
1.0
See Also:
Digital Imaging Group, `RenderedImage`, `RenderableImage`, `IIPResolutionDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for this operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for this operation.

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for this operation. For those parameters whose default value is `null`, an appropriate value is chosen by the individual implementation.

Constructor Detail

IIPDescriptor

```
public IIPDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Overrides super class's default implementation to return `true` because this operation supports renderable mode.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation. If the supplied `index` does not correspond to a numeric parameter, this method returns `null`.

Throws:

ArrayIndexOutOfBoundsException - if `index` is less than 0 or greater than 13.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

getParamMaxValue

```
public java.lang.Number getParamMaxValue(int index)
```

Returns the maximum legal value of a specified numeric parameter for this operation. If the supplied `index` does not correspond to a numeric parameter, this method returns `null`.

Throws:

ArrayIndexOutOfBoundsException - if `index` is less than 0 or greater than 13.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that:

- the supplied URL string specifies a valid protocol;
- the color twist, if not `null`, has an array length of at least 16 (all elements from index 16 and beyond are ignored and elements 12, 13, and 14 are set to 0);
- both the source and dest ROI, if not `null`, has a width and height greater than 0;
- the mirror axis, if not `null`, has a `String` of "x", "X", "y", or "Y";
- the destination rotation is one of the valid degrees (0, 90. 180, 270).

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class IIPResolutionDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.IIPResolutionDescriptor
```

public class **IIPResolutionDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "IIPResolution" operation.

This operation provides client-side support of the Internet Imaging Protocol (IIP) in the rendered mode. It is resolution-specific. It requests from the IIP server an image at a particular resolution level, and creates a `java.awt.image.RenderedImage` based on the data received from the server. Once the `RenderedImage` is created, the resolution level cannot be changed.

The layout of the created `RenderedImage` is set as follows:

- `minX`, `minY`, `tileGridXOffset`, and `tileGridYOffset` are set to 0;
- `width` and `height` are determined based on the specified resolution level;
- `tileWidth` and `tileHeight` are set to 64;
- `sampleModel` is of the type `java.awt.image.PixelInterleavedSampleModel` with byte data type and the appropriate number of bands;
- `colorModel` is of the type `java.awt.image.ComponentColorModel`, with the `ColorSpace` set to sRGB, PhotoYCC, or Grayscale, depending on the color space of the remote image; if an alpha channel is present, it will be premultiplied.

The "URL" parameter specifies the URL of the IIP image as a `java.lang.String`. It must represent a valid URL, and include any required FIF or SDS commands. It cannot be `null`.

The "resolution" parameter specifies the resolution level of the requested IIP image from the server. The lowest resolution level is 0, with larger integers representing higher resolution levels. If the requested resolution level does not exist, the nearest resolution level is used. If this parameter is not specified, it is set to the default value `IIPResolutionDescriptor.MAX_RESOLUTION` which indicates the highest resolution level.

The "subImage" parameter indicates the sub-image to be used by the server to get the image at the specified resolution level. This parameter cannot be negative. If this parameter is not specified, it is set to the default value 0.

There is no source image associated with this operation.

If available from the IIP server certain properties may be set on the `RenderedImage`. The names of properties and the class types of their associated values are listed in the following table.

Property List

Property Name	Property Value Class Type
affine-transform	java.awt.geom.AffineTransform
app-name	java.lang.String
aspect-ratio	java.lang.Float
author	java.lang.String
colorspace	int[]
color-twist	float[16]
comment	java.lang.String
contrast-adjust	java.lang.Float
copyright	java.lang.String
create-dtm	java.lang.String
edit-time	java.lang.String
filtering-value	java.lang.Float
iip	java.lang.String
iip-server	java.lang.String
keywords	java.lang.String
last-author	java.lang.String
last-printed	java.lang.String
last-save-dtm	java.lang.String
max-size	int[2]
resolution-number	java.lang.Integer
rev-number	java.lang.String
roi-iip	java.awt.geom.Rectangle2D.Float
subject	java.lang.String
title	java.lang.String

For information on the significance of each of the above properties please refer to the IIP specification.

Resource List

Name	Value
GlobalName	IIPResolution
LocalName	IIPResolution
Vendor	com.sun.media.jai
Description	Provides client-side support of the Internet Imaging Protocol in the rendered mode.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/IIPResolutionDescriptor.html
Version	1.0
arg0Desc	The URL of the IIP image.
arg1Desc	The resolution level to request.
arg2Desc	The sub-image to be used by the server.

Parameter List

Name	Class Type	Default Value
URL	java.lang.String	NO_PARAMETER_DEFAULT
resolution	java.lang.Integer	IIPResolutionDescriptor.MAX_RESOLUTION
subImage	java.lang.Integer	0

Since:
1.0
See Also:
Digital Imaging Group, `RenderedImage`, `IIPDescriptor`

Field Detail

MAX_RESOLUTION

`public static final java.lang.Integer` **`MAX_RESOLUTION`**

Convenience name for Max Resolution of an image on an IIP server.

---

resources

`private static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[]` **`paramClasses`**

The parameter class types for this operation.

---

paramNames

`private static final java.lang.String[]` **`paramNames`**

The parameter names for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for this operation.

Constructor Detail

IIPResolutionDescriptor

```
public IIPResolutionDescriptor()
```

Constructor.

Method Detail

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation. If the supplied `index` does not correspond to a numeric parameter, this method returns `null`.

Returns:

An `Integer` of value 0 if `index` is 1 or 2, or `null` if `index` is 0.

Throws:

ArrayIndexOutOfBoundsException - if `index` is less than 0 or greater than 2.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that the supplied URL string specifies a valid protocol.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class ImageFunctionDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ImageFunctionDescriptor
```

public class **ImageFunctionDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "ImageFunction" operation.

The "ImageFunction" operation generates an image on the basis of a functional description provided by an object which is an instance of a class which implements the `ImageFunction` interface. The *(x,y)* coordinates passed to the `getElements()` methods of the `ImageFunction` object are derived by applying an optional translation and scaling to the X- and Y-coordinates of the image. The image X- and Y-coordinates as usual depend on the values of the minimum X- and Y- coordinates of the image which need not be zero. Specifically, the function coordinates passed to `getElements()` are calculated from the image coordinates as:

```
 functionX = xScale*(imageX - xTrans);
 functionY = yScale*(imageY - yTrans);
```

This implies that the pixel at coordinates *(xTrans,yTrans)* will be assigned the value of the function at *(0,0)*.

The number of bands in the destination image must be equal to the value returned by the `getNumElements()` method of the `ImageFunction` unless the `isComplex()` method of the `ImageFunction` returns `true` in which case it will be twice that. The data type of the destination image is determined by the `SampleModel` specified by an `ImageLayout` object provided via a hint. If no layout hint is provided, the data type will default to single-precision floating point. The double precision floating point form of the `getElements()` method of the `ImageFunction` will be invoked if and only if the data type is specified to be `double`. For all other data types the single precision form of `getElements()` will be invoked and the destination sample values will be clamped to the data type of the image.

The width and height of the image are provided explicitely as parameters. These values override the width and height specified via an `ImageLayout` if such is provided.

"ImageFunction" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.TRUE` or `java.lang.Boolean.FALSE` depending on whether the `isComplex()` method of the `ImageFunction` parameter returns `true` or `false`, respectively. This property may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	ImageFunction
LocalName	ImageFunction
Vendor	com.sun.media.jai
Description	Generates an image from a functional description.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ImageFunctionDescriptor.html
Version	1.0
arg0Desc	The functional description.
arg1Desc	The image width.
arg2Desc	The image height.
arg3Desc	The X scale factor.
arg4Desc	The Y scale factor.
arg5Desc	The X translation.
arg6Desc	The Y translation.

Parameter List

Name	Class Type	Default Value
function	javax.media.jai.ImageFunction	NO_PARAMETER_DEFAULT
width	java.lang.Integer	NO_PARAMETER_DEFAULT
height	java.lang.Integer	NO_PARAMETER_DEFAULT
xScale	java.lang.Float	1.0F
yScale	java.lang.Float	1.0F
xTrans	java.lang.Float	0.0F
yTrans	java.lang.Float	0.0F

See Also:
`AffineTransform`, `OperationDescriptor`, `ImageFunction`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

ImageFunctionDescriptor

`public **ImageFunctionDescriptor**()`

Constructor.

Method Detail

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "ImageFunction" operation.

Returns:
An array of property generators.

Overrides:
getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class ImageFunctionPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.ImageFunctionPropertyGenerator
```

class **ImageFunctionPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "ImageFunction" dynamically.

Constructor Detail

ImageFunctionPropertyGenerator

```
public ImageFunctionPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "ImageFunction".

Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

javax.media.jai.operator
Class InvertDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.InvertDescriptor
```

public class **InvertDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Invert" operation.

The "Invert" operation inverts the pixel values of an image. For source images with signed data types, the pixel values of the destination image are defined by the pseudocode:

```
dst[x][y][b] = -src[x][y][b]
```

For unsigned data types, the destination values are defined by:

```
dst[x][y][b] = MAX_VALUE - src[x][y][b]
```

where `MAX_VALUE` is the maximum value supported by the system of the data type of the source pixel.

Resource List

Name	Value
GlobalName	Invert
LocalName	Invert
Vendor	com.sun.media.jai
Description	Invert the pixel values of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/InvertDescriptor.html
Version	1.0

No parameters are needed for the "Invert" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

InvertDescriptor

`public **InvertDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class JPEGDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.JPEGDescriptor
```

public class **JPEGDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "JPEG" operation.

The "JPEG" operation reads an image from a JPEG (JFIF) stream.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	JPEG
LocalName	JPEG
Vendor	com.sun.media.jai
Description	Reads an image from a JFIF (JPEG) stream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/JPEGDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT

See Also:
`SeekableStream`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "JPEG" operation.

paramNames

`private static final java.lang.String[] `**`paramNames`**

The parameter names for the "JPEG" operation.

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class types for the "JPEG" operation.

---

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default values for the "JPEG" operation.

Constructor Detail

JPEGDescriptor

`public `**`JPEGDescriptor`**`()`

Constructor.

javax.media.jai.operator

Class JaiI18N

```
java.lang.Object
  |
  +--javax.media.jai.operator.JaiI18N
```

---

class **JaiI18N**
extends java.lang.Object

Field Detail

packageName

`static java.lang.String packageName`

Constructor Detail

JaiI18N

`JaiI18N()`

Method Detail

getString

`public static java.lang.String getString(java.lang.String key)`

javax.media.jai.operator

Class LogDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.LogDescriptor
```

public class **LogDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Log" operation.

The "Log" operation takes the natural logarithm of the pixel values of an image. The operation is done on a per-pixel, per-band basis. For integral data types, the result will be rounded and clamped as needed. The pixel values of the destination image are defined as:

```
 dst[x][y][b] = java.lang.Math.log(src[x][y][b])
```

For all integral data types, the log of 0 is set to 0. For signed integral data types (`short` and `int`), the log of a negative pixel value is set to -1.

For all floating point data types ((`float` and `double`), the log of 0 is set to `-Infinity`, and the log of a negative pixel value is set to `NaN`.

Resource List

Name	Value
GlobalName	Log
LocalName	Log
Vendor	com.sun.media.jai
Description	Computes the natural logarithm of the pixel values of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/LogDescriptor.html
Version	1.0

No parameters are needed for the "Log" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

LogDescriptor

`public LogDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator

Class LookupDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.LookupDescriptor
```

public class **LookupDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Lookup" operation.

The Lookup operation takes a rendered or renderable image and a lookup table, and performs general table lookup by passing the source image through the table.

The source may be a single- or multi-banded image of data types `byte`, `ushort`, `short`, or `int`. The lookup table may be single- or multi-banded and of any JAI supported data types. The destination image must have the same data type as the lookup table, and its number of bands is determined based on the number of bands of the source and the table. If the source is single-banded, the destination has the same number of bands as the lookup table; otherwise, the destination has the same number of bands as the source.

If either the source or the table is single-banded and the other one is multi-banded, then the single band is applied to every band of the multi-banded object. If both are multi-banded, then their corresponding bands are matched up.

The table may have a set of offset values, one for each band. This value is subtracted from the source pixel values before indexing into the table data array.

It is the user's responsibility to make certain the lookup table supplied is suitable for the source image. Specifically, the table data covers the entire range of the source data. Otherwise, the result of this operation is undefined.

By the nature of this operation, the destination may have a different number of bands and/or data type from the source. The `SampleModel` of the destination is created in accordance with the actual lookup table used in a specific case.

The destination pixel values are defined by the pseudocode:

- If the source image is single-banded and the lookup table is single- or multi-banded, then the destination image has the same number of bands as the lookup table:
  ```
  dst[x][y][b] = table[b][src[x][y][0] - offsets[b]]
  ```
- If the source image is multi-banded and the lookup table is single-banded, then the destination image has the same number of bands as the source image:
  ```
  dst[x][y][b] = table[0][src[x][y][b] - offsets[0]]
  ```
- If the source image is multi-banded and the lookup table is multi-banded, with the same number of bands as the source image, then the destination image will have the same number of bands as the source image:
  ```
  dst[x][y][b] = table[b][src[x][y][b] - offsets[b]]
  ```

Resource List

Name	Value
GlobalName	Lookup
LocalName	Lookup
Vendor	com.sun.media.jai
Description	Performs general table lookup on a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/LookupDescriptor.html
Version	1.0
arg0Desc	The lookup table the source image is passed through.

Parameter List

Name	Class Type	Default Value
table	javax.media.jai.LookupTableJAI	NO_PARAMETER_DEFAULT

See Also:
`LookupTableJAI`, `OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

LookupDescriptor

```
public LookupDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input source.

In addition to the standard checks performed by the superclass method, this method checks that the source image is of integral data type.

Overrides:
validateSources in class OperationDescriptorImpl

javax.media.jai.operator

Class MagnitudeDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MagnitudeDescriptor
```

public class **MagnitudeDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Magnitude" operation.

The "Magnitude" operation computes the magnitude of each pixel of a complex image. The source image must have an even number of bands, with the even bands (0, 2, ...) representing the real parts and the odd bands (1, 3, ...) the imaginary parts of each complex pixel. The destination image has at most half the number of bands of the source image with each sample in a pixel representing the magnitude of the corresponding complex source sample. The magnitude values of the destination image are defined for a given sample by the pseudocode:

```
dstPixel[x][y][b] = sqrt(src[x][y][2*b]^2 + src[x][y][2*b + 1]^2)
```

where the number of bands *b* varies from zero to one less than the number of bands in the destination image.

For integral image datatypes, the result will be rounded and clamped as needed.

"Magnitude" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.FALSE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	Magnitude
LocalName	Magnitude
Vendor	com.sun.media.jai
Description	Find the magnitude of each pixel of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MagnitudeDescriptor.html
Version	1.0

No parameters are needed for the "Magnitude" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MagnitudeDescriptor

`public MagnitudeDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input source.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "Magnitude" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class MagnitudePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.MagnitudePropertyGenerator
```

---

class **MagnitudePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Magnitude" dynamically.

Constructor Detail

MagnitudePropertyGenerator

```
public MagnitudePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Magnitude".

Specified by:

getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator
Class MagnitudeSquaredDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MagnitudeSquaredDescriptor
```

public class **MagnitudeSquaredDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "MagnitudeSquared" operation.

The "MagnitudeSquared" operation computes the squared magnitude or of each pixel of a complex image. The source image must have an even number of bands, with the even bands (0, 2, ...) representing the real parts and the odd bands (1, 3, ...) the imaginary parts of each complex pixel. The destination image has at most half the number of bands of the source image with each sample in a pixel representing the magnitude of the corresponding complex source sample. The magnitude squared values of the destination image are defined for a given sample by the pseudocode:

```
dstPixel[x][y][b] = src[x][y][2*b]^2 + src[x][y][2*b + 1]^2
```

where the number of bands *b* varies from zero to one less than the number of bands in the destination image.

For integral image datatypes, the result will be rounded and clamped as needed.

"MagnitudeSquared" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.FALSE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	MagnitudeSquared
LocalName	MagnitudeSquared
Vendor	com.sun.media.jai
Description	Computes the squared magnitude of each pixel of a complex image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MagnitudeSquaredDescriptor.html
Version	1.0

No parameters are needed for the "MagnitudeSquared" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MagnitudeSquaredDescriptor

`public `**`MagnitudeSquaredDescriptor`**`()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "MagnitudeSquared" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input source.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator
Class MagnitudeSquaredPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.MagnitudeSquaredPropertyGenerator
```

---

class **MagnitudeSquaredPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "MagnitudeSquared" dynamically.

Constructor Detail

MagnitudeSquaredPropertyGenerator

```
public MagnitudeSquaredPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "MagnitudeSquared".

Specified by:

getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator

Class MatchCDFDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MatchCDFDescriptor
```

public class **MatchCDFDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "MatchCDF" operation.

The "MatchCDF" operation performs a piecewise linear mapping of the pixel values of an image such that the Cumulative Distribution Function (CDF) of the destination image matches as closely as possible a specified Cumulative Distribution Function. The desired CDF is described by an array of the form

`float CDF[numBands][numBins[b]]`

where

`numBins[b]`

denotes the number of bins in the histogram of the source image for band *b*. Each element in the array

`CDF[b]`

must be non-negative, the array must represent a non- decreasing sequence, and the last element of the array must be 1.0F. The source image must have a `Histogram` object available via its `getProperty()` method.

Resource List

Name	Value
GlobalName	MatchCDF
LocalName	MatchCDF
Vendor	com.sun.media.jai
Description	Matches pixel values to a supplied CDF.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MatchCDFDescriptor.html
Version	1.0
arg0Desc	The desired Cumulative Distribution Function.

Parameter List

Name	Class Type	Default Value
CDF	float[][]	NO_PARAMETER_DEFAULT

See Also:
`DataBuffer`, `ImageLayout`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

MatchCDFDescriptor

```
public MatchCDFDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input sources and parameter.

In addition to the standard checks performed by the superclass method, this method checks that the source image contains a "histogram" property and that the "CDF" array is appropriate for it.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class MaxDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MaxDescriptor
```

public class **MaxDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Max" operation.

The Max operation takes two rendered images, and for every pair of pixels, one from each source image of the corresponding position and band, finds the maximum pixel value. No additional parameters are required.

The two sources may have different number of bands and/or data types. By default, the destination image bound is the intersection of the two source image bounds. If the two sources don't intersect, the destination will have a width and a height of 0. The number of bands of the destination image is the same as the least number of bands of the sources, and the data type is the biggest data type of the sources.

The destination pixel values are defined by the pseudocode:

```
 if (srcs[0][x][y][b] > srcs[1][x][y][b]) {
     dst[x][y][b] = srcs[0][x][y][b];
 } else {
     dst[x][y][b] = srcs[1][x][y][b];
 }
```

Resource List

Name	Value
GlobalName	Max
LocalName	Max
Vendor	com.sun.media.jai
Description	Computes the pixel-wise maximum of two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MaxDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MaxDescriptor

`public MaxDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean` **`isRenderableSupported`**`()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class MeanDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MeanDescriptor
```

public class **MeanDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Mean" operation.

The Mean operation scans a specific region of a rendered image and computes the mean pixel value for each band within that region of the image. The image data pass through this operation unchanged.

The region-wise mean pixel value for each band may be retrieved by calling the `getProperty` method on this operation with "mean" as the property name. The return value has type `double[#bands]`.

The region of interest (ROI) does not have to be a rectangle. It may be `null`, in which case the entire image is scanned to find the image-wise mean pixel value for each band.

The set of pixels scanned may be further reduced by specifying the "xPeriod" and "yPeriod" parameters that represent the sampling rate along each axis. These variables may not be less than 1. However, they may be `null`, in which case the sampling rate is set to 1; that is, every pixel in the ROI is processed.

Resource List

Name	**Value**
GlobalName	Mean
LocalName	Mean
Vendor	com.sun.media.jai
Description	Calculates the region-wise mean pixel value for each band of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MeanDescriptor.html
Version	1.0
arg0Desc	The region of the image to scan.
arg1Desc	The horizontal sampling rate, may not be less than 1.
arg2Desc	The vertical sampling rate, may not be less than 1.

Parameter List

Name	**Class Type**	**Default Value**
roi	javax.media.jai.ROI	null
xPeriod	java.lang.Integer	1
yPeriod	java.lang.Integer	1

See Also:
`ROI`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

MeanDescriptor

`public **MeanDescriptor**()`

Constructor.

Method Detail

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.
Overrides:
getParamMinValue in class OperationDescriptorImpl

javax.media.jai.operator

Class MedianFilterDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MedianFilterDescriptor
```

public class **MedianFilterDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "MedianFilter" operation.

The "MedianFilter" operation is a non-linear filter which is useful for removing isolated lines or pixels while preserving the overall appearance of an image. The filter is implemented by moving a mask over the image. For each position of the mask, the center pixel is replaced by the median of the pixel values covered by the mask.

There are several shapes possible for the mask. The MedianFilter operation supports three shapes, as follows:

Square Mask:

```
x x x
x x x
x x x
```

Plus Mask:

```
  x
x x x
  x
```

X Mask:

```
x   x
  x
x   x
```

The Median operation may also be used to compute the "separable median" of a 3x3 or 5x5 region of pixels. The separable median is defined as the median of the medians of each row. For example, if the pixel values in a 3x3 window are equal to:

```
 [ 1 2 3 ]
 [ 5 6 7 ]
 [ 4 8 9 ]
```

then the overall (non-separable) median value is 5, while the separable median is equal to the median of the three row medians: median(1, 2, 3) = 2, median(5, 6, 7) = 6, and median(4, 8, 9) = 8, yielding an overall median of 6. The separable median may be obtained by specifying a mask of type MEDIAN_MASK_SQUARE_SEPARABLE.

Resource List

Name	Value
GlobalName	MedianFilter
LocallName	MedianFilter
Vendor	com.sun.media.jai
Description	Performs median filtering on an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jaiapi/javax.media.jai.operator.MedianFilterDescriptor.html
Version	1.0
arg0Desc	The shape of the mask to be used for Median Filtering.
arg1Desc	The size (width/height) of the mask to be used in Median Filtering.

Parameter List

Name	Class Type	Default Value
maskShape	java.lang.Integer	MEDIAN_MASK_SQUARE

maskSize	java.lang.Integer	3

See Also:
`OperationDescriptor`

Field Detail

MEDIAN_MASK_SQUARE
`public static final int` **`MEDIAN_MASK_SQUARE`**
Square shaped mask.

MEDIAN_MASK_PLUS
`public static final int` **`MEDIAN_MASK_PLUS`**
Plus shaped mask.

MEDIAN_MASK_X
`public static final int` **`MEDIAN_MASK_X`**
X shaped mask.

MEDIAN_MASK_SQUARE_SEPARABLE
`public static final int` **`MEDIAN_MASK_SQUARE_SEPARABLE`**
Separable square mask.

---

resources
`private static final java.lang.String[][]` **`resources`**
The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses
`private static final java.lang.Class[]` **`paramClasses`**
The parameter class list for this operation.

---

paramNames
`private static final java.lang.String[]` **`paramNames`**
The parameter name list for this operation.

---

paramDefaults
`private static final java.lang.Object[]` **`paramDefaults`**
The parameter default value list for this operation.

Constructor Detail

MedianFilterDescriptor
`public` **`MedianFilterDescriptor`**`()`
Constructor for the MedianFilterDescriptor.

Method Detail

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

---

getParamMaxValue

`public java.lang.Number **getParamMaxValue**(int index)`

Returns the maximum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of

javax.media.jai.operator

Class MinDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MinDescriptor
```

public class **MinDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Min" operation.

The Min operation takes two rendered images, and for every pair of pixels, one from each source image of the corresponding position and band, finds the minimum pixel value. No additional parameters are required.

The two sources may have different number of bands and/or data types. By default, the destination image bound is the intersection of the two source image bounds. If the two sources don't intersect, the destination will have a width and a height of 0. The number of bands of the destination image is the same as the least number of bands of the sources, and the data type is the biggest data type of the sources.

The destination pixel values are defined by the pseudocode:

```
 if (srcs[0][x][y][b] < srcs[1][x][y][b]) {
     dst[x][y][b] = srcs[0][x][y][b];
 } else {
     dst[x][y][b] = srcs[1][x][y][b];
 }
```

Resource List

Name	Value
GlobalName	Min
LocalName	Min
Vendor	com.sun.media.jai
Description	Computes the pixel-wise minimum of two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MinDescriptor.html
Version	1.0

See Also:
    `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MinDescriptor

`public MinDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class MultiplyComplexDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MultiplyComplexDescriptor
```

public class **MultiplyComplexDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "MultiplyComplex" operation.

The "MultiplyComplex" operation multiplies two images representing complex data. The source images must each contain an even number of bands with the even-indexed bands (0, 2, ...) representing the real and the odd-indexed bands (1, 3, ...) the imaginary parts of each pixel. The destination image similarly contains an even number of bands with the same interpretation and with contents defined by:

```
 a = src0[x][y][2*k];
 b = src0[x][y][2*k+1];
 c = src1[x][y][2*k];
 d = src1[x][y][2*k+1];

 dst[x][y][2*k]   = a*c - b*d;
 dst[x][y][2*k+1] = a*d + b*c;
```

where 0 <= *k* < numBands/2. With one exception, the number of bands of the destination image is the same as the minimum of the number of bands of the two sources, and the data type is the biggest data type of the sources. The exception occurs when one of the source images has two bands, the other source image has N = 2*K bands where K > 1, and an `ImageLayout` hint is provided containing a destination `SampleModel` which specifies M = 2*L bands for the destination image where L > 1 and L <= K. In this special case each of the first L complex components in the N-band source will be multiplied by the single complex component in the 1-band source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

"MultiplyComplex" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.TRUE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	MultiplyComplex
LocalName	MultiplyComplex
Vendor	com.sun.media.jai
Description	Computes the complex product of two images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MultiplyComplexDescriptor.html
Version	1.0

No parameters are needed for the "MultiplyComplex" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MultiplyComplexDescriptor

```
public MultiplyComplexDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source images each have an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "MultiplyComplex" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class MultiplyComplexPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.MultiplyComplexPropertyGenerator
```

class **MultiplyComplexPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "MultiplyComplex" dynamically.

Constructor Detail

MultiplyComplexPropertyGenerator

```
public MultiplyComplexPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "MultiplyComplex".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator

Class MultiplyConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MultiplyConstDescriptor
```

public class **MultiplyConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "MultiplyConst" operation.

The MultiplyConst operation takes one rendered or renderable image and an array of double constants, and multiplies every pixel of the same band of the source by the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

By default, the destination image bound, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are calculated as:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = srcs[x][y][b]*constants[0];
} else {
    dst[x][y][b] = srcs[x][y][b]*constants[b];
}
```

Resource List

Name	Value
GlobalName	MultiplyConst
LocalName	MultiplyConst
Vendor	com.sun.media.jai
Description	Multiplies a rendered image by constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MultiplyConstDescriptor.html
Version	1.0
arg0Desc	The constants to be multiplied.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

MultiplyConstDescriptor

```
public MultiplyConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameter.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class MultiplyDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.MultiplyDescriptor
```

public class **MultiplyDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Multiply" operation.

The Multiply operation takes two rendered or renderable source images, and multiplies every pair of pixels, one from each source image of the corresponding position and band. No additional parameters are required.

The two source images may have different numbers of bands and data types. By default, the destination image bounds are the intersection of the two source image bounds. If the sources don't intersect, the destination will have a width and height of 0.

The default number of bands of the destination image is equal to the smallest number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types (not necessarily the range of their sums).

As a special case, if one of the source images has N bands (N > 1), the other source has 1 band, and an `ImageLayout` hint is provided containing a destination `SampleModel` with K bands (1 < K <= N), then the single band of the 1-banded source is added to each of the first K bands of the N-band source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
 dst[x][y][dstBand] = clamp(srcs[0][x][y][src0Band] *
                            srcs[1][x][y][src1Band]);
```

Resource List

Name	Value
GlobalName	Multiply
LocalName	Multiply
Vendor	com.sun.media.jai
Description	Multiplies two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/MultiplyDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

MultiplyDescriptor

`public **MultiplyDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator
Class NotDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.NotDescriptor
```

---

public class **NotDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Not" operation.

The Not operation takes one rendered or renderable image, and performs bit-wise logical "not" on every pixel from every band of the source image. No additional parameters are required.

The source image must have an integral data type. By default, the destination image bound, data type, and number of bands are the same as the source image.

The following matrix defines the logical "not" operation.

Logical "not"

src	Result
1	0
0	1

The destination pixel values are defined by the pseudocode:

```
dst[x][y][b] = ~src[x][y][b];
```

Resource List

Name	Value
GlobalName	Not
LocalName	Not
Vendor	com.sun.media.jai
Description	Logically "nots" a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/NotDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

NotDescriptor

```
public NotDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input source.

In addition to the standard checks performed by the superclass method, this method checks that the source image is of integral data type.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator
Class OrConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.OrConstDescriptor
```

public class **OrConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "OrConst" operation.

The Or operation takes one rendered or renderable image and an array of integer constants, and performs a bit-wise logical "or" between every pixel in the same band of the source and the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

The source image must have an integral data type. By default, the destination image bound, data type, and number of bands are the same as the source image.

The following matrix defines the logical "or" operation.

Logical "or"

src	const	Result
0	0	0
0	1	1
1	0	1
1	1	1

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = src[x][y][b] | constants[0];
} else {
    dst[x][y][b] = src[x][y][b] | constants[b];
}
```

Resource List

Name	Value
GlobalName	OrConst
LocalName	OrConst
Vendor	com.sun.media.jai
Description	Logically "ors" a rendered image with constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/OrConstDescriptor.html
Version	1.0
arg0Desc	The constants to logically "or" with.

Parameter List

Name	Class Type	Default Value
constants	int[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

OrConstDescriptor

```
public OrConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer message)
```

Validates the input source and parameter.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an integral data type and that "constants" has length at least 1.

Overrides:
validateArguments in class OperationDescriptorImpl

javax.media.jai.operator

Class OrDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.OrDescriptor
```

public class **OrDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Or" operation.

The Or operation takes two rendered or renderable images, and performs bit-wise logical "or" on every pair of pixels, one from each source image of the corresponding position and band. No additional parameters are required.

Both source images must have integral data types. The two data types may be different.

Unless altered by an `ImageLayout` hint, the destination image bound is the intersection of the two source image bounds. If the two sources don't intersect, the destination will have a width and height of 0. The number of bands of the destination image is equal to the lesser number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types.

The following matrix defines the logical "or" operation.

Logical "or"

src1	src2	Result
0	0	0
0	1	1
1	0	1
1	1	1

The destination pixel values are defined by the pseudocode:

```
dst[x][y][b] = srcs[0][x][y][b] | srcs[1][x][y][b];
```

Resource List

Name	Value
GlobalName	Or
LocalName	Or
Vendor	com.sun.media.jai
Description	Logically "ors" two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/OrDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

OrDescriptor

```
public OrDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source images are of integral data type.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator
Class OrderedDitherDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.OrderedDitherDescriptor
```

public class **OrderedDitherDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "OrderedDither" operation.

The "OrderedDither" operation performs color quantization by finding the nearest color to each pixel in a supplied color cube and "shifting" the resulting index value by a pseudo-random amount determined by the values of a supplied dither mask.

The dither mask is supplied as an array of `KernelJAI` objects the length of which must equal the number of bands in the image. Each element of the array is a `KernelJAI` object which represents the dither mask matrix for the corresponding band. All `KernelJAI` objects in the array must have the same dimensions and contain floating point values greater than or equal to 0.0 and less than or equal to 1.0.

For all integral data types, the source image samples are presumed to occupy the full range of the respective types. For floating point data types it is assumed that the data samples have been scaled to the range [0.0, 1.0].

Resource List

Name	Value
GlobalName	OrderedDither
LocalName	OrderedDither
Vendor	com.sun.media.jai
Description	Performs ordered dither color quantization using a specified color cube and dither mask.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/OrderedDitherDescriptor.html
Version	1.0
arg0Desc	The color cube.
arg1Desc	The dither mask.

Parameter List

Name	Class Type	Default Value
colorMap	javax.media.jai.ColorCube	ColorCube.BYTE_496
ditherMask	javax.media.jai.KernelJAI[]	KernelJAI.DITHER_MASK_443

See Also:
`KernelJAI`, `ColorCube`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter names for the "OrderedDither" operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the "OrderedDither" operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the "OrderedDither" operation.

Constructor Detail

OrderedDitherDescriptor

```
public OrderedDitherDescriptor()
```

Constructor.

Method Detail

isValidColorMap

```
private static boolean isValidColorMap(java.awt.image.RenderedImage sourceImage,
                                       ColorCube colorMap,
                                       java.lang.StringBuffer msg)
```

Method to check the validity of the color map parameter. The supplied color cube must have the same data type and number of bands as the source image.

Parameters:

`sourceImage` - The source image of the operation.
`colorMap` - The color cube.
`msg` - The buffer to which messages should be appended.

Returns:

Whether the color map is valid.

---

isValidDitherMask

```
private static boolean isValidDitherMask(java.awt.image.RenderedImage sourceImage,
                                         KernelJAI[] ditherMask,
                                         java.lang.StringBuffer msg)
```

Method to check the validity of the dither mask parameter. The dither mask is an array of `KernelJAI` objects wherein the number of elements in the array must equal the number of bands in the source image. Furthermore all kernels in the array must have the same width and height. Finally all data elements of all kernels must be greater than or equal to zero and less than or equal to unity.

Parameters:

`sourceImage` - The source image of the operation.
`ditherMask` - The dither mask.
`msg` - The buffer to which messages should be appended.

Returns:

Whether the dither mask is valid.

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameters.

In addition to the standard checks performed by the superclass method, this method checks that "colorMap" and "ditherMask" are valid for the given source image.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator

Class OverlayDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.OverlayDescriptor
```

public class **OverlayDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Overlay" operation.

The Overlay operation takes two rendered or renderable source images, and overlays the second source image on top of the first source image. No additional parameters are required.

The two source images must have the same data type and number of bands. However, their `SampleModel` types may differ. The destination image will always have the same bounding rectangle as the first source image, that is, the image on the bottom, and the same data type and number of bands as the two sources. In case the two sources don't intersect, the destination will be the same as the first source.

The destination pixel values are defined by the pseudocode:

```
 if (srcs[1] contains the point (x, y)) {
     dst[x][y][b] = srcs[1][x][y][b];
 } else {
     dst[x][y][b] = srcs[0][x][y][b];
 }
```

Resource List

Name	Value
GlobalName	Overlay
LocalName	Overlay
Vendor	com.sun.media.jai
Description	Overlays one rendered image on top of another.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/OverlayDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

OverlayDescriptor

`public **OverlayDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source image `SampleModel`s have the same number of bands and transfer types.

Overrides:

validateSources in class OperationDescriptorImpl

javax.media.jai.operator

Class PNGDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PNGDescriptor
```

public class **PNGDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "PNG" operation.

The "PNG" operation reads a standard PNG version 1.1 input stream. The PNG (Portable Network Graphics) specification may be found at `http://www.cdrom.com/pub/png/spec`.

The "PNG" operation implements the entire PNG specification, but provides access only to the final, high-resolution version of interlaced images.

The second parameter contains an instance of `PNGDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The documentation for `PNGDecodeParam` describes the possible output formats of PNG images after decoding.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	PNG
LocalName	PNG
Vendor	com.sun.media.jai
Description	Reads an image from a PNG stream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PNGDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.
arg1Desc	The PNGDecodeParam to use.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.PNGDecodeParam	null

Properties

Property Name	Class	Comment
file_type	String	"PNG v. 1.0"
background_color	java.awt.Color	The suggested background color.
significant_bits	int[]	The number of significant bits stored in the file.
bit_depth	Integer	The bit depth of the file
color_type	String	One of "Grayscale", "Truecolor", "Index", "Grayscale with alpha" or "Truecolor with alpha"
interlace_method	String	"None" or "Adam7"
white_point_x	Float	The CIE X coordinate of the white point, if known.
white_point_y	Float	The CIE Y coordinate of the white point, if known.
red_x	Float	The CIE X coordinate of the red primary, if known.
red_y	Float	The CIE Y coordinate of the red primary, if known.
green_x	Float	The CIE X coordinate of the green primary, if known.
green_y	Float	The CIE Y coordinate of the green primary, if known.
blue_x	Float	The CIE X coordinate of the blue primary, if known.
blue_y	Float	The CIE Y coordinate of the blue primary, if known.
gamma	Float	The image gamma, if known.
x_pixels_per_unit	Integer	The number of horizontal pixels per unit.
y_pixels_per_unit	Integer	The number of vertical pixels per unit.
pixel_aspect_ratio	Float	The width of a pixel divided by its height.
pixel_units	String	"Meters" or `null`
timestamp	java.util.Date	The creation or modification time of the image.
text:*	String	The value of a tEXt chunk.
ztext:*	String	The value of a zTXt chunk (not yet implemented).
chunk:*	byte[]	The contents of any non-standard chunks.

See Also:
`PNGDecodeParam`, `SeekableStream`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "PNG" operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "PNG" operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "PNG" operation.

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "PNG" operation.

Constructor Detail

PNGDescriptor

`public **PNGDescriptor**()`

Constructor.

javax.media.jai.operator
Class PNMDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PNMDescriptor
```

public class **PNMDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "PNM" operation.

The "PNM" operation reads a standard PNM file, including PBM, PGM, and PPM images of both ASCII and raw formats. It stores the image data into an appropriate `SampleModel`,

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	PNM
LocalName	PNM
Vendor	com.sun.media.jai
Description	Reads a standard PNM file.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PNMDescriptor.html
Version	1.0
arg0Desc	A SeekableStream representing the PNM file.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT

See Also:
`SeekableStream`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "PNM" operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter names for the "PNM" operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class types for the "PNM" operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default values for the "PNM" operation.

Constructor Detail

PNMDescriptor

```
public PNMDescriptor()
```

Constructor.

javax.media.jai.operator

Class PatternDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PatternDescriptor
```

public class **PatternDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Pattern" operation.

The "Pattern" operation defines a tiled image consisting of a repeated pattern. The width and height of the destination image must be specified. The tileWidth and tileHeight are equal to pattern's width and height. Each tile of the destination image will be defined by a reference to a shared instance of the pattern.

Resource List

Name	Value
GlobalName	pattern
LocalName	pattern
Vendor	com.sun.media.jai
Description	Defines an image with a repeated pattern.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PatternDescriptor.html
Version	1.0
arg0Desc	The width of the image in pixels.
arg1Desc	The height of the image in pixels.

Parameter List

Name	Class Type	Default Value
width	java.lang.Integer	NO_PARAMETER_DEFAULT
height	java.lang.Integer	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation for the "Pattern" operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

PatternDescriptor

`public PatternDescriptor()`

Constructor.

Method Detail

getParamMinValue

`public java.lang.Number getParamMinValue(int index)`

Overrides:

getParamMinValue in class OperationDescriptorImpl

javax.media.jai.operator
Class PeriodicShiftDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PeriodicShiftDescriptor
```

public class **PeriodicShiftDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "PeriodicShift" operation.

The destination image of the "PeriodicShift" operation is the infinite periodic extension of the source image with horizontal and vertical periods equal to the image width and height, respectively, shifted by a specified amount along each axis and clipped to the bounds of the source image. Thus for each band *b* the destination image sample at location *(x,y)* is defined by:

```
if(x < width - shiftX) {
    if(y < height - shiftY) {
        dst[x][y][b] = src[x + shiftX][y + shiftY][b];
    } else {
        dst[x][y][b] = src[x + shiftX][y - height + shiftY][b];
    }
} else {
    if(y < height - shiftY) {
        dst[x][y][b] = src[x - width + shiftX][y + shiftY][b];
    } else {
        dst[x][y][b] = src[x - width + shiftX][y - height + shiftY][b];
    }
}
```

where *shiftX* and `shiftY` denote the translation factors along the *X* and *Y* axes, respectively.

Resource List

Name	Value
GlobalName	PeriodicShift
LocalName	PeriodicShift
Vendor	com.sun.media.jai
Description	Computes the periodic translation of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PeriodicShiftDescriptor.html
Version	1.0
arg0Desc	The displacement in the X direction.
arg1Desc	The displacement in the Y direction.

Parameter List

Name	Class Type	Default Value
shiftX	java.lang.Integer	sourceWidth/2
shiftY	java.lang.Integer	sourceHeight/2

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

PeriodicShiftDescriptor

`public PeriodicShiftDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "shiftX" and "shiftY" are between 0 and the source image width and height, respectively.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator

Class PhaseDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PhaseDescriptor
```

public class **PhaseDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Phase" operation.

The "Phase" operation computes the phase angle of each pixel of a complex image. The source image must have an even number of bands, with the even bands (0, 2, ...) representing the real parts and the odd bands (1, 3, ...) the imaginary parts of each complex pixel. The destination image has at most half the number of bands of the source image with each sample in a pixel representing the phase angle of the corresponding complex source sample. The angular values of the destination image are defined for a given sample by the pseudocode:

```
dst[x][y][b] = Math.atan2(src[x][y][2*b+1], src[x][y][2*b])
```

where the number of bands *b* varies from zero to one less than the number of bands in the destination image.

For integral image datatypes, the result will be rounded and scaled so the the "natural" arctangent range [-PI, PI) is remapped into the range [0, MAX_VALUE); the result for floating point image datatypes is the value returned by the atan2() method.

"Phase" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.FALSE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	Phase
LocalName	Phase
Vendor	com.sun.media.jai
Description	Computes the phase angle of each pixel of an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PhaseDescriptor.html
Version	1.0

No parameters are needed for the "Phase" operation.

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

PhaseDescriptor

`public PhaseDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input source.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an even number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "Phase" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class PhasePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.PhasePropertyGenerator
```

class **PhasePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Phase" dynamically.

Constructor Detail

PhasePropertyGenerator

`public PhasePropertyGenerator()`

Constructor.

Method Detail

getPropertyNames

`public java.lang.String[] getPropertyNames()`

Returns the valid property names for the operation "Phase".
Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

javax.media.jai.operator
Class PiecewiseDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PiecewiseDescriptor
```

public class **PiecewiseDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Piecewise" operation.

The "Piecewise" operation performs a piecewise linear mapping of the pixel values of an image. The piecewise linear mapping is described by a set of breakpoints which are provided as an array of the form

`float breakPoints[N][2][numBreakPoints]`

where the value of *N* may be either unity or the number of bands in the source image. If *N* is unity then the same set of breakpoints will be applied to all bands in the image. The abscissas of the supplied breakpoints must be monotonically increasing.

The pixel values of the destination image are defined by the pseudocode:

```
if (src[x][y][b] < breakPoints[b][0][0]) {
    dst[x][y][b] = breakPoints[b][1][0]);
} else if (src[x][y][b] > breakPoints[b][0][numBreakPoints-1]) {
    dst[x][y][b] = breakPoints[b][1][numBreakPoints-1]);
} else {
    int i = 0;
    while(breakPoints[b][0][i+1] < src[x][y][b]) {
        i++;
    }
    dst[x][y][b] = breakPoints[b][1][i] +
                       (src[x][y][b] - breakPoints[b][0][i])*
                       (breakPoints[b][1][i+1] - breakPoints[b][1][i])/
                       (breakPoints[b][0][i+1] - breakPoints[b][0][i]);
}
```

Resource List

Name	Value
GlobalName	Piecewise
LocalName	Piecewise
Vendor	com.sun.media.jai
Description	Applies a piecewise pixel value mapping.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PiecewiseDescriptor.html
Version	1.0
arg0Desc	The breakpoint array.

Parameter List

Name	Class Type	Default Value
breakPoints	float[][][]	NO_PARAMETER_DEFAULT

See Also:
`DataBuffer`, `ImageLayout`, `OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

PiecewiseDescriptor

```
public PiecewiseDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer msg)
```

Validates the input source and parameter.

In addition to the standard checks performed by the superclass method, this method checks that the number of bands in "breakPoints" is either 1 or the number of bands in the source image, the second breakpoint array dimension is 2, the third dimension is the same for abscissas and ordinates, and that the absicssas are monotonically increasing.

Overrides:

validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class PolarToComplexDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.PolarToComplexDescriptor
```

public class **PolarToComplexDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "PolarToComplex" operation.

The "PolarToComplex" operation creates an image with complex-valued pixels from two images the respective pixel values of which represent the magnitude (modulus) and phase of the corresponding complex pixel in the destination image. The source images should have the same number of bands. The first source image contains the magnitude values and the second source image the phase values. The destination will have twice as many bands with the even-indexed bands (0, 2, ...) representing the real and the odd-indexed bands (1, 3, ...) the imaginary parts of each pixel. The pixel values of the destination image are defined for a given complex sample by the pseudocode:

```
 dst[x][y][2*b]   = src0[x][y][b]*Math.cos(src1[x][y][b])
 dst[x][y][2*b+1] = src0[x][y][b]*Math.sin(src1[x][y][b])
```

where the index *b* varies from zero to one less than the number of bands in the source images.

For phase images with integral data type, it is assumed that the actual phase angle is scaled from the range [-PI, PI] to the range [0, MAX_VALUE] where MAX_VALUE is the maximum value of the data type in question.

"PolarToComplex" defines a PropertyGenerator that sets the "COMPLEX" property of the image to `java.lang.Boolean.TRUE`, which may be retrieved by calling the `getProperty()` method with "COMPLEX" as the property name.

Resource List

Name	Value
GlobalName	PolarToComplex
LocalName	PolarToComplex
Vendor	com.sun.media.jai
Description	Computes a complex image from a magnitude and a phase image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/PolarToComplexDescriptor.html
Version	1.0

No parameters are needed for the "PolarToComplex" operation.

See Also:
`OperationDescriptor`, `PhaseDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

PolarToComplexDescriptor

`public PolarToComplexDescriptor()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source images have the same number of bands.

Overrides:

validateSources in class OperationDescriptorImpl

---

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "Conjugate" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class PolarToComplexPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.PolarToComplexPropertyGenerator
```

---

class **PolarToComplexPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "PolarToComplex" dynamically.

Constructor Detail

PolarToComplexPropertyGenerator

`public PolarToComplexPropertyGenerator()`

Constructor.

Method Detail

getPropertyNames

`public java.lang.String[] getPropertyNames()`

Returns the valid property names for the operation "PolarToComplex".
Specified by:
getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

javax.media.jai.operator
Class RenderableDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.RenderableDescriptor
```

public class **RenderableDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Renderable" operation.

In renderable image mode the "Renderable" operation produces from a `RenderedImage` source a `RenderableImage` consisting of a "pyramid" of `RenderedImage`s at progressively lower resolutions. This operation does not support rendered image mode.

Lower resolution images are produced by invoking the chain of operations specified via the "downSampler" parameter on the image at the next higher resolution level of the pyramid. The "downSampler" operation chain must adhere to the specifications described for the constructors of the `ImageMIPMap` class which accept this type of parameter. The "downSampler" operation chain must reduce the image width and height at each level of the pyramid. The default operation chain for "downSampler" is a low pass filtering implemented using a 5x5 separable kernel derived from the one-dimensional kernel

```
 [0.05 0.25 0.40 0.25 0.05]
```

followed by downsampling by 2.

The number of levels in the pyramid will be such that the maximum of the width and height of the lowest resolution pyramid level is less than or equal to the value of the "maxLowResDim" parameter which must be positive.

The minimum X and Y coordinates and height in rendering-independent coordinates are supplied by the parameters "minX", "minY", and "height", respectively. The value of "height" must be positive. It is not necessary to supply a value for the rendering-independent width as this is derived by multiplying the supplied height by the aspect ratio (width divided by height) of the source `RenderedImage`.

Resource List

Name	Value
GlobalName	Renderable
LocalName	Renderable
Vendor	com.sun.media.jai
Description	Produces a RenderableImage from a RenderedImage.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/RenderableDescriptor.html
Version	1.0
arg0Desc	The operation chain used to derive the lower resolution images.
arg1Desc	The maximum dimension of the lowest resolution pyramid level.
arg2Desc	The minimum rendering-independent X coordinate of the destination.
arg3Desc	The minimum rendering-independent Y coordinate of the destination.
arg4Desc	The rendering-independent height.

Parameter List

Name	Class Type	Default Value
downSampler	RenderedOp	null
maxLowResDim	Integer	64
minX	Float	0.0F
minY	Float	0.0F
height	Float	1.0F

See Also:
`ImageMIPMap`, `OperationDescriptor`

Field Detail

DEFAULT_KERNEL_1D

`private static final float[] DEFAULT_KERNEL_1D`

---

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

RenderableDescriptor

`public RenderableDescriptor()`

Constructor.

Method Detail

isRenderedSupported

`public boolean isRenderedSupported()`

Indicates that rendered operation is supported.
Overrides:
isRenderedSupported in class OperationDescriptorImpl

isRenderableSupported

```
public boolean isRenderableSupported()
```

Indicates that renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates input parameters in the renderable layer.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class RescaleDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.RescaleDescriptor
```

public class **RescaleDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Rescale" operation.

The "Rescale" operation takes a rendered or renderable source image and maps the pixel values of an image from one range to another range by multiplying each pixel value by one of a set of constants and then adding another constant to the result of the multiplication. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band. There must be at least one entry in each of the contants and offsets arrays.

The destination pixel values are defined by the pseudocode:

```
 constant = (constants.length < dstNumBands) ?
            constants[0] : constants[b];
 offset = (offsets.length < dstNumBands) ?
          offsets[0] : offsets[b];

 dst[x][y][b] = src[x][y][b]*constant + offset;
```

The pixel arithmetic is performed using the data type of the destination image. By default, the destination will have the same data type as the source image unless an `ImageLayout` containing a `SampleModel` with a different data type is supplied as a rendering hint.

Resource List

Name	Value
GlobalName	Rescale
LocalName	Rescale
Vendor	com.sun.media.jai
Description	Maps the pixels values of an image from one range to another range.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/RescaleDescriptor.html
Version	1.0
arg0Desc	The per-band constants to multiply by.
arg1Desc	The per-band offsets to be added.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT
offsets	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

RescaleDescriptor

```
public RescaleDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" and "offsets" arrays are each at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class RotateDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.RotateDescriptor
```

public class **RotateDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Rotate" operation.

The "Rotate" operation rotates an image about a given point by a given angle, specified in radians. The origin defaults to (0, 0).

"Rotate" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	Rotate
LocalName	Rotate
Vendor	com.sun.media.jai
Description	Rotate an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/RotateDescriptor.html
Version	1.0
arg0Desc	The X origin to rotate about.
arg1Desc	The Y origin to rotate about.
arg2Desc	The rotation angle in radians.
arg3Desc	The interpolation method.

Parameter List

Name	Class Type	Default Value
xOrigin	java.lang.Float	0.0F
yOrigin	java.lang.Float	0.0F
angle	java.lang.Float	NO_PARAMETER_DEFAULT
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`Interpolation`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for the "Rotate" operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "Rotate" operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "Rotate" operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "Rotate" operation.

Constructor Detail

RotateDescriptor

`public **RotateDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Rotate" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class RotatePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.RotatePropertyGenerator
```

class **RotatePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Rotate" dynamically.

Constructor Detail

RotatePropertyGenerator

```
public RotatePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Rotate".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator

Class ScaleDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ScaleDescriptor
```

public class **ScaleDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Scale" operation.

The "Scale" operation translates and resizes an image. For each pixel (x, y) of the destination, the source value at the fractional subpixel position ((x - xTrans)/xScale, (y - yTrans)/yScale) is constructed by means of an Interpolation object and written to the destination.

When applying scale factors of scale_x, scale_y to a source image with width of src_width and height of src_height, the resulting image is defined to have the following dimensions: `dst_width = src_width * scale_x dst_height = src_height * scale_y`

When interpolations which require padding the source such as Bilinear or Bicubic interpolation are specified, the source needs to be extended such that it has the extra pixels needed to compute all the destination pixels. This extension is performed via the `BorderExtender` class. The type of Border Extension can be specified as a `RenderingHint` to the `JAI.create` method.

If no Border Extension is specified, the source will not be extended. The scaled image size is still calculated according to the formula specified above. However since there isn’t enough source to compute all the destination pixels, only that subset of the destination image’s pixels, which can be computed, will be written in the destination. The rest of the destination will not be written.

Specifying a scale factor of greater than 1 increases the size of the image, specifying a scale factor between 0 and 1 (non-inclusive) decreases the size of an image. An IllegalArgumentException will be thrown if the specified scale factors are negative or equal to zero.

"Scale" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	Scale
LocalName	Scale
Vendor	com.sun.media.jai
Description	Resizes an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ScaleDescriptor.html
Version	1.0
arg0Desc	The X scale factor.
arg1Desc	The Y scale factor.
arg2Desc	The X translation.
arg3Desc	The Y translation.
arg4Desc	The interpolation method for resampling.

Parameter List

Name	Class Type	Default Value
xScale	java.lang.Float	1.0F
yScale	java.lang.Float	1.0F
xTrans	java.lang.Float	0.0F
yTrans	java.lang.Float	0.0F
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`Interpolation`, `BorderExtender`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class list for this operation.

paramNames

`private static final java.lang.String[] `**`paramNames`**

The parameter name list for this operation.

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default value list for this operation.

Constructor Detail

ScaleDescriptor

`public `**`ScaleDescriptor`**`()`

Constructor.

Method Detail

isRenderableSupported

`public boolean `**`isRenderableSupported`**`()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

getPropertyGenerators

```
public PropertyGenerator[] getPropertyGenerators()
```

Returns an array of `PropertyGenerators` implementing property inheritance for the "Scale" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates the input parameters.

In addition to the standard checks performed by the superclass method, this method checks that "xScale" and "yScale" are both greater than 0.

Overrides:

validateParameters in class OperationDescriptorImpl

---

getParamMinValue

```
public java.lang.Number getParamMinValue(int index)
```

Returns the minimum legal value of a specified numeric parameter for this operation.

For the minimum value of "xScale" and "yScale", this method returns 0. However, the scale factors must be a positive floating number and can not be 0.

Overrides:

getParamMinValue in class OperationDescriptorImpl

javax.media.jai.operator

Class ScalePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.ScalePropertyGenerator
```

---

class **ScalePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Scale" dynamically.

Constructor Detail

ScalePropertyGenerator

```
public ScalePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Scale".

Specified by:
getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

javax.media.jai.operator

Class ShearDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ShearDescriptor
```

public class **ShearDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Shear" operation.

The "Shear" operation shears an image either horizontally or vertically. For each pixel (x, y) of the destination, the source value at the fractional subpixel position (x’, y’) is constructed by means of an Interpolation object and written to the destination.

If the "shearDir" parameter is equal to SHEAR_HORIZONTAL then `x’ = (x - xTrans - y*shear)` and `y’ = y`. If the "shearDir" parameter is equal to SHEAR_VERTICAL then `x’ = x` and `y’ = (y - yTrans - x*shear)`.

"Shear" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	shear
LocalName	shear
Vendor	com.sun.media.jai
Description	Shears an image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ShearDescriptor.html
Version	1.0
arg0Desc	The shear value.
arg1Desc	The shear direction.
arg2Desc	The X translation.
arg3Desc	The Y translation.
arg4Desc	The interpolation method for resampling.

Parameter List

Name	Class Type	Default Value
shear	java.lang.Float	NO_PARAMETER_DEFAULT
shearDir	java.lang.Integer	NO_PARAMETER_DEFAULT
xTrans	java.lang.Float	0.0F
yTrans	java.lang.Float	0.0F
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`Interpolation`, `OperationDescriptor`

Field Detail

SHEAR_HORIZONTAL

`public static final int` **`SHEAR_HORIZONTAL`**

SHEAR_VERTICAL

`public static final int` **`SHEAR_VERTICAL`**

---

resources

`private static final java.lang.String[][]` **`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "Shear" operation.

---

paramNames

`private static final java.lang.String[]` **`paramNames`**

The parameter names for the "Shear" operation.

---

paramClasses

`private static final java.lang.Class[]` **`paramClasses`**

The parameter class types for the "Shear" operation.

---

paramDefaults

`private static final java.lang.Object[]` **`paramDefaults`**

The parameter default values for the "Shear" operation.

Constructor Detail

ShearDescriptor

`public` **`ShearDescriptor`**`()`

Constructor.

Method Detail

getPropertyGenerators

`public PropertyGenerator[]` **`getPropertyGenerators`**`()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Shear" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator
Class ShearPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.ShearPropertyGenerator
```

---

class **ShearPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Shear" dynamically.

Constructor Detail

ShearPropertyGenerator

```
public ShearPropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Shear".
Specified by:
getPropertyNames in interface PropertyGenerator

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

---

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.
Specified by:
getProperty in interface PropertyGenerator
Parameters:
`name` - Property name.

javax.media.jai.operator

Class StreamDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.StreamDescriptor
```

public class **StreamDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Stream" operation.

The Stream operation produces an image by decoding data from a `SeekableStream`. The allowable formats are those registered with the `com.sun.media.jai.codec.ImageCodec` class.

The allowable formats are those registered with the `com.sun.media.jai.codec.ImageCodec` class.

The second parameter contains an instance of `ImageDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	stream
LocalName	stream
Vendor	com.sun.media.jai
Description	Reads an image from a SeekableStream.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/StreamDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.
arg1Desc	The ImageDecodeParam to use.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.ImageDecodeParam	null

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "Stream" operation.

paramNames

`private static final java.lang.String[] `**`paramNames`**

The parameter names for the "Stream" operation.

---

paramClasses

`private static final java.lang.Class[] `**`paramClasses`**

The parameter class types for the "Stream" operation.

---

paramDefaults

`private static final java.lang.Object[] `**`paramDefaults`**

The parameter default values for the "Stream" operation.

Constructor Detail

StreamDescriptor

`public `**`StreamDescriptor`**`()`

Constructor.

javax.media.jai.operator
Class SubtractConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.SubtractConstDescriptor
```

public class **SubtractConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "SubtractConst" operation.

The SubtractConst operation takes one rendered or renderable source image and an array of double constants, and subtracts a constant from every pixel of its corresponding band of the source. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

By default, the destination image bounds, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = src[x][y][b] - constants[0];
} else {
    dst[x][y][b] = src[x][y][b] - constants[b];
}
```

Resource List

Name	Value
GlobalName	SubtractConst
LocalName	SubtractConst
Vendor	com.sun.media.jai
Description	Subtracts constants from a rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/SubtractConstDescriptor.html
Version	1.0
arg0Desc	The constants to be subtracted.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

SubtractConstDescriptor

```
public SubtractConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameter.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class SubtractDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.SubtractDescriptor
```

public class **SubtractDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Subtract" operation.

The Subtract operation takes two rendered images, and for every pair of pixels, one from each source image of the corresponding position and band, subtracts the pixel from the second source from the pixel from the first source. No additional parameters are required.

The two source images may have different numbers of bands and data types. By default, the destination image bounds are the intersection of the two source image bounds. If the sources don't intersect, the destination will have a width and height of 0.

The default number of bands of the destination image is equal to the smallest number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types (not necessarily the range of their sums).

As a special case, if one of the source images has N bands (N > 1), the other source has 1 band, and an `ImageLayout` hint is provided containing a destination `SampleModel` with K bands (1 < K <= N), then the single band of the 1-banded source is subtracted from or into each of the first K bands of the N-band source.

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
 dst[x][y][dstBand] = clamp(srcs[0][x][y][src0Band] -
                            srcs[1][x][y][src1Band]);
```

Resource List

Name	Value
GlobalName	Subtract
LocalName	Subtract
Vendor	com.sun.media.jai
Description	Subtracts one rendered image from another rendered image.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/SubtractDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

SubtractDescriptor

`public **SubtractDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

javax.media.jai.operator

Class SubtractFromConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.SubtractFromConstDescriptor
```

---

public class **SubtractFromConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "SubtractFromConst" operation.

The SubtractFromConst operation takes one rendered or renderable image and an array of double constants, and subtracts every pixel of the same band of the source from the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

By default, the destination image bound, data type, and number of bands are the same as the source image. If the result of the operation underflows/overflows the minimum/maximum value supported by the destination data type, then it will be clamped to the minimum/maximum value respectively.

The destination pixel values are defined by the pseudocode:

```
 if (constants.length < dstNumBands) {
     dst[x][y][b] = constants[0] - src[x][y][b];
 } else {
     dst[x][y][b] = constants[b] - src[x][y][b];
 }
```

Resource List

Name	Value
GlobalName	SubtractFromConst
LocalName	SubtractFromConst
Vendor	com.sun.media.jai
Description	Subtracts a rendered image from constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/SubtractFromConstDescriptor.html
Version	1.0
arg0Desc	The constants to be subtracted from.

Parameter List

Name	Class Type	Default Value
constants	double[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

```
private static final java.lang.Class[] paramClasses
```

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

---

paramNames

```
private static final java.lang.String[] paramNames
```

The parameter name list for this operation.

---

paramDefaults

```
private static final java.lang.Object[] paramDefaults
```

The parameter default value list for this operation.

Constructor Detail

SubtractFromConstDescriptor

```
public SubtractFromConstDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer message)
```

Validates the input parameter.

In addition to the standard checks performed by the superclass method, this method checks that the length of the "constants" array is at least 1.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator

Class TIFFDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.TIFFDescriptor
```

public class **TIFFDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "TIFF" operation.

The "TIFF" operation reads TIFF 6.0 data from an `SeekableStream`.

TIFF version 6.0 was finalized in June, 1992. Since that time there have been two technical notes extending the specification.

JAI's property inheritance mechanism has been designed to allow the tag information of a TIFF file, or other tagged file, to be made available to applications in a straightforward way. User code may additionally supply `PropertyGenerator` objects that allow tag information to be propagated through chains of operations.

TIFF extensions, such as GeoTIFF (see The GeoTIFF Web Page), operate by defining additional private tags, usually referenced by a small number of globally registered TIFF tags. JAI allows such tags to be decoded into first-class tags by adding suitable PropertyGenerator objects.

The TIFF format consists of a short header that points to a linked list of Image File Directories (IFDs). An IFD is essentially a list of tags. The TIFFDirectory class encapsulates the set of common operations performed on an IFD. Each tag has a numeric value, a datatype, and a byte offset at which the tag's data may be found. This mechanism allows TIFF files to contain multiple images, each in its own IFD, and to order their contents flexibly since (apart from the header) nothing is required to appear at a fixed offset.

The following properties are provided by the TIFF reader from standard TIFF tags, shown here with their numerical values:

Name	Value	Name	Value	Name	Value
TIFF_NewSubfileType	254	TIFF_YPosition	287	TIFF_TargetPrinter	337
TIFF_SubfileType	255	TIFF_FreeOffsets	288	TIFF_ExtraSamples	338
TIFF_ImageWidth	256	TIFF_FreeByteCounts	289	TIFF_SampleFormat	339
TIFF_ImageLength	257	TIFF_GrayResponseUnit	290	TIFF_SMinSampleValue	340
TIFF_BitsPerSample	258	TIFF_GrayResponseCurve	291	TIFF_SMaxSampleValue	341
TIFF_Compression	259	TIFF_T4Options	292	TIFF_TransferRange	342
TIFF_PhotometricInterpretation	262	TIFF_T6Options	293	TIFF_JPEGProc	512
TIFF_Thresholding	263	TIFF_ResolutionUnit	296	TIFF_JPEGInterchangeFormat	513
TIFF_CellWidth	264	TIFF_PageNumber	297	TIFF_JPEGInterchangeFormatLngth	514
TIFF_CellLength	265	TIFF_Software	301	TIFF_JPEGRestartInterval	515
TIFF_FillOrder	266	TIFF_Software	305	TIFF_JPEGLosslessPredictors	517
TIFF_DocumentName	269	TIFF_DateTime	306	TIFF_JPEGPointTransforms	518
TIFF_ImageDescription	270	TIFF_Artist	315	TIFF_QTables	519
TIFF_Make	271	TIFF_HostComputer	316	TIFF_DCTables	520
TIFF_Model	272	TIFF_Predictor	317	TIFF_ACTables	521
TIFF_StripOffsets	273	TIFF_WhitePoint	318	TIFF_YCbCrCoefficients	529
TIFF_Orientation	274	TIFF_PrimaryChromaticities	319	TIFF_YCbCrSubSampling	530
TIFF_SamplesPerPixel	277	TIFF_ColorMap	320	TIFF_YCbCrPositioning	531
TIFF_RowsPerStrip	278	TIFF_HalftoneHints	321	TIFF_ReferenceBlackWhite	532
TIFF_StripByteCounts	279	TIFF_TileWidth	322	TIFF_Copyright	33432
TIFF_MinSampleValue	280	TIFF_TileLength	323	TIFF_ModelPixelScaleTag	33550
TIFF_MaxSampleValue	281	TIFF_TileOffsets	324	TIFF_ModelTransformationTag	33920
TIFF_XResolution	282	TIFF_TileByteCounts	325	TIFF_ModelTiepointTag	33922
TIFF_YResolution	283	TIFF_InkSet	332	TIFF_GeoKeyDirectoryTag	34735
TIFF_PlanarConfiguration	284	TIFF_InkNames	333	TIFF_GeoDoubleParamsTag	34736
TIFF_PageName	285	TIFF_NumberOfInks	334	TIFF_GeoAsciiParamsTag	34737
TIFF_XPosition	286	TIFF_DotRange	336	TIFF_Private	n/a

Non-standard tags are handled in TIFF by the inclusion of a tag with a number of 32768 or above. Any tag from this range is treated as an uninterpreted `TIFF_Private` tag, and its raw data is made available as a property of class TIFFField. A user-specified PropertyGenerator can then interpret this directory to produce comprehensible values.

Some TIFF extensions make use of a mechanism known as "private IFDs." A private IFD is one that is not referenced by the standard linked list of IFDs that starts in the file header. To a standard TIFF reader, it appears as an unreferenced area in the file. However, the byte offset of the private IFD is stored as the value of a private tag, allowing readers that understand the tag to locate and interpret the IFD.

The second parameter contains an instance of `TIFFDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	TIFF
LocalName	TIFF
Vendor	com.sun.media.jai
Description	Reads a TIFF 6.0 file.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/TIFFDescriptor.html
Version	1.0
arg0Desc	The SeekableStream to read from.
arg1Desc	The TIFFDecodeParam to use.

Parameter List

Name	Class Type	Default Value
stream	com.sun.media.jai.codec.SeekableStream	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.TIFFDecodeParam	null

See Also:
`SeekableStream`, `TIFFDecodeParam`, `TIFFDirectory`, `TIFFField`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for the "TIFF" operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "TIFF" operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "TIFF" operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "TIFF" operation.

Constructor Detail

TIFFDescriptor

`public **TIFFDescriptor**()`

Constructor.

javax.media.jai.operator

Class ThresholdDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.ThresholdDescriptor
```

public class **ThresholdDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Threshold" operation.

The Threshold operation takes one rendered image, and maps all the pixels of this image whose value falls within a specified range to a specified constant. The range is specified by a low value and a high value.

If the number of elements supplied via the "high", "low", and "constants" arrays are less than the number of bands of the source image, then the element from entry 0 is applied to all the bands. Otherwise, the element from a different entry is applied to its corresponding band.

The destination pixel values are defined by the pseudocode:

```
lowVal = (low.length < dstNumBands) ?
         low[0] : low[b];
highVal = (high.length < dstNumBands) ?
          high[0] : high[b];
const = (constants.length < dstNumBands) ?
          constants[0] : constants[b];

if (src[x][y][b] >= lowVal && src[x][y][b] <= highVal) {
    dst[x][y][b] = const;
} else {
    dst[x][y][b] = src[x][y][b];
}
```

Resource List

Name	Value
GlobalName	Threshold
LocalName	Threshold
Vendor	com.sun.media.jai
Description	Maps the pixels whose value falls between a low value and a high value to a constant.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/ThresholdDescriptor.html
Version	1.0
arg0Desc	The low value.
arg1Desc	The high value.
arg2Desc	The constant the pixels are mapped to.

Parameter List

Name	Class Type	Default Value
low	double[]	NO_PARAMETER_DEFAULT
high	double[]	NO_PARAMETER_DEFAULT
constants	double[]	NO_PARAMETER_DEFAULT

See Also:

`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

ThresholdDescriptor

`public **ThresholdDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateParameters

```
protected boolean validateParameters(java.awt.image.renderable.ParameterBlock args,
                                     java.lang.StringBuffer msg)
```

Validates input parameters.

Overrides:

validateParameters in class OperationDescriptorImpl

javax.media.jai.operator
Class TranslateDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.TranslateDescriptor
```

public class **TranslateDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Translate" operation.

The "Translate" operation copies an image to a new location in the plane.

For each pixel (x, y) of the destination, the source value at the fractional subpixel position (x - xTrans, y - yTrans) is constructed by means of an Interpolation object and written to the destination. If both xTrans and yTrans are integral, the operation simply "wraps" its source image to change the image's position in the coordinate plane.

"Translate" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	Translate
LocalName	Translate
Vendor	com.sun.media.jai
Description	Moves an image to a new location.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/TranslateDescriptor.html
Version	1.0
arg0Desc	The displacement in X direction.
arg1Desc	The displacement in Y direction.
arg2Desc	The interpolation method.

Parameter List

Name	Class Type	Default Value
xTrans	java.lang.Float	0.0F
yTrans	java.lang.Float	0.0F
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`Interpolation`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for the "Translate" operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter names for the "Translate" operation.

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class types for the "Translate" operation.

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default values for the "Translate" operation.

Constructor Detail

TranslateDescriptor

`public TranslateDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

getPropertyGenerators

`public PropertyGenerator[] getPropertyGenerators()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Translate" operation

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator
Class TranslatePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.TranslatePropertyGenerator
```

class **TranslatePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Translate" dynamically.

Constructor Detail

TranslatePropertyGenerator

```
public TranslatePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Translate".

Specified by:

getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.

Specified by:

getProperty in interface PropertyGenerator

Parameters:

`name` - Property name.

javax.media.jai.operator
Class TransposeDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.TransposeDescriptor
```

public class **TransposeDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Transpose" operation.

The "Transpose" operation performs the following operations:

- Flip an image across an imaginary horizontal line that runs through the center of the image (FLIP_VERTICAL).
- Flip an image across an imaginary vertical line that runs through the center of the image (FLIP_HORIZONTAL).
- Flip an image across its main diagonal that runs from the upper left to the lower right corner (FLIP_DIAGONAL).
- Flip an image across its main antidiagonal that runs from the upper right to the lower left corner(FLIP_ANTIDIAGONAL).
- Rotate an image clockwise by 90, 180, or 270 degrees (ROTATE_90, ROTATE_180, ROTATE_270).

In all cases, the resulting image will have the same origin (as defined by the return values of its `getMinX()` and `getMinY()` methods) as the source image.

"Transpose" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	transpose
LocalName	transpose
Vendor	com.sun.media.jai
Description	Reflects an image in a specified direction or rotates an image in multiples of 90 degrees.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/TransposeDescriptor.html
Version	1.0
arg0Desc	The type of flip operation to be performed.

Parameter List

Name	Class Type	Default Value
type	java.lang.Integer	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

FLIP_VERTICAL

`public static final int` **`FLIP_VERTICAL`**

FLIP_HORIZONTAL

`public static final int **FLIP_HORIZONTAL**`

FLIP_DIAGONAL

`public static final int **FLIP_DIAGONAL**`

FLIP_ANTIDIAGONAL

`public static final int **FLIP_ANTIDIAGONAL**`

ROTATE_90

`public static final int **ROTATE_90**`

ROTATE_180

`public static final int **ROTATE_180**`

ROTATE_270

`public static final int **ROTATE_270**`

---

resources

`private static final java.lang.String[][] **resources**`

The resource strings that provide the general documentation and specify the parameter list for this operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class list for this operation.

---

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter name list for this operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default value list for this operation.

Constructor Detail

TransposeDescriptor

`public **TransposeDescriptor**()`

Constructor.

Method Detail

isRenderableSupported

`public boolean **isRenderableSupported**()`

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Transpose" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

getParamMinValue

`public java.lang.Number **getParamMinValue**(int index)`

Returns the minimum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMinValue in class OperationDescriptorImpl

getParamMaxValue

`public java.lang.Number **getParamMaxValue**(int index)`

Returns the maximum legal value of a specified numeric parameter for this operation.

Overrides:

getParamMaxValue in class OperationDescriptorImpl

javax.media.jai.operator

Class TransposePropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.TransposePropertyGenerator
```

class **TransposePropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Transpose" dynamically.

Constructor Detail

TransposePropertyGenerator

```
public TransposePropertyGenerator()
```

Constructor.

Method Detail

getPropertyNames

```
public java.lang.String[] getPropertyNames()
```

Returns the valid property names for the operation "Transpose".

Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

javax.media.jai.operator
Class URLDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.URLDescriptor
```

public class **URLDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "URL" operation.

The URL operation creates an output image whose source is specified by a Uniform Resource Locator (URL).

The allowable formats are those registered with the `com.sun.media.jai.codec.ImageCodec` class.

The second parameter contains an instance of `ImageDecodeParam` to be used during the decoding. It may be set to `null` in order to perform default decoding, or equivalently may be omitted.

The classes in the `com.sun.media.jai.codec` package are not a committed part of the JAI API. Future releases of JAI will make use of new classes in their place. This class will change accordingly.

Resource List

Name	Value
GlobalName	fileload
LocalName	fileload
Vendor	com.sun.media.jai
Description	Reads an image from a file.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/URLDescriptor.html
Version	1.0
arg0Desc	The path of the file to read from.
arg1Desc	The ImageDecodeParam to use.

Parameter List

Name	Class Type	Default Value
URL	java.net.URL	NO_PARAMETER_DEFAULT
param	com.sun.media.jai.codec.ImageDecodeParam	null

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] `**`resources`**

The resource strings that provide the general documentation and specify the parameter list for the "URL" operation.

paramNames

`private static final java.lang.String[] **paramNames**`

The parameter names for the "URL" operation.

---

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "URL" operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "URL" operation.

Constructor Detail

URLDescriptor

`public **URLDescriptor**()`

Constructor.

javax.media.jai.operator
Class WarpDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.WarpDescriptor
```

public class **WarpDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Warp" operation.

The "Warp" operation performs (possibly filtered) general warping on an image.

"Warp" defines a PropertyGenerator that performs an identical transformation on the "ROI" property of the source image, which can be retrieved by calling the `getProperty` method with "ROI" as the property name.

Resource List

Name	Value
GlobalName	Warp
LocalName	Warp
Vendor	com.sun.media.jai
Description	Warps an image according to a specified Warp object.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/WarpDescriptor.html
Version	1.0
arg0Desc	The Warp object.
arg1Desc	The interpolation method.

Parameter List

Name	Class Type	Default Value
warp	javax.media.jai.Warp	NO_PARAMETER_DEFAULT
interpolation	javax.media.jai.Interpolation	InterpolationNearest

See Also:
`Interpolation`, `Warp`, `OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for the "Warp" operation.

paramNames

`private static final java.lang.String[] paramNames`

The parameter names for the "Warp" operation.

paramClasses

`private static final java.lang.Class[] **paramClasses**`

The parameter class types for the "Warp" operation.

---

paramDefaults

`private static final java.lang.Object[] **paramDefaults**`

The parameter default values for the "Warp" operation.

Constructor Detail

WarpDescriptor

`public **WarpDescriptor**()`

Constructor.

Method Detail

getPropertyGenerators

`public PropertyGenerator[] **getPropertyGenerators**()`

Returns an array of `PropertyGenerators` implementing property inheritance for the "Warp" operation.

Returns:

An array of property generators.

Overrides:

getPropertyGenerators in class OperationDescriptorImpl

javax.media.jai.operator

Class WarpPropertyGenerator

```
java.lang.Object
  |
  +--javax.media.jai.operator.WarpPropertyGenerator
```

class **WarpPropertyGenerator**
extends java.lang.Object
implements PropertyGenerator

This property generator computes the properties for the operation "Warp" dynamically.

Constructor Detail

WarpPropertyGenerator

`public WarpPropertyGenerator()`

Constructor.

Method Detail

getPropertyNames

`public java.lang.String[] getPropertyNames()`

Returns the valid property names for the operation "Warp".

Specified by:
getPropertyNames in interface PropertyGenerator

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderedOp op)
```

Returns the specified property.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

getProperty

```
public java.lang.Object getProperty(java.lang.String name,
                                    RenderableOp op)
```

Returns null.

Specified by:
getProperty in interface PropertyGenerator

Parameters:
`name` - Property name.

javax.media.jai.operator
Class XorConstDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.XorConstDescriptor
```

public class **XorConstDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "XorConst" operation.

The XorConst operation takes one rendered or renderable image and an array of integer constants, and performs a bit-wise logical "xor" between every pixel in the same band of the source and the constant from the corresponding array entry. If the number of constants supplied is less than the number of bands of the destination, then the constant from entry 0 is applied to all the bands. Otherwise, a constant from a different entry is applied to each band.

The source image must have an integral data type. By default, the destination image bound, data type, and number of bands are the same as the source image.

The following matrix defines the "xor" operation.

Logical "xor"

src	const	Result
0	0	0
0	1	1
1	0	1
1	1	0

The destination pixel values are defined by the pseudocode:

```
if (constants.length < dstNumBands) {
    dst[x][y][b] = src[x][y][b] ^ constants[0];
} else {
    dst[x][y][b] = src[x][y][b] ^ constants[b];
}
```

Resource List

Name	Value
GlobalName	XorConst
LocalName	XorConst
Vendor	com.sun.media.jai
Description	Logically "xors" a rendered image with constants.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/XorConstDescriptor.html
Version	1.0
arg0Desc	The constants to logically "xor" with.

Parameter List

Name	Class Type	Default Value
constants	int[]	NO_PARAMETER_DEFAULT

See Also:
`OperationDescriptor`

Field Detail

resources

`private static final java.lang.String[][] resources`

The resource strings that provide the general documentation and specify the parameter list for this operation.

paramClasses

`private static final java.lang.Class[] paramClasses`

The parameter class list for this operation. The number of constants provided should be either 1, in which case this same constant is applied to all the source bands; or the same number as the source bands, in which case one contant is applied to each band.

paramNames

`private static final java.lang.String[] paramNames`

The parameter name list for this operation.

paramDefaults

`private static final java.lang.Object[] paramDefaults`

The parameter default value list for this operation.

Constructor Detail

XorConstDescriptor

`public XorConstDescriptor()`

Constructor.

Method Detail

isRenderableSupported

`public boolean isRenderableSupported()`

Returns `true` since renderable operation is supported.

Overrides:
isRenderableSupported in class OperationDescriptorImpl

validateArguments

```
public boolean validateArguments(java.awt.image.renderable.ParameterBlock args,
                                 java.lang.StringBuffer message)
```

Validates the input source and parameter.

In addition to the standard checks performed by the superclass method, this method checks that the source image has an integral data type and that "constants" has length at least 1.

Overrides:
validateArguments in class OperationDescriptorImpl

javax.media.jai.operator
Class XorDescriptor

```
java.lang.Object
  |
  +--javax.media.jai.OperationDescriptorImpl
        |
        +--javax.media.jai.operator.XorDescriptor
```

public class **XorDescriptor**
extends OperationDescriptorImpl

An `OperationDescriptor` describing the "Xor" operation.

The Xor operation takes two rendered or renderable images, and performs bit-wise logical "xor" on every pair of pixels, one from each source image of the corresponding position and band. No additional parameters are required.

Both source images must have integral data types. The two data types may be different.

Unless altered by an `ImageLayout` hint, the destination image bound is the intersection of the two source image bounds. If the two sources don't intersect, the destination will have a width and height of 0. The number of bands of the destination image is equal to the lesser number of bands of the sources, and the data type is the smallest data type with sufficient range to cover the range of both source data types.

The following matrix defines the "xor" operation.

Logical "xor"

src1	src2	Result
0	0	0
0	1	1
1	0	1
1	1	0

The destination pixel values are defined by the pseudocode:

```
dst[x][y][b] = srcs[0][x][y][b] ^ srcs[0][x][y][b];
```

Resource List

Name	Value
GlobalName	Xor
LocalName	Xor
Vendor	com.sun.media.jai
Description	Logically "xors" two rendered images.
DocURL	http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/operator/XorDescriptor.html
Version	1.0

See Also:
`OperationDescriptor`

Field Detail

resources

```
private static final java.lang.String[][] resources
```

The resource strings that provide the general documentation and specify the parameter list for this operation.

Constructor Detail

XorDescriptor

```
public XorDescriptor()
```

Constructor.

Method Detail

isRenderableSupported

```
public boolean isRenderableSupported()
```

Returns `true` since renderable operation is supported.

Overrides:

isRenderableSupported in class OperationDescriptorImpl

---

validateSources

```
protected boolean validateSources(java.awt.image.renderable.ParameterBlock args,
                                  java.lang.StringBuffer msg)
```

Validates the input sources.

In addition to the standard checks performed by the superclass method, this method checks that the source images are of integral data type.

Overrides:

validateSources in class OperationDescriptorImpl

Package javax.media.jai.widget

Interface Summary	
ViewportListener	An interface used by the `ScrollingImagePanel` class to inform listeners of the current viewable area of the image.

Class Summary	
ImageCanvas	A simple output widget for a RenderedImage.
JaiI18N	
ScrollingImagePanel	An extension of java.awt.Panel that contains an ImageCanvas and vertical and horizontal scrollbars.

javax.media.jai.widget

Class ImageCanvas

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Canvas
              |
              +--javax.media.jai.widget.ImageCanvas
```

public class **ImageCanvas**
extends java.awt.Canvas

A simple output widget for a RenderedImage. ImageCanvas subclasses java.awt.Canvas, and can be used in any context that calls for a Canvas. It monitors resize and update events and automatically requests tiles from its source on demand. Any displayed area outside the image is displayed in grey.

There is currently no policy regarding what sorts of widgets, if any, will be part of JAI.

Due to the limitations of BufferedImage, only TYPE_BYTE of band 1, 2, 3, 4, and TYPE_USHORT of band 1, 2, 3 images can be displayed using this widget.

Field Detail

im

`protected java.awt.image.RenderedImage **im**`

The source RenderedImage.

sampleModel

`protected java.awt.image.SampleModel **sampleModel**`

The image's SampleModel.

colorModel

`protected java.awt.image.ColorModel **colorModel**`

The image's ColorModel or one we supply.

minTileX

`protected int **minTileX**`

The image's min X tile.

maxTileX

`protected int **maxTileX**`

The image's max X tile.

minTileY

`protected int **minTileY**`

The image's min Y tile.

maxTileY

`protected int **maxTileY**`

The image's max Y tile.

tileWidth

`protected int` **`tileWidth`**

The image's tile width.

---

tileHeight

`protected int` **`tileHeight`**

The image's tile height.

---

tileGridXOffset

`protected int` **`tileGridXOffset`**

The image's tile grid X offset.

---

tileGridYOffset

`protected int` **`tileGridYOffset`**

The image's tile grid Y offset.

---

imWidth

`protected int` **`imWidth`**

---

imHeight

`protected int` **`imHeight`**

---

padX

`protected int` **`padX`**

used to center image in it's container

---

padY

`protected int` **`padY`**

---

drawBorder

`protected boolean` **`drawBorder`**

---

originX

`protected int` **`originX`**

The pixel to display in the upper left corner or the canvas.

---

originY

`protected int` **`originY`**

The pixel to display in the upper left corner or the canvas.

---

canvasWidth

`protected int` **`canvasWidth`**

The width of the canvas.

---

canvasHeight

```
protected int canvasHeight
```

The height of the canvas.

---

grayColor

```
private java.awt.Color grayColor
```

---

backgroundColor

```
private java.awt.Color backgroundColor
```

Constructor Detail

ImageCanvas

```
public ImageCanvas(java.awt.image.RenderedImage im,
                   boolean drawBorder)
```

Constructs an ImageCanvas to display a RenderedImage.

Parameters:

`im` - a RenderedImage to be displayed.
`drawBorder` - true if a raised border is desired.

ImageCanvas

```
public ImageCanvas(java.awt.image.RenderedImage im)
```

Constructs an ImageCanvas to display a RenderedImage.

Parameters:

`im` - a RenderedImage to be displayed.

Method Detail

initialize

```
private void initialize()
```

Initializes the ImageCanvas.

---

addNotify

```
public void addNotify()
```

Overrides:

addNotify in class java.awt.Canvas

---

set

```
public void set(java.awt.image.RenderedImage im)
```

Changes the source image to a new RenderedImage.

---

setOrigin

```
public void setOrigin(int x,
                      int y)
```

Changes the pixel to set Origin at x,y

---

getXOrigin

```
public int getXOrigin()
```

---

getYOrigin

```
public int getYOrigin()
```

---

getXPad

```
public int getXPad()
```

---

getYPad

```
public int getYPad()
```

---

getMinimumSize

```
public java.awt.Dimension getMinimumSize()
```

Overrides:
getMinimumSize in class java.awt.Component

---

getPreferredSize

```
public java.awt.Dimension getPreferredSize()
```

Overrides:
getPreferredSize in class java.awt.Component

---

getMaximumSize

```
public java.awt.Dimension getMaximumSize()
```

Overrides:
getMaximumSize in class java.awt.Component

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Records a new size. Called by the AWT.
Overrides:
setBounds in class java.awt.Component

XtoTileX

```
private int XtoTileX(int x)
```

YtoTileY

```
private int YtoTileY(int y)
```

TileXtoX

```
private int TileXtoX(int tx)
```

TileYtoY

```
private int TileYtoY(int ty)
```

---

update

`public void **update**(java.awt.Graphics g)`

There is no need to erase prior to drawing, so we override the default update method to simply call paint().

Overrides:

update in class java.awt.Component

---

paint

`public void **paint**(java.awt.Graphics g)`

Paint the image onto a Graphics object. The painting is performed tile-by-tile, and includes a grey region covering the unused portion of image tiles as well as the general background.

Overrides:

paint in class java.awt.Canvas

javax.media.jai.widget
Class JaiI18N

```
java.lang.Object
  |
  +--javax.media.jai.widget.JaiI18N
```

class **JaiI18N**
extends java.lang.Object

Field Detail

packageName

```
static java.lang.String packageName
```

Constructor Detail

JaiI18N

```
JaiI18N()
```

Method Detail

getString

```
public static java.lang.String getString(java.lang.String key)
```

javax.media.jai.widget
Class ScrollingImagePanel

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.ScrollPane
                    |
                    +--javax.media.jai.widget.ScrollingImagePanel
```

public class **ScrollingImagePanel**
extends java.awt.ScrollPane
implements java.awt.event.AdjustmentListener, java.awt.event.ComponentListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener

An extension of java.awt.Panel that contains an ImageCanvas and vertical and horizontal scrollbars. The origin of the ImageCanvas is set according to the value of the scrollbars. Additionally, the origin may be changed by dragging the mouse. The window cursor will be changed to Cursor.MOVE_CURSOR for the duration of the drag.

Due to the limitations of BufferedImage, only TYPE_BYTE of band 1, 2, 3, 4, and TYPE_USHORT of band 1, 2, 3 images can be displayed using this widget.

Field Detail

ic

`protected ImageCanvas` **`ic`**

The ImageCanvas we are controlling.

im

`protected java.awt.image.RenderedImage` **`im`**

The RenderedImage displayed by the ImageCanvas.

panelWidth

`protected int` **`panelWidth`**

The width of the panel.

panelHeight

`protected int` **`panelHeight`**

The height of the panel.

viewportListeners

`protected java.util.Vector` **`viewportListeners`**

Vector of ViewportListeners.

moveSource

`protected java.awt.Point` **`moveSource`**

The initial Point of a mouse drag.

beingDragged

```
protected boolean beingDragged
```

True if we are in the middle of a mouse drag.

---

defaultCursor

```
protected java.awt.Cursor defaultCursor
```

A place to save the cursor.

Constructor Detail

ScrollingImagePanel

```
public ScrollingImagePanel(java.awt.image.RenderedImage im,
                           int width,
                           int height)
```

Constructs a ScrollingImagePanel of a given size for a given RenderedImage.

Method Detail

addViewportListener

```
public void addViewportListener(ViewportListener l)
```

Adds the specified ViewportListener to the panel

---

removeViewportListener

```
public void removeViewportListener(ViewportListener l)
```

Removes the specified ViewportListener

---

notifyViewportListeners

```
private void notifyViewportListeners(int x,
                                     int y,
                                     int w,
                                     int h)
```

---

getXOrigin

```
public int getXOrigin()
```

Returns the XOrigin of the image

---

getYOrigin

```
public int getYOrigin()
```

Returns the YOrigin of the image

---

setOrigin

```
public void setOrigin(int x,
                      int y)
```

Sets the image origin to a given (x, y) position. The scrollbars are updated appropriately.

---

setCenter

```
public void setCenter(int x,
                      int y)
```

Set the center of the image to the given coordinates of the scroll window.

---

set

```
public void set(java.awt.image.RenderedImage im)
```

Sets the panel to display the specified image

---

getXCenter

```
public int getXCenter()
```

Returns the X co-ordinate of the image center.

---

getYCenter

```
public int getYCenter()
```

Returns the Y co-ordinate of the image center.

---

getPreferredSize

```
public java.awt.Dimension getPreferredSize()
```

Called by the AWT when instantiating the component.
Overrides:
getPreferredSize in class java.awt.Container

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Called by the AWT during instantiation and when events such as resize occur.
Overrides:
setBounds in class java.awt.Component

---

adjustmentValueChanged

```
public void adjustmentValueChanged(java.awt.event.AdjustmentEvent e)
```

Called by the AWT when either scrollbar changes.
Specified by:
adjustmentValueChanged in interface java.awt.event.AdjustmentListener

---

componentResized

```
public void componentResized(java.awt.event.ComponentEvent e)
```

Called when the ImagePanel is resized
Specified by:
componentResized in interface java.awt.event.ComponentListener

---

componentHidden

```
public void componentHidden(java.awt.event.ComponentEvent e)
```

Ignored
Specified by:
componentHidden in interface java.awt.event.ComponentListener

---

componentMoved

```
public void componentMoved(java.awt.event.ComponentEvent e)
```

Ignored
Specified by:
componentMoved in interface java.awt.event.ComponentListener

componentShown

`public void **componentShown**(java.awt.event.ComponentEvent e)`

Ignored
Specified by:
componentShown in interface java.awt.event.ComponentListener

startDrag

`private void **startDrag**(java.awt.Point p)`

Called at the beginning of a mouse drag.

updateDrag

`protected void **updateDrag**(java.awt.Point moveTarget)`

Called for each point of a mouse drag.

endDrag

`private void **endDrag**()`

Called at the end of a mouse drag.

mousePressed

`public void **mousePressed**(java.awt.event.MouseEvent me)`

Called by the AWT when the mouse button is pressed.
Specified by:
mousePressed in interface java.awt.event.MouseListener

mouseDragged

`public void **mouseDragged**(java.awt.event.MouseEvent me)`

Called by the AWT as the mouse is dragged.
Specified by:
mouseDragged in interface java.awt.event.MouseMotionListener

mouseReleased

`public void **mouseReleased**(java.awt.event.MouseEvent me)`

Called by the AWT when the mouse button is released.
Specified by:
mouseReleased in interface java.awt.event.MouseListener

mouseExited

`public void **mouseExited**(java.awt.event.MouseEvent me)`

Called by the AWT when the mouse leaves the component.
Specified by:
mouseExited in interface java.awt.event.MouseListener

mouseClicked

`public void **mouseClicked**(java.awt.event.MouseEvent me)`

Ignored.
Specified by:
mouseClicked in interface java.awt.event.MouseListener

mouseMoved

```
public void mouseMoved(java.awt.event.MouseEvent me)
```

Ignored.

Specified by:

mouseMoved in interface java.awt.event.MouseMotionListener

---

mouseEntered

```
public void mouseEntered(java.awt.event.MouseEvent me)
```

Ignored.

Specified by:

mouseEntered in interface java.awt.event.MouseListener

javax.media.jai.widget
Interface ViewportListener

public abstract interface **ViewportListener**

An interface used by the `ScrollingImagePanel` class to inform listeners of the current viewable area of the image.

See Also:
`ScrollingImagePanel`

Method Detail

setViewport

```
public void setViewport(int x,
                        int y,
                        int width,
                        int height)
```

Called to inform the listener of the currently viewable area od the source image.

Parameters:
- `x` - The X coordinate of the upper-left corner of the current viewable area.
- `y` - The Y coordinate of the upper-left corner of the current viewable area.
- `width` - The width of the current viewable area in pixels.
- `height` - The height of the current viewable area in pixels.

Package com.sun.media.jai.codec

Interface Summary	
ImageDecodeParam	An empty (marker) interface to be implemented by all image decoder parameter classes.
ImageDecoder	An interface describing objects that transform an InputStream into a BufferedImage or Raster.
ImageEncodeParam	An empty (marker) interface to be implemented by all image encoder parameter classes.
ImageEncoder	An interface describing objects that transform a BufferedImage or Raster into an OutputStream.
StreamSegmentMapper	An interface for use with the `SegmentedSeekableStream` class.

Class Summary	
BMPEncodeParam	An instance of `ImageEncodeParam` for encoding images in the BMP format.
ByteArraySeekableStream	A subclass of `SeekableStream` that takes input from an array of bytes.
FileCacheSeekableStream	A subclass of `SeekableStream` that may be used to wrap a regular `InputStream`.
FileSeekableStream	A subclass of `SeekableStream` that takes its input from a `File` or `RandomAccessFile`.
ForwardSeekableStream	A subclass of `SeekableStream` that may be used to wrap a regular `InputStream` efficiently.
FPXDecodeParam	An instance of `ImageDecodeParam` for decoding images in the FlashPIX format.
ImageCodec	An abstract class allowing the creation of image decoders and encoders.
ImageDecoderImpl	A partial implementation of the `ImageDecoder` interface useful for subclassing.
ImageEncoderImpl	A partial implementation of the ImageEncoder interface useful for subclassing.
JaiI18N	
JPEGEncodeParam	A class which encapsulates the most common functionality required for the parameters to a Jpeg encode operation.
MemoryCacheSeekableStream	A subclass of `SeekableStream` that may be used to wrap a regular `InputStream`.
PNGDecodeParam	An instance of `ImageDecodeParam` for decoding images in the PNG format.
PNGEncodeParam	An instance of `ImageEncodeParam` for encoding images in the PNG format.
PNGEncodeParam.Gray	
PNGEncodeParam.Palette	
PNGEncodeParam.RGB	
PNGSuggestedPaletteEntry	A class representing the fields of a PNG suggested palette entry.
PNMEncodeParam	An instance of `ImageEncodeParam` for encoding images in the PNM format.
SectorStreamSegmentMapper	An implementation of the `StreamSegmentMapper` interface for segments of equal length.
SeekableStream	An abstract subclass of `java.io.InputStream` that allows seeking within the input, similar to the `RandomAccessFile` class.
SegmentedSeekableStream	A `SegmentedSeekableStream` provides a view of a subset of another `SeekableStream` consiting of a series of segments with given starting positions in the source stream and lengths.
StreamSegment	A utility class representing a segment within a stream as a `long` starting position and an `int` length.
StreamSegmentMapperImpl	An implementation of the `StreamSegmentMapper` interface that requires an explicit list of the starting locations and lengths of the source segments.
TIFFDecodeParam	An instance of `ImageDecodeParam` for decoding images in the TIFF format.
TIFFDirectory	A class representing an Image File Directory (IFD) from a TIFF 6.0 stream.
TIFFEncodeParam	An instance of `ImageEncodeParam` for encoding images in the TIFF format.
TIFFField	A class representing a field in a TIFF 6.0 Image File Directory.

com.sun.media.jai.codec
Class BMPEncodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.BMPEncodeParam
```

public class **BMPEncodeParam**
extends java.lang.Object
implements ImageEncodeParam

An instance of `ImageEncodeParam` for encoding images in the BMP format.

This class allows for the specification of various parameters while encoding (writing) a BMP format image file. By default, the version used is VERSION_3, no compression is used, and the data layout is bottom_up, such that the pixels are stored in bottom-up order, the first scanline being stored last.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

VERSION_2

`public static final int VERSION_2`

Constant for BMP version 2.

VERSION_3

`public static final int VERSION_3`

Constant for BMP version 3.

VERSION_4

`public static final int VERSION_4`

Constant for BMP version 4.

version

`private int version`

compressed

`private boolean compressed`

topDown

`private boolean topDown`

Constructor Detail

BMPEncodeParam

`public BMPEncodeParam()`

Constructs an BMPEncodeParam object with default values for parameters.

Method Detail

setVersion

`public void **setVersion**(int versionNumber)`

Sets the BMP version to be used.

---

getVersion

`public int **getVersion**()`

Returns the BMP version to be used.

---

setCompressed

`public void **setCompressed**(boolean compressed)`

If set, the data will be written out compressed, if possible.

---

isCompressed

`public boolean **isCompressed**()`

Returns the value of the parameter `compressed`.

---

setTopDown

`public void **setTopDown**(boolean topDown)`

If set, the data will be written out in a top-down manner, the first scanline being written first.

---

isTopDown

`public boolean **isTopDown**()`

Returns the value of the `topDown` parameter.

---

checkVersion

`private void **checkVersion**(int versionNumber)`

com.sun.media.jai.codec

Class ByteArraySeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.ByteArraySeekableStream
```

---

public class **ByteArraySeekableStream**
extends SeekableStream

A subclass of `SeekableStream` that takes input from an array of bytes. Seeking backwards is supported. The `mark()` and `resest()` methods are supported.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

src

```
private byte[] src
```

Array holding the source data.

---

offset

```
private int offset
```

The starting offset within the array.

---

length

```
private int length
```

The length of the valid segment of the array.

---

pointer

```
private int pointer
```

The current output position.

Constructor Detail

ByteArraySeekableStream

```
public ByteArraySeekableStream(byte[] src,
                               int offset,
                               int length)
                        throws java.io.IOException
```

Constructs a `ByteArraySeekableStream` taking input from a given segment of an input `byte` array.

ByteArraySeekableStream

```
public ByteArraySeekableStream(byte[] src)
                        throws java.io.IOException
```

Constructs a `ByteArraySeekableStream` taking input from an entire input `byte` array.

Method Detail

canSeekBackwards

```
public boolean canSeekBackwards()
```

Returns `true` since this object supports seeking backwards.

Overrides:
canSeekBackwards in class SeekableStream

---

getFilePointer

```
public long getFilePointer()
```

Returns the current offset in this stream.

Returns:
the offset from the beginning of the stream, in bytes, at which the next read occurs.

Overrides:
getFilePointer in class SeekableStream

---

seek

```
public void seek(long pos)
```

Sets the offset, measured from the beginning of this stream, at which the next read occurs. Seeking backwards is allowed.

Parameters:
`pos` - the offset position, measured in bytes from the beginning of the stream, at which to set the stream pointer.

Overrides:
seek in class SeekableStream

---

read

```
public int read()
```

Reads the next byte of data from the input array. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned.

Overrides:
read in class SeekableStream

---

read

```
public int read(byte[] b,
                int off,
                int len)
```

Copies up to `len` bytes of data from the input array into an array of bytes. An attempt is made to copy as many as `len` bytes, but a smaller number may be copied, possibly zero. The number of bytes actually copied is returned as an integer.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are copied and `0` is returned; otherwise, there is an attempt to copy at least one byte. If no byte is available because the stream is at end of stream, the value `-1` is returned; otherwise, at least one byte is copied into `b`.

The first byte copied is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes copied is, at most, equal to `len`. Let *k* be the number of bytes actually copied; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

Parameters:
`b` - the buffer into which the data is copied.
`off` - the start offset in array `b` at which the data is written.
`len` - the maximum number of bytes to copy.

Returns:
the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Overrides:
read in class SeekableStream

---

skipBytes

`public int skipBytes(int n)`

Attempts to skip over `n` bytes of input discarding the skipped bytes.

This method may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of stream before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

Parameters:
`n` - the number of bytes to be skipped.

Returns:
the actual number of bytes skipped.

Overrides:
skipBytes in class SeekableStream

---

close

`public void close()`

Does nothing.

Overrides:
close in class java.io.InputStream

---

length

`public long length()`

Returns the number of valid bytes in the input array.

com.sun.media.jai.codec
Class FPXDecodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.FPXDecodeParam
```

public class **FPXDecodeParam**
extends java.lang.Object
implements ImageDecodeParam

An instance of `ImageDecodeParam` for decoding images in the FlashPIX format.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

resolution

`private int resolution`

Constructor Detail

FPXDecodeParam

`public FPXDecodeParam()`

Constructs a default instance of `FPXDecodeParam`.

FPXDecodeParam

`public FPXDecodeParam(int resolution)`

Constructs an instance of `FPXDecodeParam` to decode a given resolution.

Parameters:
`resolution` - The resolution number to be decoded.

Method Detail

setResolution

`public void setResolution(int resolution)`

Sets the resolution to be decoded.

Parameters:
`resolution` - The resolution number to be decoded.

getResolution

`public int getResolution()`

Returns the resolution to be decoded.

com.sun.media.jai.codec
Class FileCacheSeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.FileCacheSeekableStream
```

public final class **FileCacheSeekableStream**
extends SeekableStream

A subclass of `SeekableStream` that may be used to wrap a regular `InputStream`. Seeking backwards is supported by means of a file cache. In circumstances that do not allow the creation of a temporary file (for example, due to security consideration or the absence of local disk), the `MemoryCacheSeekableStream` class may be used instead.

The `mark()` and `reset()` methods are supported.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

stream

`private java.io.InputStream` **`stream`**

The source stream.

cacheFile

`private java.io.File` **`cacheFile`**

The cache File.

cache

`private java.io.RandomAccessFile` **`cache`**

The cache as a RandomAcessFile.

bufLen

`private int` **`bufLen`**

The length of the read buffer.

buf

`private byte[]` **`buf`**

The read buffer.

length

`private long` **`length`**

Number of bytes in the cache.

pointer

`private long` **`pointer`**

Next byte to be read.

foundEOF

```
private boolean foundEOF
```

True if we’ve encountered the end of the source stream.

Constructor Detail

FileCacheSeekableStream

```
public FileCacheSeekableStream(java.io.InputStream stream)
                        throws java.io.IOException
```

Constructs a `MemoryCacheSeekableStream` that takes its source data from a regular `InputStream`. Seeking backwards is supported by means of an file cache.

An `IOException` will be thrown if the attempt to create the cache file fails for any reason.

Method Detail

readUntil

```
private long readUntil(long pos)
                throws java.io.IOException
```

Ensures that at least `pos` bytes are cached, or the end of the source is reached. The return value is equal to the smaller of `pos` and the length of the source file.

---

canSeekBackwards

```
public boolean canSeekBackwards()
```

Returns `true` since all `FileCacheSeekableStream` instances support seeking backwards.

Overrides:

canSeekBackwards in class SeekableStream

---

getFilePointer

```
public long getFilePointer()
```

Returns the current offset in this file.

Returns:

the offset from the beginning of the file, in bytes, at which the next read occurs.

Overrides:

getFilePointer in class SeekableStream

---

seek

```
public void seek(long pos)
          throws java.io.IOException
```

Sets the file-pointer offset, measured from the beginning of this file, at which the next read occurs.

Parameters:

`pos` - the offset position, measured in bytes from the beginning of the file, at which to set the file pointer.

Throws:

java.io.IOException - if `pos` is less than `0` or if an I/O error occurs.

Overrides:

seek in class SeekableStream

---

read

```
public int read()
         throws java.io.IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of data, or `-1` if the end of the stream is reached.

Throws:

java.io.IOException - if an I/O error occurs.

Overrides:
read in class SeekableStream

---

read

```
public int read(byte[] b,
                int off,
                int len)
         throws java.io.IOException
```

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

Parameters:
`b` - the buffer into which the data is read.
`off` - the start offset in array `b` at which the data is written.
`len` - the maximum number of bytes to read.

Returns:
the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:
java.io.IOException - if an I/O error occurs.

Overrides:
read in class SeekableStream

---

close

```
public void close()
           throws java.io.IOException
```

Closes this stream and releases any system resources associated with the stream.

Throws:
java.io.IOException - if an I/O error occurs.

Overrides:
close in class java.io.InputStream

com.sun.media.jai.codec
Class FileSeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.FileSeekableStream
```

public class **FileSeekableStream**
extends SeekableStream

A subclass of `SeekableStream` that takes its input from a `File` or `RandomAccessFile`. Backwards seeking is supported. The `mark()` and `resest()` methods are supported.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

file
`private java.io.RandomAccessFile file`

markPos
`private long markPos`

PAGE_SHIFT
`private static final int PAGE_SHIFT`

PAGE_SIZE
`private static final int PAGE_SIZE`

PAGE_MASK
`private static final int PAGE_MASK`

NUM_PAGES
`private static final int NUM_PAGES`

READ_CACHE_LIMIT
`private static final int READ_CACHE_LIMIT`

pageBuf
`private byte[][] pageBuf`

currentPage
`private int[] currentPage`

length

```
private long length
```

---

pointer

```
private long pointer
```

Constructor Detail

FileSeekableStream

```
public FileSeekableStream(java.io.RandomAccessFile file)
                   throws java.io.IOException
```

Constructs a `FileSeekableStream` from a `RandomAccessFile`.

FileSeekableStream

```
public FileSeekableStream(java.io.File file)
                   throws java.io.IOException
```

Constructs a `FileSeekableStream` from a `File`.

FileSeekableStream

```
public FileSeekableStream(java.lang.String name)
                   throws java.io.IOException
```

Constructs a `FileSeekableStream` from a `String` path name.

Method Detail

canSeekBackwards

```
public final boolean canSeekBackwards()
```

Returns true since seeking backwards is supported.

Overrides:
canSeekBackwards in class SeekableStream

---

getFilePointer

```
public final long getFilePointer()
                          throws java.io.IOException
```

Returns the current offset in this stream.

Returns:
the offset from the beginning of the stream, in bytes, at which the next read occurs.

Throws:
java.io.IOException - if an I/O error occurs.

Overrides:
getFilePointer in class SeekableStream

---

seek

```
public final void seek(long pos)
                throws java.io.IOException
```

Overrides:
seek in class SeekableStream

---

skip

```
public final int skip(int n)
               throws java.io.IOException
```

---

readPage

```
private byte[] readPage(long pointer)
                 throws java.io.IOException
```

---

read

```
public final int read()
               throws java.io.IOException
```

Forwards the request to the real `File`.

Overrides:

read in class SeekableStream

---

read

```
public final int read(byte[] b,
                      int off,
                      int len)
               throws java.io.IOException
```

Forwards the request to the real `File`.

Overrides:

read in class SeekableStream

---

close

```
public final void close()
                 throws java.io.IOException
```

Forwards the request to the real `File`.

Overrides:

close in class java.io.InputStream

---

mark

```
public final void mark(int readLimit)
```

Marks the current file position for later return using the `reset()` method.

Overrides:

mark in class SeekableStream

---

reset

```
public final void reset()
                 throws java.io.IOException
```

Returns the file position to its position at the time of the immediately previous call to the `mark()` method.

Overrides:

reset in class SeekableStream

---

markSupported

```
public boolean markSupported()
```

Returns `true` since marking is supported.

Overrides:

markSupported in class SeekableStream

com.sun.media.jai.codec

Class ForwardSeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.ForwardSeekableStream
```

public class **ForwardSeekableStream**
extends SeekableStream

A subclass of `SeekableStream` that may be used to wrap a regular `InputStream` efficiently. Seeking backwards is not supported.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

src

`private java.io.InputStream src`

The source `InputStream`.

pointer

`long pointer`

The current position.

markPos

`long markPos`

The marked position.

Constructor Detail

ForwardSeekableStream

`public ForwardSeekableStream(java.io.InputStream src)`

Constructs a `InputStreamForwardSeekableStream` from a regular `InputStream`.

Method Detail

read

```
public final int read()
               throws java.io.IOException
```

Forwards the request to the real `InputStream`.

Overrides:

read in class SeekableStream

read

```
public final int read(byte[] b,
                      int off,
                      int len)
               throws java.io.IOException
```

Forwards the request to the real `InputStream`.

Overrides:

read in class SeekableStream

skip

```
public final long skip(long n)
                throws java.io.IOException
```

Forwards the request to the real `InputStream`.
Overrides:
skip in class java.io.InputStream

available

```
public final int available()
                    throws java.io.IOException
```

Forwards the request to the real `InputStream`.
Overrides:
available in class java.io.InputStream

close

```
public final void close()
                 throws java.io.IOException
```

Forwards the request to the real `InputStream`.
Overrides:
close in class java.io.InputStream

mark

```
public final void mark(int readLimit)
```

Forwards the request to the real `InputStream`.
Overrides:
mark in class SeekableStream

reset

```
public final void reset()
                 throws java.io.IOException
```

Forwards the request to the real `InputStream`.
Overrides:
reset in class SeekableStream

markSupported

```
public boolean markSupported()
```

Forwards the request to the real `InputStream`.
Overrides:
markSupported in class SeekableStream

canSeekBackwards

```
public final boolean canSeekBackwards()
```

Returns `false` since seking backwards is not supported.
Overrides:
canSeekBackwards in class SeekableStream

getFilePointer

```
public final long getFilePointer()
```

Returns the current position in the stream (bytes read).
Overrides:
getFilePointer in class SeekableStream

seek

```
public final void seek(long pos)
                throws java.io.IOException
```

Seeks forward to the given position in the stream. If `pos` is smaller than the current position as returned by `getFilePointer()`, nothing happens.

Overrides:

seek in class SeekableStream

com.sun.media.jai.codec

Class ImageCodec

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.ImageCodec
```

public abstract class **ImageCodec**
extends java.lang.Object

An abstract class allowing the creation of image decoders and encoders. Instances of `ImageCodec` may be registered. Once a codec has been registered, the format name associated with it may be used as the `name` parameter in the `createImageEncoder()` and `createImageDecoder()` methods.

Additionally, subclasses of `ImageCodec` are able to perform recognition of their particular format, wither by inspection of a fixed-length file header or by arbitrary access to the source data stream.

Format recognition is performed by two variants of the `isFormatRecognized()` method. Which variant should be called is determined by the output of the codec's getNumHeaderBytes() method, which returns 0 if arbitrary access to the stream is required, and otherwise returns the number of header bytes required to recognize the format. Each subclass of `ImageCodec` needs to implement only one of the two variants.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

codecs

`private static java.util.Hashtable` **`codecs`**

grayIndexCmaps

`private static final byte[][]` **`grayIndexCmaps`**

GrayBits8

`private static final int[]` **`GrayBits8`**

colorModelGray8

`private static final java.awt.image.ComponentColorModel` **`colorModelGray8`**

GrayAlphaBits8

`private static final int[]` **`GrayAlphaBits8`**

colorModelGrayAlpha8

`private static final java.awt.image.ComponentColorModel` **`colorModelGrayAlpha8`**

GrayBits16

`private static final int[]` **`GrayBits16`**

colorModelGray16

`private static final java.awt.image.ComponentColorModel` **`colorModelGray16`**

GrayAlphaBits16

`private static final int[] GrayAlphaBits16`

---

colorModelGrayAlpha16

`private static final java.awt.image.ComponentColorModel colorModelGrayAlpha16`

GrayBits32

`private static final int[] GrayBits32`

---

colorModelGray32

`private static final java.awt.image.ComponentColorModel colorModelGray32`

GrayAlphaBits32

`private static final int[] GrayAlphaBits32`

---

colorModelGrayAlpha32

`private static final java.awt.image.ComponentColorModel colorModelGrayAlpha32`

RGBBits8

`private static final int[] RGBBits8`

---

colorModelRGB8

`private static final java.awt.image.ComponentColorModel colorModelRGB8`

RGBABits8

`private static final int[] RGBABits8`

---

colorModelRGBA8

`private static final java.awt.image.ComponentColorModel colorModelRGBA8`

RGBBits16

`private static final int[] RGBBits16`

---

colorModelRGB16

`private static final java.awt.image.ComponentColorModel colorModelRGB16`

RGBABits16

`private static final int[] RGBABits16`

---

colorModelRGBA16

`private static final java.awt.image.ComponentColorModel colorModelRGBA16`

RGBBits32

`private static final int[] RGBBits32`

---

colorModelRGB32

`private static final java.awt.image.ComponentColorModel colorModelRGB32`

RGBABits32

`private static final int[] RGBABits32`

---

colorModelRGBA32

`private static final java.awt.image.ComponentColorModel colorModelRGBA32`

Constructor Detail

ImageCodec

`protected ImageCodec()`

Allow only subclasses to instantiate this class.

Method Detail

`static void ()`

Load the JPEG and PNM codecs.

---

getCodec

`public static ImageCodec getCodec(java.lang.String name)`

Returns the `ImageCodec` associated with the given name. `null` is returned if no codec is registered with the given name. Case is not significant.

Parameters:

`name` - The name associated with the codec.

Returns:

The associated `ImageCodec`, or `null`.

---

registerCodec

`public static void registerCodec(ImageCodec codec)`

Associates an `ImageCodec` with its format name, as determined by its `getFormatName()` method. Case is not significant. Any codec previously associated with the name is discarded.

Parameters:

`codec` - The `ImageCodec` object to be registered.

---

unregisterCodec

`public static void unregisterCodec(java.lang.String name)`

Unregisters the `ImageCodec` object currently responsible for handling the named format. Case is not significant.

Parameters:

`name` - The name associated with the codec to be removed.

---

getCodecs

`public static java.util.Enumeration getCodecs()`

Returns an `Enumeration` of all regstered `ImageCodec` objects.

---

createImageEncoder

```
public static ImageEncoder createImageEncoder(java.lang.String name,
                                              java.io.OutputStream dst,
                                              ImageEncodeParam param)
```

Returns an `ImageEncoder` object suitable for encoding to the supplied `OutputStream`, using the supplied `ImageEncoderParam` object.

Parameters:

`name` - The name associated with the codec.

`dst` - An `OutputStream` to write to.

`param` - An instance of `ImageEncoderParam` suitable for use with the named codec, or `null`.

Returns:

An instance of `ImageEncoder`, or `null`.

---

createImageDecoder

```
public static ImageDecoder createImageDecoder(java.lang.String name,
                                              java.io.InputStream src,
                                              ImageDecodeParam param)
```

Returns an `ImageDecoder` object suitable for decoding from the supplied `InputStream`, using the supplied `ImageDecodeParam` object.

Parameters:

`name` - The name associated with the codec.

`src` - An `InputStream` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the named codec, or `null`.

Returns:

An instance of `ImageDecoder`, or `null`.

---

createImageDecoder

```
public static ImageDecoder createImageDecoder(java.lang.String name,
                                              java.io.File src,
                                              ImageDecodeParam param)
                                       throws java.io.IOException
```

Returns an `ImageDecoder` object suitable for decoding from the supplied `File`, using the supplied `ImageDecodeParam` object.

Parameters:

`name` - The name associated with the codec.

`src` - A `File` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the named codec, or `null`.

Returns:

An instance of `ImageDecoder`, or `null`.

---

createImageDecoder

```
public static ImageDecoder createImageDecoder(java.lang.String name,
                                              SeekableStream src,
                                              ImageDecodeParam param)
```

Returns an `ImageDecoder` object suitable for decoding from the supplied `SeekableStream`, using the supplied `ImageDecodeParam` object.

Parameters:

`name` - The name associated with the codec.

`src` - A `SeekableStream` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the named codec, or `null`.

Returns:

An instance of `ImageDecoder`, or `null`.

---

vectorToStrings

```
private static java.lang.String[] vectorToStrings(java.util.Vector nameVec)
```

---

getDecoderNames

```
public static java.lang.String[] getDecoderNames(SeekableStream src)
```

Returns an array of `String`s indicating the names of registered `ImageCodec`s that may be appropriate for reading the given `SeekableStream`.

If the `src` `SeekableStream` does not support seeking backwards (that is, its `canSeekBackwards()` method returns `false`) then only `FormatRecognizer`s that require only a fixed-length header will be checked.

If the `src` stream does not support seeking backwards, it must support marking, as determined by its `markSupported()` method.

Parameters:
`src` - A `SeekableStream` which optionally supports seeking backwards.
Returns:
An array of `String`s.
Throws:
java.lang.IllegalArgumentException - if `src` supports neither seeking backwards nor marking.

---

getEncoderNames

```
public static java.lang.String[] getEncoderNames(java.awt.image.RenderedImage im,
                                                 ImageEncodeParam param)
```

Returns an array of `String`s indicating the names of registered `ImageCodec`s that may be appropriate for writing the given `RenderedImage`, using the optional `ImageEncodeParam`, which may be `null`.

Parameters:
`im` - A `RenderedImage` to be encodec.
`param` - An `ImageEncodeParam`, or null.
Returns:
An array of `String`s.

---

getFormatName

```
public abstract java.lang.String getFormatName()
```

Returns the name of this image format.

Returns:
A `String` containing the name of the image format supported by this codec.

---

getNumHeaderBytes

```
public int getNumHeaderBytes()
```

Returns the number of bytes of header needed to recognize the format, or 0 if an arbitrary number of bytes may be needed. The default implementation returns 0.

The return value must be a constant for all instances of each particular subclass of `ImageCodec`.

Although it is legal to always return 0, in some cases processing may be more efficient if the number of bytes needed is known in advance.

---

isFormatRecognized

```
public boolean isFormatRecognized(byte[] header)
```

Returns `true` if the format is recognized in the initial portion of a stream. The header will be passed in as a `byte` array of length `getNumHeaderBytes()`. This method should be called only if `getNumHeaderBytes()` returns a value greater than 0.

The default implementation throws an exception to indicate that it should never be called.

Parameters:
`header` - An array of `byte`s containing the input stream header.
Returns:
`true` if the format is recognized.

---

isFormatRecognized

```
public boolean isFormatRecognized(SeekableStream src)
                           throws java.io.IOException
```

Returns `true` if the format is recognized in the input data stream. This method should be called only if `getNumHeaderBytesNeeded()` returns 0.

The source `SeekableStream` is guaranteed to support seeking backwards, and should be seeked to 0 prior to calling this method.

The default implementation throws an exception to indicate that it should never be called.

Parameters:

`src` - A `SeekableStream` containing the input data.

Returns:

`true` if the format is recognized.

---

getEncodeParamClass

```
protected abstract java.lang.Class getEncodeParamClass()
```

Returns a `Class` object indicating the proper subclass of `ImageEncodeParam` to be used with this `ImageCodec`. If encoding is not supported by this codec, `null` is returned. If encoding is supported, but a parameter object is not used during encoding, Object.class is returned to signal this fact.

---

getDecodeParamClass

```
protected abstract java.lang.Class getDecodeParamClass()
```

Returns a `Class` object indicating the proper subclass of `ImageDecodeParam` to be used with this `ImageCodec`. If encoding is not supported by this codec, `null` is returned. If decoding is supported, but a parameter object is not used during decoding, Object.class is returned to signal this fact.

---

createImageEncoder

```
protected abstract ImageEncoder createImageEncoder(java.io.OutputStream dst,
                                                   ImageEncodeParam param)
```

In a concrete subclass of `ImageCodec`, returns an implementation of the `ImageEncoder` interface appropriate for that codec.

Parameters:

`dst` - An `OutputStream` to write to.

`param` - An instance of `ImageEncoderParam` suitable for use with the `ImageCodec` subclass, or `null`.

Returns:

An instance of `ImageEncoder`.

---

canEncodeImage

```
public abstract boolean canEncodeImage(java.awt.image.RenderedImage im,
                                       ImageEncodeParam param)
```

Returns `true` if the given image and encoder param object are suitable for encoding by this `ImageCodec`. For example, some codecs may only deal with images with a certain number of bands; an attempt to encode an image with an unsupported number of bands will fail.

Parameters:

`im` - a RenderedImage whose ability to be encoded is to be determined.

`param` - a suitable `ImageEncodeParam` object, or `null`.

---

createImageDecoder

```
protected ImageDecoder createImageDecoder(java.io.InputStream src,
                                          ImageDecodeParam param)
```

Returns an implementation of the `ImageDecoder` interface appropriate for that codec. Subclasses of `ImageCodec` may override this method if they wish to accept data directly from an `InputStream`; otherwise, this method will convert the source into a backwards-seekable `SeekableStream` and call the appropriate version of `createImageDecoder` for that data type.

Instances of `ImageCodec` that do not require the ability to seek backwards in their source `SeekableStream` should override this method in order to avoid the default call to `SeekableStream.wrapInputStream(src, true)`.

Parameters:

`dst` - An `InputStream` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the `ImageCodec` subclass, or `null`.

Returns:

An instance of `ImageDecoder`.

---

createImageDecoder

```
protected ImageDecoder createImageDecoder(java.io.File src,
                                          ImageDecodeParam param)
                                   throws java.io.IOException
```

Returns an implementation of the `ImageDecoder` interface appropriate for that codec. Subclasses of `ImageCodec` may override this method if they wish to accept data directly from a `File`; otherwise, this method will convert the source into a `SeekableStream` and call the appropriate version of `createImageDecoder` for that data type.

Parameters:

`dst` - A `File` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the `ImageCodec` subclass, or `null`.

Returns:

An instance of `ImageDecoder`.

---

createImageDecoder

```
protected abstract ImageDecoder createImageDecoder(SeekableStream src,
                                                   ImageDecodeParam param)
```

In a concrete subclass of `ImageCodec`, returns an implementation of the `ImageDecoder` interface appropriate for that codec.

Parameters:

`dst` - A `SeekableStream` to read from.

`param` - An instance of `ImageDecodeParam` suitable for use with the `ImageCodec` subclass, or `null`.

Returns:

An instance of `ImageDecoder`.

---

createGrayIndexColorModel

```
public static java.awt.image.ColorModel createGrayIndexColorModel(java.awt.image.SampleModel sm,
                                                                  boolean blackIsZero)
```

A convenience methods to create an instance of `IndexColorModel` suitable for the given 1-banded `SampleModel`.

Parameters:

`sm` - a 1-banded `SampleModel`.

`blackIsZero` - `true` if the gray ramp should go from black to white, `false`otherwise.

---

createComponentColorModel

```
public static java.awt.image.ColorModel createComponentColorModel(java.awt.image.SampleModel sm)
```

A convenience method to create an instance of `ComponentColorModel` suitable for use with the given `SampleModel`. The `SampleModel` should have a data type of `DataBuffer.TYPE_BYTE`, `TYPE_USHORT`, or `TYPE_INT` and between 1 and 4 bands. Depending on the number of bands of the `SampleModel`, either a gray, gray+alpha, rgb, or rgb+alpha `ColorModel` is returned.

com.sun.media.jai.codec
Interface ImageDecodeParam

All Known Subinterfaces:
ImageEncodeParam

All Known Implementing Classes:
TIFFDecodeParam, FPXDecodeParam, PNGDecodeParam

---

public abstract interface **ImageDecodeParam**
extends java.lang.Cloneable, java.io.Serializable

An empty (marker) interface to be implemented by all image decoder parameter classes.

This interface is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

---

com.sun.media.jai.codec

Interface ImageDecoder

All Known Implementing Classes:
ImageDecoderImpl

---

public abstract interface **ImageDecoder**

An interface describing objects that transform an InputStream into a BufferedImage or Raster.

This interface is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Method Detail

getParam

```
public ImageDecodeParam getParam()
```

Returns the current parameters as an instance of the ImageDecodeParam interface. Concrete implementations of this interface will return corresponding concrete implementations of the ImageDecodeParam interface. For example, a JPEGImageDecoder will return an instance of JPEGDecodeParam.

---

setParam

```
public void setParam(ImageDecodeParam param)
```

Sets the current parameters to an instance of the ImageDecodeParam interface. Concrete implementations of ImageDecoder may throw a RuntimeException if the param argument is not an instance of the appropriate subclass or subinterface. For example, a JPEGImageDecoder will expect param to be an instance of JPEGDecodeParam.

---

getInputStream

```
public SeekableStream getInputStream()
```

Returns the SeekableStream associated with this ImageDecoder.

---

getNumPages

```
public int getNumPages()
                throws java.io.IOException
```

Returns the number of pages present in the current stream.

---

decodeAsRaster

```
public java.awt.image.Raster decodeAsRaster()
                                     throws java.io.IOException
```

Returns a Raster that contains the decoded contents of the SeekableStream associated with this ImageDecoder. Only the first page of a multi-page image is decoded.

---

decodeAsRaster

```
public java.awt.image.Raster decodeAsRaster(int page)
                                     throws java.io.IOException
```

Returns a Raster that contains the decoded contents of the SeekableStream associated with this ImageDecoder. The given page of a multi-page image is decoded. If the page does not exist, an IOException will be thrown. Page numbering begins at zero.

Parameters:
`page` - The page to be decoded.

---

decodeAsRenderedImage

```
public java.awt.image.RenderedImage decodeAsRenderedImage()
                                                 throws java.io.IOException
```

Returns a RenderedImage that contains the decoded contents of the SeekableStream associated with this ImageDecoder. Only the first page of a multi-page image is decoded.

---

decodeAsRenderedImage

```
public java.awt.image.RenderedImage decodeAsRenderedImage(int page)
                                                 throws java.io.IOException
```

Returns a RenderedImage that contains the decoded contents of the SeekableStream associated with this ImageDecoder. The given page of a multi-page image is decoded. If the page does not exist, an IOException will be thrown. Page numbering begins at zero.

Parameters:

`page` - The page to be decoded.

com.sun.media.jai.codec
Class ImageDecoderImpl

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.ImageDecoderImpl
```

public abstract class **ImageDecoderImpl**
extends java.lang.Object
implements ImageDecoder

A partial implementation of the `ImageDecoder` interface useful for subclassing.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

input

`protected SeekableStream input`

The `SeekableStream` associcted with this `ImageEncoder`.

param

`protected ImageDecodeParam param`

The `ImageDecodeParam` object associated with this `ImageEncoder`.

Constructor Detail

ImageDecoderImpl

```
public ImageDecoderImpl(SeekableStream input,
                        ImageDecodeParam param)
```

Constructs an `ImageDecoderImpl` with a given `SeekableStream` and `ImageDecodeParam` instance.

ImageDecoderImpl

```
public ImageDecoderImpl(java.io.InputStream input,
                        ImageDecodeParam param)
```

Constructs an `ImageDecoderImpl` with a given `InputStream` and `ImageDecodeParam` instance. The `input` parameter will be used to construct a `ForwardSeekableStream`; if the ability to seek backwards is required, the caller should construct an instance of `SeekableStream` and make use of the other contructor.

Method Detail

getParam

`public ImageDecodeParam getParam()`

Returns the current parameters as an instance of the `ImageDecodeParam` interface. Concrete implementations of this interface will return corresponding concrete implementations of the `ImageDecodeParam` interface. For example, a `JPEGImageDecoder` will return an instance of `JPEGDecodeParam`.

Specified by:
getParam in interface ImageDecoder

setParam

`public void setParam(ImageDecodeParam param)`

Sets the current parameters to an instance of the `ImageDecodeParam` interface. Concrete implementations of `ImageDecoder` may throw a `RuntimeException` if the `param` argument is not an instance of the appropriate subclass or subinterface. For example, a `JPEGImageDecoder` will expect `param` to be an instance of `JPEGDecodeParam`.

Specified by:
setParam in interface ImageDecoder

getInputStream

```
public SeekableStream getInputStream()
```

Returns the `SeekableStream` associated with this `ImageDecoder`.

Specified by:
getInputStream in interface ImageDecoder

getNumPages

```
public int getNumPages()
                throws java.io.IOException
```

Returns the number of pages present in the current stream. By default, the return value is 1. Subclasses that deal with multi-page formats should override this method.

Specified by:
getNumPages in interface ImageDecoder

decodeAsRaster

```
public java.awt.image.Raster decodeAsRaster()
                                     throws java.io.IOException
```

Returns a `Raster` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`. Only the first page of a multi-page image is decoded.

Specified by:
decodeAsRaster in interface ImageDecoder

decodeAsRaster

```
public java.awt.image.Raster decodeAsRaster(int page)
                                     throws java.io.IOException
```

Returns a `Raster` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`. The given page of a multi-page image is decoded. If the page does not exist, an IOException will be thrown. Page numbering begins at zero.

Specified by:
decodeAsRaster in interface ImageDecoder

Parameters:
`page` - The page to be decoded.

decodeAsRenderedImage

```
public java.awt.image.RenderedImage decodeAsRenderedImage()
                                                   throws java.io.IOException
```

Returns a `RenderedImage` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`. Only the first page of a multi-page image is decoded.

Specified by:
decodeAsRenderedImage in interface ImageDecoder

decodeAsRenderedImage

```
public abstract java.awt.image.RenderedImage decodeAsRenderedImage(int page)
                                                            throws java.io.IOException
```

Returns a `RenderedImage` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`. The given page of a multi-page image is decoded. If the page does not exist, an IOException will be thrown. Page numbering begins at zero.

Specified by:
decodeAsRenderedImage in interface ImageDecoder

Parameters:
`page` - The page to be decoded.

com.sun.media.jai.codec

Interface ImageEncodeParam

All Known Implementing Classes:
BMPEncodeParam, PNGEncodeParam, TIFFEncodeParam, JPEGEncodeParam, PNMEncodeParam

public abstract interface **ImageEncodeParam**
extends ImageDecodeParam, java.lang.Cloneable, java.io.Serializable

An empty (marker) interface to be implemented by all image encoder parameter classes.

This interface is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

com.sun.media.jai.codec
Interface ImageEncoder

All Known Implementing Classes:
ImageEncoderImpl

---

public abstract interface **ImageEncoder**

An interface describing objects that transform a BufferedImage or Raster into an OutputStream.

This interface is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Method Detail

getParam

```
public ImageEncodeParam getParam()
```

Returns the current parameters as an instance of the ImageEncodeParam interface. Concrete implementations of this interface will return corresponding concrete implementations of the ImageEncodeParam interface. For example, a JPEGImageEncoder will return an instance of JPEGEncodeParam.

---

setParam

```
public void setParam(ImageEncodeParam param)
```

Sets the current parameters to an instance of the ImageEncodeParam interface. Concrete implementations of ImageEncoder may throw a RuntimeException if the params argument is not an instance of the appropriate subclass or subinterface. For example, a JPEGImageEncoder will expect param to be an instance of JPEGEncodeParam.

---

getOutputStream

```
public java.io.OutputStream getOutputStream()
```

Returns the OutputStream associated with this ImageEncoder.

---

encode

```
public void encode(java.awt.image.Raster ras,
                   java.awt.image.ColorModel cm)
            throws java.io.IOException
```

Encodes a Raster with a given ColorModel and writes the output to the OutputStream associated with this ImageEncoder.

---

encode

```
public void encode(java.awt.image.RenderedImage im)
            throws java.io.IOException
```

Encodes a RenderedImage and writes the output to the OutputStream associated with this ImageEncoder.

com.sun.media.jai.codec
Class ImageEncoderImpl

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.ImageEncoderImpl
```

---

public abstract class **ImageEncoderImpl**
extends java.lang.Object
implements ImageEncoder

A partial implementation of the ImageEncoder interface useful for subclassing.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

output

`protected java.io.OutputStream **output**`

The OutputStream associcted with this ImageEncoder.

---

param

`protected ImageEncodeParam **param**`

The ImageEncodeParam object associcted with this ImageEncoder.

Constructor Detail

ImageEncoderImpl

```
public ImageEncoderImpl(java.io.OutputStream output,
                        ImageEncodeParam param)
```

Constructs an ImageEncoderImpl with a given OutputStream and ImageEncoderParam instance.

Method Detail

getParam

`public ImageEncodeParam **getParam**()`

Returns the current parameters as an instance of the ImageEncodeParam interface. Concrete implementations of this interface will return corresponding concrete implementations of the ImageEncodeParam interface. For example, a JPEGImageEncoder will return an instance of JPEGEncodeParam.

Specified by:
getParam in interface ImageEncoder

---

setParam

`public void **setParam**(ImageEncodeParam param)`

Sets the current parameters to an instance of the ImageEncodeParam interface. Concrete implementations of ImageEncoder may throw a RuntimeException if the params argument is not an instance of the appropriate subclass or subinterface. For example, a JPEGImageEncoder will expect param to be an instance of JPEGEncodeParam.

Specified by:
setParam in interface ImageEncoder

---

getOutputStream

`public java.io.OutputStream **getOutputStream**()`

Returns the OutputStream associated with this ImageEncoder.

Specified by:
getOutputStream in interface ImageEncoder

encode

```
public void encode(java.awt.image.Raster ras,
                   java.awt.image.ColorModel cm)
            throws java.io.IOException
```

Encodes a Raster with a given ColorModel and writes the output to the OutputStream associated with this ImageEncoder.

Specified by:

encode in interface ImageEncoder

encode

```
public abstract void encode(java.awt.image.RenderedImage im)
                     throws java.io.IOException
```

Encodes a RenderedImage and writes the output to the OutputStream associated with this ImageEncoder.

Specified by:

encode in interface ImageEncoder

com.sun.media.jai.codec
Class JPEGEncodeParam

```
java.lang.Object
   |
   +--com.sun.media.jai.codec.JPEGEncodeParam
```

---

public class **JPEGEncodeParam**
extends java.lang.Object
implements ImageEncodeParam

A class which encapsulates the most common functionality required for the parameters to a Jpeg encode operation. It does not include all of the parameters of the `com.sun.image.codec.jpeg` classes. Users needing that additional functionality should use those classes directly, bearing in mind that they are part of an uncommitted non-core interface that may be modified or removed in the future. This class makes very simple assumptions about the image colorspaces. Images with a single band are assumed to be grayscale. Images with three bands are assumed to be RGB and are encoded to YCbCr.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

JPEG_MAX_BANDS
`private static int` **`JPEG_MAX_BANDS`**

---

hSamp
`private int[]` **`hSamp`**

---

vSamp
`private int[]` **`vSamp`**

---

qTab
`private int[][]` **`qTab`**

---

qTabSlot
`private int[]` **`qTabSlot`**

---

qual
`private float` **`qual`**

---

rstInterval
`private int` **`rstInterval`**

---

writeImageOnly
`private boolean` **`writeImageOnly`**

---

writeTablesOnly
`private boolean` **`writeTablesOnly`**

---

writeJFIFHeader

```
private boolean writeJFIFHeader
```

---

qualitySet

```
private boolean qualitySet
```

---

qTabSet

```
private boolean[] qTabSet
```

Constructor Detail

JPEGEncodeParam

```
public JPEGEncodeParam()
```

Constructs a JAI JPEGEncodeParam object with default settings.

Method Detail

setHorizontalSubsampling

```
public void setHorizontalSubsampling(int component,
                                     int subsample)
```

Sets the horizontal subsampling to be applied to an image band. Defaults to 1 for grayscale and (1,2,2) for RGB.

Parameters:

`component` - The band for which to set horizontal subsampling.
`subsample` - The horizontal subsampling factor.

---

getHorizontalSubsampling

```
public int getHorizontalSubsampling(int component)
```

Get the horizontal subsampling factor for a band.

Parameters:

`component` - The band of the image for which to retrieve subsampling.

Returns:

The horizontal subsampling factor to be applied to this band

---

setVerticalSubsampling

```
public void setVerticalSubsampling(int component,
                                   int subsample)
```

Sets the vertical subsampling to be applied to an image band. Defaults to 1 for grayscale and (1,2,2) for RGB.

Parameters:

`component` - The band for which to set vertical subsampling.
`subsample` - The vertical subsampling factor.

---

getVerticalSubsampling

```
public int getVerticalSubsampling(int component)
```

Get the vertical subsampling factor for a band.

Parameters:

`component` - The band of the image for which to retrieve subsampling.

Returns:

The vertical subsampling factor to be applied to this band

---

setLumaQTable

```
public void setLumaQTable(int[] qTable)
```

Sets the quantization table to be used for luminance data. This is a convenience method which explicitly sets the contents of quantization table 0. The length of the table must be 64. This disables any quality setting.

Parameters:
`qTable` - Quantization table values in "zig-zag" order.

---

setChromaQTable

```
public void setChromaQTable(int[] qTable)
```

Sets the quantization table to be used for chrominance data. This is a convenience method which explicitly sets the contents of quantization table 1. The length of the table must be 64. This method assumes that all chroma components will use the same table. This disables any quality setting.

Parameters:
`qTable` - Quantization table values in "zig-zag" order.

---

setQTable

```
public void setQTable(int component,
                      int tableSlot,
                      int[] qTable)
```

Sets a quantization table to be used for a component. This method allows up to four independent tables to be specified. This disables any quality setting.

Parameters:
`component` - The band to which this table applies.
`tableSlot` - The table number that this table is assigned to (0 to 3).
`qTable` - Quantization table values in "zig-zag" order.

---

isQTableSet

```
public boolean isQTableSet(int component)
```

Tests if a Quantization table has been set.

Returns:
Returns true is the specified quantization table has been set.

---

getQTable

```
public int[] getQTable(int component)
```

Retrieve the contents of the quantization table used for a component.

Parameters:
`component` - The band to which this table applies.

Returns:
The contents of the quantization table as a reference.

Throws:
java.lang.IllegalStateException - if table has not been previously set for this component.

---

getQTableSlot

```
public int getQTableSlot(int component)
```

Retrieve the quantization table slot used for a component.

Parameters:
`component` - The band to which this table slot applies.

Returns:
The table slot used for this band.

Throws:
java.lang.IllegalStateException - if table has not been previously set for this component.

---

setRestartInterval

```
public void setRestartInterval(int restartInterval)
```

Sets the restart interval in Minimum Coded Units (MCUs). This can be useful in some environments to limit the effect of bitstream errors to a single restart interval. The default is zero (no restart interval markers).

Parameters:
`restartInterval` - Number of MCUs between restart markers.

---

getRestartInterval

`public int getRestartInterval()`

Gets the restart interval in Minimum Coded Units (MCUs).

Returns:

The restart interval in MCUs (0 if not set).

---

setQuality

`public void setQuality(float quality)`

This creates new quantization tables that replace the currently installed quantization tables. The created quantization table varies from very high compression, very low quality, (0.0) to low compression, very high quality (1.0) based on the quality parameter.

At a quality level of 1.0 the table will be all 1's which will lead to no loss of data due to quantization (however chrominace subsampling, if used, and roundoff error in the DCT will still degrade the image some what).

The default setting is 0.75 which provides high quality while insuring a good compression ratio.

```
Some guidelines: 0.75 high quality
                 0.5  medium quality
                 0.25 low quality
```

Parameters:

`quality` - 0.0-1.0 setting of desired quality level.

---

isQualitySet

`public boolean isQualitySet()`

Tests if the quality parameter has been set in this JPEGEncodeParam.

Returns:

True/false flag indicating if quality has been set.

---

getQuality

`public float getQuality()`

Retrieve the quality setting for this encoding. This is a number between 0.0 and 1.0.

Returns:

The specified quality setting (0.75 if not set).

---

setWriteTablesOnly

`public void setWriteTablesOnly(boolean tablesOnly)`

Instructs the encoder to write only the table data to the output stream. This is considered an abbreviated JPEG stream. Defaults to false -- normally both tables and encoded image data are written.

Parameters:

`tablesOnly` - If true, only the tables will be written.

---

getWriteTablesOnly

`public boolean getWriteTablesOnly()`

Retrieve the setting of the writeTablesOnly flag.

Returns:

The setting of the writeTablesOnly flag (false if not set).

---

setWriteImageOnly

`public void setWriteImageOnly(boolean imageOnly)`

Controls whether the encoder writes only the compressed image data to the output stream. This is considered an abbreviated JPEG stream. Defaults to false -- normally both tables and compressed image data are written.

Parameters:

`imageOnly` - If true, only the compressed image will be written.

---

getWriteImageOnly

`public boolean **getWriteImageOnly**()`

Retrieve the setting of the writeImageOnly flag.

Returns:

The setting of the writeImageOnly flag (false if not set).

---

setWriteJFIFHeader

`public void **setWriteJFIFHeader**(boolean writeJFIF)`

Controls whether the encoder writes a JFIF header using the APP0 marker. By default an APP0 marker is written to create a JFIF file.

Parameters:

`writeJFIF` - If true, writes a JFIF header.

---

getWriteJFIFHeader

`public boolean **getWriteJFIFHeader**()`

Retrieve the setting of the writeJFIF flag.

Returns:

The setting of the writeJFIF flag (true if not set).

com.sun.media.jai.codec
Class JaiI18N

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.JaiI18N
```

---

class **JaiI18N**
extends java.lang.Object

Field Detail

packageName

`static java.lang.String packageName`

Constructor Detail

JaiI18N

`JaiI18N()`

Method Detail

getString

`public static java.lang.String getString(java.lang.String key)`

com.sun.media.jai.codec

Class MemoryCacheSeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.MemoryCacheSeekableStream
```

public final class **MemoryCacheSeekableStream**
extends SeekableStream

A subclass of `SeekableStream` that may be used to wrap a regular `InputStream`. Seeking backwards is supported by means of an in-memory cache. For greater efficiency, `FileCacheSeekableStream` should be used in circumstances that allow the creation of a temporary file.

The `mark()` and `reset()` methods are supported.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

src

`private java.io.InputStream **src**`

The source input stream.

pointer

`private long **pointer**`

Position of first unread byte.

SECTOR_SHIFT

`private static final int **SECTOR_SHIFT**`

Log_2 of the sector size.

SECTOR_SIZE

`private static final int **SECTOR_SIZE**`

The sector size.

SECTOR_MASK

`private static final int **SECTOR_MASK**`

A mask to determine the offset within a sector.

data

`private java.util.Vector **data**`

A Vector of source sectors.

sectors

`int **sectors**`

Number of sectors stored.

length

`int length`

Number of bytes read.

---

foundEOS

`boolean foundEOS`

True if we've previously reached the end of the source stream

Constructor Detail

MemoryCacheSeekableStream

`public MemoryCacheSeekableStream(java.io.InputStream src)`

Constructs a `MemoryCacheSeekableStream` that takes its source data from a regular `InputStream`. Seeking backwards is supported by means of an in-memory cache.

Method Detail

readUntil

```
private long readUntil(long pos)
                throws java.io.IOException
```

Ensures that at least `pos` bytes are cached, or the end of the source is reached. The return value is equal to the smaller of `pos` and the length of the source stream.

---

canSeekBackwards

`public boolean canSeekBackwards()`

Returns `true` since all `MemoryCacheSeekableStream` instances support seeking backwards.

Overrides:
canSeekBackwards in class SeekableStream

---

getFilePointer

`public long getFilePointer()`

Returns the current offset in this file.

Returns:
the offset from the beginning of the file, in bytes, at which the next read occurs.

Overrides:
getFilePointer in class SeekableStream

---

seek

```
public void seek(long pos)
          throws java.io.IOException
```

Sets the file-pointer offset, measured from the beginning of this file, at which the next read occurs.

Parameters:
`pos` - the offset position, measured in bytes from the beginning of the file, at which to set the file pointer.

Throws:
java.io.IOException - if `pos` is less than `0` or if an I/O error occurs.

Overrides:
seek in class SeekableStream

---

read

```
public int read()
         throws java.io.IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

Returns:
  the next byte of data, or `-1` if the end of the stream is reached.
Overrides:
  read in class SeekableStream

---

read

```
public int read(byte[] b,
                int off,
                int len)
         throws java.io.IOException
```

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

Parameters:
  `b` - the buffer into which the data is read.
  `off` - the start offset in array `b` at which the data is written.
  `len` - the maximum number of bytes to read.
Returns:
  the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.
Overrides:
  read in class SeekableStream

com.sun.media.jai.codec
Class PNGDecodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGDecodeParam
```

public class **PNGDecodeParam**
extends java.lang.Object
implements ImageDecodeParam

An instance of `ImageDecodeParam` for decoding images in the PNG format. `PNGDecodeParam` allows several aspects of the decoding process for PNG images to be controlled. By default, decoding produces output images with the following properties:

Images with a bit depth of 8 or less use a `DataBufferByte` to hold the pixel data. 16-bit images use a `DataBufferUShort`.

Palette color images and non-transparent grayscale images with bit depths of 1, 2, or 4 will have a `MultiPixelPackedSampleModel` and an `IndexColorModel`. For palette color images, the `ColorModel` palette contains the red, green, blue, and optionally alpha palette information. For grayscale images, the palette is used to expand the pixel data to cover the range 0-255. The pixels are stored packed 8, 4, or 2 to the byte.

All other images are stored using a `PixelInterleavedSampleModel` with each sample of a pixel occupying its own `byte` or `short` within the `DataBuffer`. A `ComponentColorModel` is used which simply extracts the red, green, blue, gray, and/or alpha information from separate `DataBuffer` entries.

Five aspects of this process may be altered by means of methods in this class.

`setSuppressAlpha()` prevents an alpha channel from appearing in the output.

`setExpandPalette()` turns palette-color images into 3-or 4-channel full-color images.

`setOutput8BitGray()` causes 1, 2, or 4 bit grayscale images to be output in 8-bit form, using a `ComponentSampleModel` and `ComponentColorModel`.

`setDecodingExponent()` causes the output image to be gamma-corrected using a supplied output gamma value.

`setExpandGrayAlpha()` causes 2-channel gray/alpha (GA) images to be output as full-color (GGGA) images, which may simplify further processing and display.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

suppressAlpha

`private boolean` **`suppressAlpha`**

expandPalette

`private boolean` **`expandPalette`**

output8BitGray

`private boolean` **`output8BitGray`**

performGammaCorrection

`private boolean` **`performGammaCorrection`**

userExponent

`private float` **`userExponent`**

displayExponent

`private float displayExponent`

---

expandGrayAlpha

`private boolean expandGrayAlpha`

---

generateEncodeParam

`private boolean generateEncodeParam`

---

encodeParam

`private PNGEncodeParam encodeParam`

Constructor Detail

PNGDecodeParam

`public PNGDecodeParam()`

Constructs a default instance of `PNGDecodeParam`.

Method Detail

getSuppressAlpha

`public boolean getSuppressAlpha()`

Returns `true` if alpha (transparency) will be prevented from appearing in the output.

---

setSuppressAlpha

`public void setSuppressAlpha(boolean suppressAlpha)`

If set, no alpha (transparency) channel will appear in the output image.

The default is to allow transparency to appear in the output image.

---

getExpandPalette

`public boolean getExpandPalette()`

Returns true if palette-color images will be expanded to produce full-color output.

---

setExpandPalette

`public void setExpandPalette(boolean expandPalette)`

If set, palette color images (PNG color type 3) will be decoded into full-color (RGB) output images. The output image may have 3 or 4 channels, depending on the presence of transparency information.

The default is to output palette images using a single channel. The palette information is used to construct the output image's `ColorModel`.

---

getOutput8BitGray

`public boolean getOutput8BitGray()`

Returns the current value of the 8-bit gray output parameter.

---

setOutput8BitGray

`public void setOutput8BitGray(boolean output8BitGray)`

If set, grayscale images with a bit depth less than 8 (1, 2, or 4) will be output in 8 bit form. The output values will occupy the full 8-bit range. For example, gray values 0, 1, 2, and 3 of a 2-bit image will be output as 0, 85, 170, and 255.

The decoding of non-grayscale images and grayscale images with a bit depth of 8 or 16 are unaffected by this setting.

The default is not to perform expansion. Grayscale images with a depth of 1, 2, or 4 bits will be represented using a `MultiPixelPackedSampleModel` and an `IndexColorModel`.

---

getPerformGammaCorrection

`public boolean` **`getPerformGammaCorrection`**`()`

Returns `true` if gamma correction is to be performed on the image data. The default is `true`.

If gamma correction is to be performed, the `getUserExponent()` and `getDisplayExponent()` methods are used in addition to the gamma value stored within the file (or the default value of 1/2.2 used if no value is found) to produce a single exponent using the formula:

```
 decoding_exponent = user_exponent/(gamma_from_file * display_exponent)
```

---

setPerformGammaCorrection

`public void` **`setPerformGammaCorrection`**`(boolean performGammaCorrection)`

Turns gamma corection of the image data on or off.

---

getUserExponent

`public float` **`getUserExponent`**`()`

Returns the current value of the user exponent parameter. By default, the user exponent is equal to 1.0F.

---

setUserExponent

`public void` **`setUserExponent`**`(float userExponent)`

Sets the user exponent to a given value. The exponent must be positive. If not, an `IllegalArgumentException` will be thrown.

The output image pixels will be placed through a transformation of the form:

```
 sample = integer_sample / (2^bitdepth - 1.0)
 decoding_exponent = user_exponent/(gamma_from_file * display_exponent)
 output = sample ^ decoding_exponent
```

where `gamma_from_file` is the gamma of the file data, as determined by the `gAMA`, sRGB, and/or `iCCP` chunks, and `display_exponent` is the exponent of the intrinsic transfer curve of the display, generally 2.2.

Input files which do not specify any gamma are assumed to have a gamma of `1/2.2`; such images may be displayed on a CRT with an exponent of 2.2 using the default user exponent of 1.0.

The user exponent may be used in order to change the effective gamma of a file. If a file has a stored gamma of X, but the decoder believes that the true file gamma is Y, setting a user exponent of Y/X will produce the same result as changing the file gamma.

This parameter affects the decoding of all image types.

Throws:

java.lang.IllegalArgumentException - if `userExponent` is negative.

---

getDisplayExponent

`public float` **`getDisplayExponent`**`()`

Returns the current value of the display exponent parameter. By default, the display exponent is equal to 2.2F.

---

setDisplayExponent

`public void` **`setDisplayExponent`**`(float displayExponent)`

Sets the display exponent to a given value. The exponent must be positive. If not, an `IllegalArgumentException` will be thrown.

The output image pixels will be placed through a transformation of the form:

```
 sample = integer_sample / (2^bitdepth - 1.0)
 decoding_exponent = user_exponent/(gamma_from_file * display_exponent)
 output = sample ^ decoding_exponent
```

where `gamma_from_file` is the gamma of the file data, as determined by the `gAMA`, sRGB, and/or `iCCP` chunks, and `user_exponent` is an additional user-supplied parameter.

Input files which do not specify any gamma are assumed to have a gamma of `1/2.2`; such images should be decoding using the default display exponent of 2.2.

If an image is to be processed further before being displayed, it may be preferable to set the display exponent to 1.0 in order to produce a linear output image.

This parameter affects the decoding of all image types.

Throws:

java.lang.IllegalArgumentException - if `userExponent` is negative.

---

getExpandGrayAlpha

`public boolean` **`getExpandGrayAlpha`**`()`

Returns the current setting of the gray/alpha expansion.

---

setExpandGrayAlpha

`public void` **`setExpandGrayAlpha`**`(boolean expandGrayAlpha)`

If set, images containing one channel of gray and one channel of alpha (GA) will be output in a 4-channel format (GGGA). This produces output that may be simpler to process and display.

This setting affects both images of color type 4 (explicit alpha) and images of color type 0 (grayscale) that contain transparency information.

By default, no expansion is performed.

---

getGenerateEncodeParam

`public boolean` **`getGenerateEncodeParam`**`()`

Returns `true` if an instance of `PNGEncodeParam` will be available after an image has been decoded via the `getEncodeParam` method.

---

setGenerateEncodeParam

`public void` **`setGenerateEncodeParam`**`(boolean generateEncodeParam)`

If set, an instance of `PNGEncodeParam` will be available after an image has been decoded via the `getEncodeParam` method that encapsulates information about the contents of the PNG file. If not set, this information will not be recorded and `getEncodeParam()` will return `null`.

---

getEncodeParam

`public PNGEncodeParam` **`getEncodeParam`**`()`

If `getGenerateEncodeParam()` is `true`, this method may be called after decoding has completed, and will return an instance of `PNGEncodeParam` containing information about the contents of the PNG file just decoded.

---

setEncodeParam

`public void` **`setEncodeParam`**`(PNGEncodeParam encodeParam)`

Sets the current encoder param instance. This method is intended to be called by the PNG decoder and will overwrite the current instance returned by `getEncodeParam`.

com.sun.media.jai.codec
Class PNGEncodeParam.Gray

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGEncodeParam
        |
        +--com.sun.media.jai.codec.PNGEncodeParam.Gray
```

---

public static class **PNGEncodeParam.Gray**
extends PNGEncodeParam

Field Detail

backgroundSet
`private boolean backgroundSet`

---

backgroundPaletteGray
`private int backgroundPaletteGray`

---

transparency
`private int[] transparency`

---

bitShift
`private int bitShift`

---

bitShiftSet
`private boolean bitShiftSet`

Constructor Detail

PNGEncodeParam.Gray
`public PNGEncodeParam.Gray()`

Constructs an instance of `PNGEncodeParam.Gray`.

Method Detail

unsetBackground
`public void unsetBackground()`

Suppresses the ’bKGD’ chunk from being output.
Overrides:
unsetBackground in class PNGEncodeParam

---

isBackgroundSet
`public boolean isBackgroundSet()`

Returns true if a ’bKGD’ chunk will be output.
Overrides:
isBackgroundSet in class PNGEncodeParam

---

setBitDepth

`public void **setBitDepth**(int bitDepth)`

Sets the desired bit depth for a grayscale image. The bit depth must be one of 1, 2, 4, 8, or 16.

When encoding a source image of a greater bit depth, pixel values will be clamped to the smaller range after shifting by the value given by `getBitShift()`. When encoding a source image of a smaller bit depth, pixel values will be shifted and left-filled with zeroes.

Overrides:
setBitDepth in class PNGEncodeParam

---

setBackgroundGray

`public void **setBackgroundGray**(int gray)`

Sets the suggested gray level of the background.

The 'bKGD' chunk will encode this information.

---

getBackgroundGray

`public int **getBackgroundGray**()`

Returns the suggested gray level of the background.

If the background gray level has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:
java.lang.IllegalStateException - if the background gray level is not set.

---

setTransparentGray

`public void **setTransparentGray**(int transparentGray)`

Sets the gray value to be used to denote transparency.

Setting this attribute will cause the alpha channel of the input image to be ignored.

The 'tRNS' chunk will encode this information.

---

getTransparentGray

`public int **getTransparentGray**()`

Returns the gray value to be used to denote transparency.

If the transparent gray value has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:
java.lang.IllegalStateException - if the transparent gray value is not set.

---

setBitShift

`public void **setBitShift**(int bitShift)`

Sets the desired bit shift for a grayscale image. Pixels in the source image will be shifted right by the given amount prior to being clamped to the maximum value given by the encoded image's bit depth.

---

getBitShift

`public int **getBitShift**()`

Returns the desired bit shift for a grayscale image.

If the bit shift has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:
java.lang.IllegalStateException - if the bit shift is not set.

---

unsetBitShift

`public void **unsetBitShift**()`

Suppresses the setting of the bit shift of a grayscale image. Pixels in the source image will not be shifted prior to encoding.

---

isBitShiftSet

`public boolean **isBitShiftSet**()`

Returns true if the bit shift has been set.

---

isBitDepthSet

`public boolean **isBitDepthSet**()`

Returns true if the bit depth has been set.

com.sun.media.jai.codec
Class PNGEncodeParam.Palette

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGEncodeParam
        |
        +--com.sun.media.jai.codec.PNGEncodeParam.Palette
```

public static class **PNGEncodeParam.Palette**
extends PNGEncodeParam

Field Detail

backgroundSet
`private boolean backgroundSet`

palette
`private int[] palette`

paletteSet
`private boolean paletteSet`

backgroundPaletteIndex
`private int backgroundPaletteIndex`

transparency
`private int[] transparency`

Constructor Detail

PNGEncodeParam.Palette
`public PNGEncodeParam.Palette()`

Constructs an instance of `PNGEncodeParam.Palette`.

Method Detail

unsetBackground
`public void unsetBackground()`

Suppresses the ’bKGD’ chunk from being output.
Overrides:
unsetBackground in class PNGEncodeParam

isBackgroundSet
`public boolean isBackgroundSet()`

Returns true if a ’bKGD’ chunk will be output.
Overrides:
isBackgroundSet in class PNGEncodeParam

setBitDepth

`public void **setBitDepth**(int bitDepth)`

Sets the desired bit depth for a palette image. The bit depth must be one of 1, 2, 4, or 8, or else an `IllegalArgumentException` will be thrown.

Overrides:

setBitDepth in class PNGEncodeParam

---

setPalette

`public void **setPalette**(int[] rgb)`

Sets the RGB palette of the image to be encoded. The `rgb` parameter contains alternating R, G, B values for each color index used in the image. The number of elements must be a multiple of 3 between 3 and 3*256.

The 'PLTE' chunk will encode this information.

Parameters:

`rgb` - An array of `int`s.

---

getPalette

`public int[] **getPalette**()`

Returns the current RGB palette.

If the palette has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Returns:

An array of `int`s.

Throws:

java.lang.IllegalStateException - if the palette is not set.

---

unsetPalette

`public void **unsetPalette**()`

Suppresses the 'PLTE' chunk from being output.

---

isPaletteSet

`public boolean **isPaletteSet**()`

Returns true if a 'PLTE' chunk will be output.

---

setBackgroundPaletteIndex

`public void **setBackgroundPaletteIndex**(int index)`

Sets the palette index of the suggested background color.

The 'bKGD' chunk will encode this information.

---

getBackgroundPaletteIndex

`public int **getBackgroundPaletteIndex**()`

Returns the palette index of the suggested background color.

If the background palette index has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the palette index is not set.

---

setPaletteTransparency

`public void **setPaletteTransparency**(byte[] alpha)`

Sets the alpha values associated with each palette entry. The `alpha` parameter should have as many entries as there are RGB triples in the palette.

The 'tRNS' chunk will encode this information.

getPaletteTransparency

```
public byte[] getPaletteTransparency()
```

Returns the alpha values associated with each palette entry.

If the palette transparency has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the palette transparency is not set.

com.sun.media.jai.codec
Class PNGEncodeParam.RGB

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGEncodeParam
        |
        +--com.sun.media.jai.codec.PNGEncodeParam.RGB
```

public static class **PNGEncodeParam.RGB**
extends PNGEncodeParam

Field Detail

backgroundSet

`private boolean backgroundSet`

backgroundRGB

`private int[] backgroundRGB`

transparency

`private int[] transparency`

Constructor Detail

PNGEncodeParam.RGB

`public PNGEncodeParam.RGB()`

Constructs an instance of `PNGEncodeParam.RGB`.

Method Detail

unsetBackground

`public void unsetBackground()`

Suppresses the ’bKGD’ chunk from being output.
Overrides:
unsetBackground in class PNGEncodeParam

isBackgroundSet

`public boolean isBackgroundSet()`

Returns true if a ’bKGD’ chunk will be output.
Overrides:
isBackgroundSet in class PNGEncodeParam

setBitDepth

`public void setBitDepth(int bitDepth)`

Sets the desired bit depth for an RGB image. The bit depth must be 8 or 16.
Overrides:
setBitDepth in class PNGEncodeParam

setBackgroundRGB

`public void **setBackgroundRGB**(int[] rgb)`

Sets the RGB value of the suggested background color. The `rgb` parameter should have 3 entries.

The ’bKGD’ chunk will encode this information.

---

getBackgroundRGB

`public int[] **getBackgroundRGB**()`

Returns the RGB value of the suggested background color.

If the background color has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the background color is not set.

---

setTransparentRGB

`public void **setTransparentRGB**(int[] transparentRGB)`

Sets the RGB value to be used to denote transparency.

Setting this attribute will cause the alpha channel of the input image to be ignored.

The ’tRNS’ chunk will encode this information.

---

getTransparentRGB

`public int[] **getTransparentRGB**()`

Returns the RGB value to be used to denote transparency.

If the transparent color has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the transparent color is not set.

com.sun.media.jai.codec

Class PNGEncodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGEncodeParam
```

Direct Known Subclasses:

PNGEncodeParam.Gray, PNGEncodeParam.Palette, PNGEncodeParam.RGB

---

public abstract class **PNGEncodeParam**
extends java.lang.Object
implements ImageEncodeParam

An instance of `ImageEncodeParam` for encoding images in the PNG format.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

INTENT_PERCEPTUAL

`public static final int` **`INTENT_PERCEPTUAL`**

Constant for use with the sRGB chunk.

INTENT_RELATIVE

`public static final int` **`INTENT_RELATIVE`**

Constant for use with the sRGB chunk.

INTENT_SATURATION

`public static final int` **`INTENT_SATURATION`**

Constant for use with the sRGB chunk.

INTENT_ABSOLUTE

`public static final int` **`INTENT_ABSOLUTE`**

Constant for use with the sRGB chunk.

PNG_FILTER_NONE

`public static final int` **`PNG_FILTER_NONE`**

Constant for use in filtering.

PNG_FILTER_SUB

`public static final int` **`PNG_FILTER_SUB`**

Constant for use in filtering.

PNG_FILTER_UP

`public static final int` **`PNG_FILTER_UP`**

Constant for use in filtering.

PNG_FILTER_AVERAGE

`public static final int` **`PNG_FILTER_AVERAGE`**

Constant for use in filtering.

PNG_FILTER_PAETH

`public static final int` **`PNG_FILTER_PAETH`**

Constant for use in filtering.

---

bitDepth

`protected int` **`bitDepth`**

---

bitDepthSet

`protected boolean` **`bitDepthSet`**

---

useInterlacing

`private boolean` **`useInterlacing`**

---

chromaticity

`private float[]` **`chromaticity`**

---

chromaticitySet

`private boolean` **`chromaticitySet`**

---

gamma

`private float` **`gamma`**

---

gammaSet

`private boolean` **`gammaSet`**

---

paletteHistogram

`private int[]` **`paletteHistogram`**

---

paletteHistogramSet

`private boolean` **`paletteHistogramSet`**

ICCProfileData

`private byte[]` **`ICCProfileData`**

ICCProfileDataSet

`private boolean` **`ICCProfileDataSet`**

---

physicalDimension

`private int[]` **`physicalDimension`**

---

physicalDimensionSet

`private boolean` **`physicalDimensionSet`**

---

suggestedPalette

`private PNGSuggestedPaletteEntry[]` **`suggestedPalette`**

---

suggestedPaletteSet

`private boolean` **`suggestedPaletteSet`**

---

significantBits

`private int[]` **`significantBits`**

---

significantBitsSet

`private boolean` **`significantBitsSet`**

SRGBIntent

`private int` **`SRGBIntent`**

SRGBIntentSet

`private boolean` **`SRGBIntentSet`**

---

text

`private java.lang.String[]` **`text`**

---

textSet

`private boolean` **`textSet`**

---

modificationTime

`private java.util.Date` **`modificationTime`**

---

modificationTimeSet

`private boolean` **`modificationTimeSet`**

---

transparencySet

`boolean` **`transparencySet`**

---

zText

`private java.lang.String[]` **`zText`**

---

zTextSet

`private boolean` **`zTextSet`**

---

chunkType

`java.util.Vector **chunkType**`

---

chunkData

`java.util.Vector **chunkData**`

Constructor Detail

PNGEncodeParam

`public **PNGEncodeParam**()`

Method Detail

getDefaultEncodeParam

`public static PNGEncodeParam **getDefaultEncodeParam**(java.awt.image.RenderedImage im)`

Returns an instance of `PNGEncodeParam.Palette`, `PNGEncodeParam.Gray`, or `PNGEncodeParam.RGB` appropriate for encoding the given image.

If the image has an `IndexColorModel`, an instance of `PNGEncodeParam.Palette` is returned. Otherwise, if the image has 1 or 2 bands an instance of `PNGEncodeParam.Gray` is returned. In all other cases an instance of `PNGEncodeParam.RGB` is returned.

Note that this method does not provide any guarantee that the given image will be successfully encoded by the PNG encoder, as it only performs a very superficial analysis of the image structure.

---

setBitDepth

`public abstract void **setBitDepth**(int bitDepth)`

Sets the desired bit depth of an image.

---

getBitDepth

`public int **getBitDepth**()`

Returns the desired bit depth for a grayscale image.

If the bit depth has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the bit depth is not set.

---

unsetBitDepth

`public void **unsetBitDepth**()`

Suppresses the setting of the bit depth of a grayscale image. The depth of the encoded image will be inferred from the source image bit depth, rounded up to the next power of 2 between 1 and 16.

---

setInterlacing

`public void **setInterlacing**(boolean useInterlacing)`

Turns Adam7 interlacing on or off.

---

getInterlacing

`public boolean **getInterlacing**()`

Returns `true` if Adam7 interlacing will be used.

---

unsetBackground

```
public void unsetBackground()
```

Suppresses the 'bKGD' chunk from being output. For API compatibility with JAI 1.0, the superclass defines this method to throw a `RuntimeException`; accordingly, subclasses must provide their own implementations.

---

isBackgroundSet

```
public boolean isBackgroundSet()
```

Returns true if a 'bKGD' chunk will be output. For API compatibility with JAI 1.0, the superclass defines this method to throw a `RuntimeException`; accordingly, subclasses must provide their own implementations.

---

setChromaticity

```
public void setChromaticity(float[] chromaticity)
```

Sets the white point and primary chromaticities in CIE (x, y) space.

The `chromaticity` parameter should be a `float` array of length 8 containing the white point X and Y, red X and Y, green X and Y, and blue X and Y values in order.

The 'cHRM' chunk will encode this information.

---

setChromaticity

```
public void setChromaticity(float whitePointX,
                            float whitePointY,
                            float redX,
                            float redY,
                            float greenX,
                            float greenY,
                            float blueX,
                            float blueY)
```

A convenience method that calls the array version.

---

getChromaticity

```
public float[] getChromaticity()
```

Returns the white point and primary chromaticities in CIE (x, y) space.

See the documentation for the `setChromaticity` method for the format of the returned data.

If the chromaticity has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the chromaticity is not set.

---

unsetChromaticity

```
public void unsetChromaticity()
```

Suppresses the 'cHRM' chunk from being output.

---

isChromaticitySet

```
public boolean isChromaticitySet()
```

Returns true if a 'cHRM' chunk will be output.

---

setGamma

```
public void setGamma(float gamma)
```

Sets the file gamma value for the image.

The 'gAMA' chunk will encode this information.

---

getGamma

`public float getGamma()`

Returns the file gamma value for the image.

If the file gamma has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the gamma is not set.

---

unsetGamma

`public void unsetGamma()`

Suppresses the ’gAMA’ chunk from being output.

---

isGammaSet

`public boolean isGammaSet()`

Returns true if a ’gAMA’ chunk will be output.

---

setPaletteHistogram

`public void setPaletteHistogram(int[] paletteHistogram)`

Sets the palette histogram to be stored with this image. The histogram consists of an array of integers, one per palette entry.

The ’hIST’ chunk will encode this information.

---

getPaletteHistogram

`public int[] getPaletteHistogram()`

Returns the palette histogram to be stored with this image.

If the histogram has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the histogram is not set.

---

unsetPaletteHistogram

`public void unsetPaletteHistogram()`

Suppresses the ’hIST’ chunk from being output.

---

isPaletteHistogramSet

`public boolean isPaletteHistogramSet()`

Returns true if a ’hIST’ chunk will be output.

---

setICCProfileData

`public void setICCProfileData(byte[] ICCProfileData)`

Sets the ICC profile data to be stored with this image. The profile is represented in raw binary form.

The ’iCCP’ chunk will encode this information.

---

getICCProfileData

`public byte[] getICCProfileData()`

Returns the ICC profile data to be stored with this image.

If the ICC profile has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the ICC profile is not set.

---

unsetICCProfileData

`public void` **`unsetICCProfileData`**`()`

Suppresses the 'iCCP' chunk from being output.

---

isICCProfileDataSet

`public boolean` **`isICCProfileDataSet`**`()`

Returns true if a 'iCCP' chunk will be output.

---

setPhysicalDimension

`public void` **`setPhysicalDimension`**`(int[] physicalDimension)`

Sets the physical dimension information to be stored with this image. The physicalDimension parameter should be a 3-entry array containing the number of pixels per unit in the X direction, the number of pixels per unit in the Y direction, and the unit specifier (0 = unknown, 1 = meters).

The 'pHYS' chunk will encode this information.

---

setPhysicalDimension

```
public void setPhysicalDimension(int xPixelsPerUnit,
                                 int yPixelsPerUnit,
                                 int unitSpecifier)
```

A convenience method that calls the array version.

---

getPhysicalDimension

`public int[]` **`getPhysicalDimension`**`()`

Returns the physical dimension information to be stored with this image.

If the physical dimension information has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the physical dimension information is not set.

---

unsetPhysicalDimension

`public void` **`unsetPhysicalDimension`**`()`

Suppresses the 'pHYS' chunk from being output.

---

isPhysicalDimensionSet

`public boolean` **`isPhysicalDimensionSet`**`()`

Returns true if a 'pHYS' chunk will be output.

---

setSuggestedPalette

`public void` **`setSuggestedPalette`**`(PNGSuggestedPaletteEntry[] palette)`

Sets the suggested palette information to be stored with this image. The information is passed to this method as an array of `PNGSuggestedPaletteEntry` objects.

The 'sPLT' chunk will encode this information.

---

getSuggestedPalette

`public PNGSuggestedPaletteEntry[]` **`getSuggestedPalette`**`()`

Returns the suggested palette information to be stored with this image.

If the suggested palette information has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the suggested palette information is not set.

unsetSuggestedPalette

`public void **unsetSuggestedPalette**()`

Suppresses the 'sPLT' chunk from being output.

isSuggestedPaletteSet

`public boolean **isSuggestedPaletteSet**()`

Returns true if a 'sPLT' chunk will be output.

setSignificantBits

`public void **setSignificantBits**(int[] significantBits)`

Sets the number of significant bits for each band of the image.

The number of entries in the `significantBits` array must be equal to the number of output bands in the image: 1 for a gray image, 2 for gray+alpha, 3 for index or truecolor, and 4 for truecolor+alpha.

The 'sBIT' chunk will encode this information.

getSignificantBits

`public int[] **getSignificantBits**()`

Returns the number of significant bits for each band of the image.

If the significant bits values have not previously been set, or have been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the significant bits values are not set.

unsetSignificantBits

`public void **unsetSignificantBits**()`

Suppresses the 'sBIT' chunk from being output.

isSignificantBitsSet

`public boolean **isSignificantBitsSet**()`

Returns true if an 'sBIT' chunk will be output.

setSRGBIntent

`public void **setSRGBIntent**(int SRGBIntent)`

Sets the sRGB rendering intent to be stored with this image. The legal values are 0 = Perceptual, 1 = Relative Colorimetric, 2 = Saturation, and 3 = Absolute Colorimetric. Refer to the PNG specification for information on these values.

The 'sRGB' chunk will encode this information.

getSRGBIntent

`public int **getSRGBIntent**()`

Returns the sRGB rendering intent to be stored with this image.

If the sRGB intent has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the sRGB intent is not set.

unsetSRGBIntent

`public void **unsetSRGBIntent**()`

Suppresses the 'sRGB' chunk from being output.

isSRGBIntentSet

`public boolean **isSRGBIntentSet**()`

Returns true if an 'sRGB' chunk will be output.

---

setText

`public void **setText**(java.lang.String[] text)`

Sets the textual data to be stored in uncompressed form with this image. The data is passed to this method as an array of `String`s.

The 'tEXt' chunk will encode this information.

---

getText

`public java.lang.String[] **getText**()`

Returns the text strings to be stored in uncompressed form with this image as an array of `String`s.

If the text strings have not previously been set, or have been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the text strings are not set.

---

unsetText

`public void **unsetText**()`

Suppresses the 'tEXt' chunk from being output.

---

isTextSet

`public boolean **isTextSet**()`

Returns true if a 'tEXt' chunk will be output.

---

setModificationTime

`public void **setModificationTime**(java.util.Date modificationTime)`

Sets the modification time, as a `Date`, to be stored with this image. The internal storage format will use UTC regardless of how the `modificationTime` parameter was created.

The 'tIME' chunk will encode this information.

---

getModificationTime

`public java.util.Date **getModificationTime**()`

Returns the modification time to be stored with this image.

If the bit depth has not previously been set, or has been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the bit depth is not set.

---

unsetModificationTime

`public void **unsetModificationTime**()`

Suppresses the 'tIME' chunk from being output.

---

isModificationTimeSet

`public boolean **isModificationTimeSet**()`

Returns true if a 'tIME' chunk will be output.

---

unsetTransparency

`public void` **`unsetTransparency`**`()`

Suppresses the ’tRNS’ chunk from being output.

---

isTransparencySet

`public boolean` **`isTransparencySet`**`()`

Returns true if a ’tRNS’ chunk will be output.

---

setCompressedText

`public void` **`setCompressedText`**`(java.lang.String[] text)`

Sets the text strings to be stored in compressed form with this image. The data is passed to this method as an array of `String`s.

The ’zTXt’ chunk will encode this information.

---

getCompressedText

`public java.lang.String[]` **`getCompressedText`**`()`

Returns the text strings to be stored in compressed form with this image as an array of `String`s.

If the compressed text strings have not previously been set, or have been unset, an `IllegalStateException` will be thrown.

Throws:

java.lang.IllegalStateException - if the compressed text strings are not set.

---

unsetCompressedText

`public void` **`unsetCompressedText`**`()`

Suppresses the ’zTXt’ chunk from being output.

---

isCompressedTextSet

`public boolean` **`isCompressedTextSet`**`()`

Returns true if a ’zTXT’ chunk will be output.

---

addPrivateChunk

```
public void addPrivateChunk(java.lang.String type,
                            byte[] data)
```

Adds a private chunk, in binary form, to the list of chunks to be stored with this image.

Parameters:

`type` - a 4-character String giving the chunk type name.
`data` - an array of `byte`s containing the chunk data.

---

getNumPrivateChunks

`public int` **`getNumPrivateChunks`**`()`

Returns the number of private chunks to be written to the output file.

---

getPrivateChunkType

`public java.lang.String` **`getPrivateChunkType`**`(int index)`

Returns the type of the private chunk at a given index, as a 4-character `String`. The index must be smaller than the return value of `getNumPrivateChunks`.

---

getPrivateChunkData

```
public byte[] getPrivateChunkData(int index)
```

Returns the data associated of the private chunk at a given index, as an array of `byte`s. The index must be smaller than the return value of `getNumPrivateChunks`.

removeUnsafeToCopyPrivateChunks

```
public void removeUnsafeToCopyPrivateChunks()
```

Remove all private chunks associated with this parameter instance whose 'safe-to-copy' bit is not set. This may be advisable when transcoding PNG images.

removeAllPrivateChunks

```
public void removeAllPrivateChunks()
```

Remove all private chunks associated with this parameter instance.

abs

```
private static final int abs(int x)
```

An abs() function for use by the Paeth predictor.

paethPredictor

```
public static final int paethPredictor(int a,
                                       int b,
                                       int c)
```

The Paeth predictor routine used in PNG encoding. This routine is included as a convenience to subclasses that override the `filterRow` method.

filterRow

```
public int filterRow(byte[] currRow,
                     byte[] prevRow,
                     byte[][] scratchRows,
                     int bytesPerRow,
                     int bytesPerPixel)
```

Performs filtering on a row of an image. This method may be overridden in order to provide a custom algorithm for choosing the filter type for a given row.

The method is supplied with the current and previous rows of the image. For the first row of the image, or of an interlacing pass, the previous row array will be filled with zeros as required by the PNG specification.

The method is also supplied with five scratch arrays. These arrays may be used within the method for any purpose. At method exit, the array at the index given by the return value of the method should contain the filtered data. The return value will also be used as the filter type.

The default implementation of the method performs a trial encoding with each of the filter types, and computes the sum of absolute values of the differences between the raw bytes of the current row and the predicted values. The index of the filter producing the smallest result is returned.

As an example, to perform only 'sub' filtering, this method could be implemented (non-optimally) as follows:

```
for (int i = bytesPerPixel; i < bytesPerRow + bytesPerPixel; i++) {
    int curr = currRow[i] & 0xff;
    int left = currRow[i - bytesPerPixel] & 0xff;
    scratchRow[PNG_FILTER_SUB][i] = (byte)(curr - left);
}
return PNG_FILTER_SUB;
```

Parameters:

`currRow` - The current row as an array of `byte`s of length at least `bytesPerRow + bytesPerPixel`. The pixel data starts at index `bytesPerPixel`; the initial `bytesPerPixel` bytes are zero.

`prevRow` - The current row as an array of `byte`s The pixel data starts at index `bytesPerPixel`; the initial `bytesPerPixel` bytes are zero.

`scratchRows` - An array of 5 `byte` arrays of length at least `bytesPerRow + bytesPerPixel`, useable to hold temporary results. The filtered row will be returned as one of the entries of this array. The returned filtered data should start at index `bytesPerPixel`; The initial `bytesPerPixel` bytes are not used.

`bytesPerRow` - The number of bytes in the image row. This value will always be greater than 0.

`bytesPerPixel` - The number of bytes representing a single pixel, rounded up to an integer. This is the 'bpp' parameter described in the PNG specification.

Returns:

The filter type to be used. The entry of `scratchRows[]` at this index holds the filtered data.

com.sun.media.jai.codec

Class PNGSuggestedPaletteEntry

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNGSuggestedPaletteEntry
```

public class **PNGSuggestedPaletteEntry**
extends java.lang.Object
implements java.io.Serializable

A class representing the fields of a PNG suggested palette entry.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

name

`public java.lang.String name`

The name of the entry.

sampleDepth

`public int sampleDepth`

The depth of the color samples.

red

`public int red`

The red color value of the entry.

green

`public int green`

The green color value of the entry.

blue

`public int blue`

The blue color value of the entry.

alpha

`public int alpha`

The alpha opacity value of the entry.

frequency

`public int frequency`

The probable frequency of the color in the image.

Constructor Detail

PNGSuggestedPaletteEntry

`public PNGSuggestedPaletteEntry()`

com.sun.media.jai.codec

Class PNMEncodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.PNMEncodeParam
```

public class **PNMEncodeParam**
extends java.lang.Object
implements ImageEncodeParam

An instance of `ImageEncodeParam` for encoding images in the PNM format.

This class allows for the specification of whether to encode in the ASCII or raw variants of the PBM, PGM, and PPM formats. By default, raw encoding is used.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

raw

`private boolean raw`

Constructor Detail

PNMEncodeParam

`public PNMEncodeParam()`

Constructs a PNMEncodeParam object with default values for parameters.

Method Detail

setRaw

`public void setRaw(boolean raw)`

Sets the representation to be used. If the `raw` parameter is `true`, raw encoding will be used; otherwise ASCII encoding will be used.

Parameters:

`raw` - `true` if raw format is to be used.

getRaw

`public boolean getRaw()`

Returns the value of the `raw` parameter.

com.sun.media.jai.codec

Class SectorStreamSegmentMapper

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.SectorStreamSegmentMapper
```

---

class **SectorStreamSegmentMapper**
extends java.lang.Object
implements StreamSegmentMapper

An implementation of the `StreamSegmentMapper` interface for segments of equal length.

Field Detail

segmentPositions

```
long[] segmentPositions
```

---

segmentLength

```
int segmentLength
```

---

totalLength

```
int totalLength
```

---

lastSegmentLength

```
int lastSegmentLength
```

Constructor Detail

SectorStreamSegmentMapper

```
public SectorStreamSegmentMapper(long[] segmentPositions,
                                 int segmentLength,
                                 int totalLength)
```

Method Detail

getStreamSegment

```
public StreamSegment getStreamSegment(long position,
                                      int length)
```

Specified by:
getStreamSegment in interface StreamSegmentMapper

---

getStreamSegment

```
public void getStreamSegment(long position,
                             int length,
                             StreamSegment seg)
```

Specified by:
getStreamSegment in interface StreamSegmentMapper

com.sun.media.jai.codec
Class SeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
```

Direct Known Subclasses:

ByteArraySeekableStream, FileCacheSeekableStream, FileSeekableStream, ForwardSeekableStream, MemoryCacheSeekableStream, SegmentedSeekableStream

---

public abstract class **SeekableStream**
extends java.io.InputStream
implements java.io.DataInput

An abstract subclass of `java.io.InputStream` that allows seeking within the input, similar to the `RandomAccessFile` class. Additionally, the `DataInput` interface is supported and extended to include support for little-endian representations of fundamental data types.

In addition to the familiar methods from `InputStream`, the methods `getFilePointer()`, `seek()`, are defined as in the `RandomAccessFile` class. The `canSeekBackwards()` method will return `true` if it is permissible to seek to a position earlier in the stream than the current value of `getFilePointer()`. Some subclasses of `SeekableStream` guarantee the ability to seek backwards while others may not offer this feature in the interest of providing greater efficiency for those users who do not require it.

The `DataInput` interface is supported as well. This included the `skipBytes()` and `readFully()` methods and a variety of `read` methods for various data types.

A number of concrete subclasses of `SeekableStream` are supplied in the `com.sun.media.jai.codec` package.

Three classes are provided for the purpose of adapting a standard `InputStream` to the `SeekableStream` interface. `ForwardSeekableStream` does not allows seeking backwards, but is inexpensive to use. `FileCacheSeekableStream` maintains a copy of all of the data read from the input in a temporary file; this file will be discarded automatically when the `FileSeekableStream` is finalized, or when the JVM exits normally. `FileCacheSeekableStream` is intended to be reasonably efficient apart from the unavoidable use of disk space. In circumstances where the creation of a temporary file is not possible, `MemoryCacheSeekableStream` may be used. `MemoryCacheSeekableStream` creates a potentially large in-memory buffer to store the stream data and so should be avoided when possible.

The `FileSeekableStream` class wraps a `File` or `RandomAccessFile`. It forwards requests to the real underlying file. It performs a limited amount of caching in order to avoid excessive I/O costs.

The `SegmentedSeekableStream` class performs a different sort of function. It creates a `SeekableStream` from another `SeekableStream` by selecting a series of portions or "segments". Each segment starts at a specified location within the source `SeekableStream` and extends for a specified number of bytes. The `StreamSegmentMapper` interface and `StreamSegment` class may be used to compute the segment positions dynamically.

A convenience methods, `wrapInputStream` is provided to construct a suitable `SeekableStream` instance whose data is supplied by a given `InputStream`. The caller, by means of the `canSeekBackwards` parameter, determines whether support for seeking backwards is required.

See Also:

`DataInput`, `InputStream`, `RandomAccessFile`, `ByteArraySeekableStream`, `FileCacheSeekableStream`, `FileSeekableStream`, `ForwardSeekableStream`, `MemoryCacheSeekableStream`, `SegmentedSeekableStream`, `StreamSegment`,

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

markPos

`protected long` **`markPos`**

Marked position

---

ruileBuf

```
private byte[] ruileBuf
```

Constructor Detail

SeekableStream

```
public SeekableStream()
```

Method Detail

wrapInputStream

```
public static SeekableStream wrapInputStream(java.io.InputStream is,
                                             boolean canSeekBackwards)
```

Returns a `SeekableStream` that will read from a given `InputStream`, optionally including support for seeking backwards. This is a convenience method that avoids the need to instantiate specific subclasses of `SeekableStream` depending on the current security model.

Parameters:

`is` - An `InputStream`.

`canSeekBackwards` - `true` if the ability to seek backwards in the output is required.

Returns:

An instance of `SeekableStream`.

---

read

```
public abstract int read()
                  throws java.io.IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns:

the next byte of data, or `-1` if the end of the stream is reached.

Throws:

java.io.IOException - if an I/O error occurs.

Overrides:

read in class java.io.InputStream

---

read

```
public abstract int read(byte[] b,
                         int off,
                         int len)
                  throws java.io.IOException
```

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of stream is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of stream, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of stream, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

A subclass must provide an implementation of this method.

Parameters:
`b` - the buffer into which the data is read.
`off` - the start offset in array `b` at which the data is written.
`len` - the maximum number of bytes to read.
Returns:
the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.
Throws:
java.io.IOException - if an I/O error occurs.
Overrides:
read in class java.io.InputStream

---

mark

```
public void mark(int readLimit)
```

Marks the current file position for later return using the `reset()` method.
Overrides:
mark in class java.io.InputStream

---

reset

```
public void reset()
          throws java.io.IOException
```

Returns the file position to its position at the time of the immediately previous call to the `mark()` method.
Overrides:
reset in class java.io.InputStream

---

markSupported

```
public boolean markSupported()
```

Returns `true` if marking is supported. Marking is automatically supported for `SeekableStream` subclasses that support seeking backeards. Subclasses that do not support seeking backwards but do support marking must override this method.
Overrides:
markSupported in class java.io.InputStream

---

canSeekBackwards

```
public boolean canSeekBackwards()
```

Returns `true` if this object supports calls to `seek(pos)` with an offset `pos` smaller than the current offset, as returned by `getFilePointer`.

---

getFilePointer

```
public abstract long getFilePointer()
                             throws java.io.IOException
```

Returns the current offset in this stream.
Returns:
the offset from the beginning of the stream, in bytes, at which the next read occurs.
Throws:
java.io.IOException - if an I/O error occurs.

---

seek

```
public abstract void seek(long pos)
                   throws java.io.IOException
```

Sets the offset, measured from the beginning of this stream, at which the next read occurs.

If `canSeekBackwards()` returns `false`, then setting `pos` to an offset smaller than the current value of `getFilePointer()` will have no effect.
Parameters:
`pos` - the offset position, measured in bytes from the beginning of the stream, at which to set the stream pointer.
Throws:
java.io.IOException - if `pos` is less than `0` or if an I/O error occurs.

readFully

```
public final void readFully(byte[] b)
                     throws java.io.IOException
```

Reads `b.length` bytes from this stream into the byte array, starting at the current stream pointer. This method reads repeatedly from the stream until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
readFully in interface java.io.DataInput

Parameters:
`b` - the buffer into which the data is read.

Throws:
java.io.EOFException - if this stream reaches the end before reading all the bytes.
java.io.IOException - if an I/O error occurs.

readFully

```
public final void readFully(byte[] b,
                            int off,
                            int len)
                     throws java.io.IOException
```

Reads exactly `len` bytes from this stream into the byte array, starting at the current stream pointer. This method reads repeatedly from the stream until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
readFully in interface java.io.DataInput

Parameters:
`b` - the buffer into which the data is read.
`off` - the start offset of the data.
`len` - the number of bytes to read.

Throws:
java.io.EOFException - if this stream reaches the end before reading all the bytes.
java.io.IOException - if an I/O error occurs.

skipBytes

```
public int skipBytes(int n)
              throws java.io.IOException
```

Attempts to skip over `n` bytes of input discarding the skipped bytes.

This method may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of stream before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

Specified by:
skipBytes in interface java.io.DataInput

Parameters:
`n` - the number of bytes to be skipped.

Returns:
the actual number of bytes skipped.

Throws:
java.io.IOException - if an I/O error occurs.

readBoolean

```
public final boolean readBoolean()
                          throws java.io.IOException
```

Reads a `boolean` from this stream. This method reads a single byte from the stream, starting at the current stream pointer. A value of `0` represents `false`. Any other value represents `true`. This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Specified by:
readBoolean in interface java.io.DataInput

Returns:
the `boolean` value read.

Throws:
java.io.EOFException - if this stream has reached the end.
java.io.IOException - if an I/O error occurs.

---

readByte

```
public final byte readByte()
                    throws java.io.IOException
```

Reads a signed eight-bit value from this stream. This method reads a byte from the stream, starting from the current stream pointer. If the byte read is `b`, where `0 <= b <= 255`, then the result is:

```
       (byte)(b)
```

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Specified by:
readByte in interface java.io.DataInput

Returns:
the next byte of this stream as a signed eight-bit `byte`.

Throws:
java.io.EOFException - if this stream has reached the end.
java.io.IOException - if an I/O error occurs.

---

readUnsignedByte

```
public final int readUnsignedByte()
                           throws java.io.IOException
```

Reads an unsigned eight-bit number from this stream. This method reads a byte from this stream, starting at the current stream pointer, and returns that byte.

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Specified by:
readUnsignedByte in interface java.io.DataInput

Returns:
the next byte of this stream, interpreted as an unsigned eight-bit number.

Throws:
java.io.EOFException - if this stream has reached the end.
java.io.IOException - if an I/O error occurs.

---

readShort

```
public final short readShort()
                      throws java.io.IOException
```

Reads a signed 16-bit number from this stream. The method reads two bytes from this stream, starting at the current stream pointer. If the two bytes read, in order, are `b1` and `b2`, where each of the two values is between `0` and `255`, inclusive, then the result is equal to:

```
       (short)((b1 << 8) | b2)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
readShort in interface java.io.DataInput

Returns:
the next two bytes of this stream, interpreted as a signed 16-bit number.

Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readShortLE

```
public final short readShortLE()
                        throws java.io.IOException
```

Reads a signed 16-bit number from this stream in little-endian order. The method reads two bytes from this stream, starting at the current stream pointer. If the two bytes read, in order, are `b1` and `b2`, where each of the two values is between `0` and `255`, inclusive, then the result is equal to:

```
       (short)((b2 << 8) | b1)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:
the next two bytes of this stream, interpreted as a signed 16-bit number.
Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readUnsignedShort

```
public final int readUnsignedShort()
                            throws java.io.IOException
```

Reads an unsigned 16-bit number from this stream. This method reads two bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
(b1 << 8) | b2
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.
Specified by:
readUnsignedShort in interface java.io.DataInput
Returns:
the next two bytes of this stream, interpreted as an unsigned 16-bit integer.
Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readUnsignedShortLE

```
public final int readUnsignedShortLE()
                              throws java.io.IOException
```

Reads an unsigned 16-bit number from this stream in little-endian order. This method reads two bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
(b2 << 8) | b1
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.
Returns:
the next two bytes of this stream, interpreted as an unsigned 16-bit integer.
Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readChar

```
public final char readChar()
                    throws java.io.IOException
```

Reads a Unicode character from this stream. This method reads two bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
(char)((b1 << 8) | b2)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.
Specified by:
readChar in interface java.io.DataInput
Returns:
the next two bytes of this stream as a Unicode character.
Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readCharLE

```
public final char readCharLE()
                      throws java.io.IOException
```

Reads a Unicode character from this stream in little-endian order. This method reads two bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
     (char)((b2 << 8) | b1)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.
Returns:
the next two bytes of this stream as a Unicode character.
Throws:
java.io.EOFException - if this stream reaches the end before reading two bytes.
java.io.IOException - if an I/O error occurs.

---

readInt

```
public final int readInt()
                  throws java.io.IOException
```

Reads a signed 32-bit integer from this stream. This method reads 4 bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where `0 <= b1, b2, b3, b4 <= 255`, then the result is equal to:

```
     (b1 << 24) | (b2 << 16) + (b3 << 8) + b4
```

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.
Specified by:
readInt in interface java.io.DataInput
Returns:
the next four bytes of this stream, interpreted as an `int`.
Throws:
java.io.EOFException - if this stream reaches the end before reading four bytes.
java.io.IOException - if an I/O error occurs.

---

readIntLE

```
public final int readIntLE()
                    throws java.io.IOException
```

Reads a signed 32-bit integer from this stream in little-endian order. This method reads 4 bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where `0 <= b1, b2, b3, b4 <= 255`, then the result is equal to:

```
     (b4 << 24) | (b3 << 16) + (b2 << 8) + b1
```

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.
Returns:
the next four bytes of this stream, interpreted as an `int`.
Throws:
java.io.EOFException - if this stream reaches the end before reading four bytes.
java.io.IOException - if an I/O error occurs.

---

readUnsignedInt

```
public final long readUnsignedInt()
                           throws java.io.IOException
```

Reads an unsigned 32-bit integer from this stream. This method reads 4 bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where `0 <= b1, b2, b3, b4 <= 255`, then the result is equal to:

```
     (b1 << 24) | (b2 << 16) + (b3 << 8) + b4
```

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.
Returns:
the next four bytes of this stream, interpreted as a `long`.
Throws:
java.io.EOFException - if this stream reaches the end before reading four bytes.
java.io.IOException - if an I/O error occurs.

readUnsignedIntLE

```
public final long readUnsignedIntLE()
                              throws java.io.IOException
```

Reads an unsigned 32-bit integer from this stream in little-endian order. This method reads 4 bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where `0 <= b1, b2, b3, b4 <= 255`, then the result is equal to:

```
(b4 << 24) | (b3 << 16) + (b2 << 8) + b1
```

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:
the next four bytes of this stream, interpreted as a `long`.

Throws:
java.io.EOFException - if this stream reaches the end before reading four bytes.
java.io.IOException - if an I/O error occurs.

readLong

```
public final long readLong()
                     throws java.io.IOException
```

Reads a signed 64-bit integer from this stream. This method reads eight bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `b7`, and `b8,` where:

```
0 <= b1, b2, b3, b4, b5, b6, b7, b8 <=255,
```

then the result is equal to:

```
((long)b1 << 56) + ((long)b2 << 48)
+ ((long)b3 << 40) + ((long)b4 << 32)
+ ((long)b5 << 24) + ((long)b6 << 16)
+ ((long)b7 << 8) + b8
```

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
readLong in interface java.io.DataInput

Returns:
the next eight bytes of this stream, interpreted as a `long`.

Throws:
java.io.EOFException - if this stream reaches the end before reading eight bytes.
java.io.IOException - if an I/O error occurs.

readLongLE

```
public final long readLongLE()
                       throws java.io.IOException
```

Reads a signed 64-bit integer from this stream in little-endian order. This method reads eight bytes from the stream, starting at the current stream pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `b7`, and `b8,` where:

```
0 <= b1, b2, b3, b4, b5, b6, b7, b8 <=255,
```

then the result is equal to:

```
((long)b1 << 56) + ((long)b2 << 48)
+ ((long)b3 << 40) + ((long)b4 << 32)
+ ((long)b5 << 24) + ((long)b6 << 16)
+ ((long)b7 << 8) + b8
```

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:
the next eight bytes of this stream, interpreted as a `long`.

Throws:
java.io.EOFException - if this stream reaches the end before reading eight bytes.
java.io.IOException - if an I/O error occurs.

readFloat

```
public final float readFloat()
                      throws java.io.IOException
```

Reads a `float` from this stream. This method reads an `int` value, starting at the current stream pointer, as if by the `readInt` method and then converts that `int` to a `float` using the `intBitsToFloat` method in class `Float`.

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
: readFloat in interface java.io.DataInput

Returns:
: the next four bytes of this stream, interpreted as a `float`.

Throws:
: java.io.EOFException - if this stream reaches the end before reading four bytes.
: java.io.IOException - if an I/O error occurs.

readFloatLE

```
public final float readFloatLE()
                        throws java.io.IOException
```

Reads a `float` from this stream in little-endian order. This method reads an `int` value, starting at the current stream pointer, as if by the `readInt` method and then converts that `int` to a `float` using the `intBitsToFloat` method in class `Float`.

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:
: the next four bytes of this stream, interpreted as a `float`.

Throws:
: java.io.EOFException - if this stream reaches the end before reading four bytes.
: java.io.IOException - if an I/O error occurs.

readDouble

```
public final double readDouble()
                        throws java.io.IOException
```

Reads a `double` from this stream. This method reads a `long` value, starting at the current stream pointer, as if by the `readLong` method and then converts that `long` to a `double` using the `longBitsToDouble` method in class `Double`.

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
: readDouble in interface java.io.DataInput

Returns:
: the next eight bytes of this stream, interpreted as a `double`.

Throws:
: java.io.EOFException - if this stream reaches the end before reading eight bytes.
: java.io.IOException - if an I/O error occurs.

readDoubleLE

```
public final double readDoubleLE()
                          throws java.io.IOException
```

Reads a `double` from this stream in little-endian order. This method reads a `long` value, starting at the current stream pointer, as if by the `readLong` method and then converts that `long` to a `double` using the `longBitsToDouble` method in class `Double`.

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:
: the next eight bytes of this stream, interpreted as a `double`.

Throws:
: java.io.EOFException - if this stream reaches the end before reading eight bytes.
: java.io.IOException - if an I/O error occurs.

readLine

```
public final java.lang.String readLine()
                                throws java.io.IOException
```

Reads the next line of text from this stream. This method successively reads bytes from the stream, starting at the current stream pointer, until it reaches a line terminator or the end of the stream. Each byte is converted into a character by taking the byte's value for the lower eight bits of the character and setting the high eight bits of the character to zero. This method does not, therefore, support the full Unicode character set.

A line of text is terminated by a carriage-return character (`'\r'`), a newline character (`'\n'`), a carriage-return character immediately followed by a newline character, or the end of the stream. Line-terminating characters are discarded and are not included as part of the string returned.

This method blocks until a newline character is read, a carriage return and the byte following it are read (to see if it is a newline), the end of the stream is reached, or an exception is thrown.

Specified by:
readLine in interface java.io.DataInput
Returns:
the next line of text from this stream, or null if end of stream is encountered before even one byte is read.
Throws:
java.io.IOException - if an I/O error occurs.

---

readUTF

```
public final java.lang.String readUTF()
                               throws java.io.IOException
```

Reads in a string from this stream. The string has been encoded using a modified UTF-8 format.

The first two bytes are read, starting from the current stream pointer, as if by `readUnsignedShort`. This value gives the number of following bytes that are in the encoded string, not the length of the resulting string. The following bytes are then interpreted as bytes encoding characters in the UTF-8 format and are converted into characters.

This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

Specified by:
readUTF in interface java.io.DataInput
Returns:
a Unicode string.
Throws:
java.io.EOFException - if this stream reaches the end before reading all the bytes.
java.io.IOException - if an I/O error occurs.
UTFDataFormatException - if the bytes do not represent valid UTF-8 encoding of a Unicode string.

---

finalize

```
protected void finalize()
                 throws java.lang.Throwable
```

Releases any system resources associated with this stream by calling the `close()` method.

Overrides:
finalize in class java.lang.Object

com.sun.media.jai.codec

Class SegmentedSeekableStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--com.sun.media.jai.codec.SeekableStream
              |
              +--com.sun.media.jai.codec.SegmentedSeekableStream
```

public class **SegmentedSeekableStream**
extends SeekableStream

A `SegmentedSeekableStream` provides a view of a subset of another `SeekableStream` consiting of a series of segments with given starting positions in the source stream and lengths. The resulting stream behaves like an ordinary `SeekableStream`.

For example, given a `SeekableStream` containing data in a format consisting of a number of sub-streams stored in non-contiguous sectors indexed by a directory, it is possible to construct a set of `SegmentedSeekableStream`s, one for each sub-stream, that each provide a view of the sectors comprising a particular stream by providing the positions and lengths of the stream's sectors as indicated by the directory. The complex multi-stream structure of the original stream may be ignored by users of the `SegmentedSeekableStream`, who see a separate `SeekableStream` for each sub-stream and do not need to understand the directory structure at all.

For further efficiency, a directory structure such as in the example described above need not be fully parsed in order to build a `SegmentedSeekableStream`. Instead, the `StreamSegmentMapper` interface allows the association between a desired region of the output and an input segment to be provided dynamically. This mapping might be computed by reading from a directory in piecemeal fashion in order to avoid consuming memory resources.

It is the responsibility of the user of this class to determine whether backwards seeking should be enabled. If the source stream supports only forward seeking, backwards seeking must be disabled and the `StreamSegmentMapper` must be monotone; that is, forward motion in the destination must always result in forward motion within the source. If the source stream supports backwards seeking, there are no restrictions on the `StreamSegmentMapper` and backwards seeking may always be enabled for the `SegmentedSeekableStream`.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

stream

`private SeekableStream` **`stream`**

mapper

`private StreamSegmentMapper` **`mapper`**

pointer

`private long` **`pointer`**

canSeekBackwards

`private boolean` **`canSeekBackwards`**

streamSegment

`private StreamSegment` **`streamSegment`**

Constructor Detail

SegmentedSeekableStream

```
public SegmentedSeekableStream(SeekableStream stream,
                               StreamSegmentMapper mapper,
                               boolean canSeekBackwards)
```

Constructs a `SegmentedSeekableStream` given a `SeekableStream` as input, an instance of `StreamSegmentMapper`, and a `boolean` indicating whether the output `SegmentedSeekableStream` should support seeking backwards. If `canSeekBackwards` is `true`, the source stream must itself support seeking backwards.

Parameters:

`stream` - A source `SeekableStream`
`mapper` - An instance of the `StreamSegmentMapper` interface.
`canSeekBackwards` - `true` if the ability to seek backwards is desired.

SegmentedSeekableStream

```
public SegmentedSeekableStream(SeekableStream stream,
                               long[] segmentPositions,
                               int[] segmentLengths,
                               boolean canSeekBackwards)
```

Constructs a `SegmentedSeekableStream` given a `SeekableStream` as input, a list of the starting positions and lengths of the segments of the source stream, and a `boolean` indicating whether the output `SegmentedSeekableStream` should support seeking backwards. If `canSeekBakckwards` is `true`, the source stream must itself support seeking backwards.

Parameters:

`stream` - A source `SeekableStream`
`segmentPositions` - An array of `long`s giving the starting positions of the segments in the source stream.
`segmentLengths` - An array of `int`s giving the lengths of segments in the source stream.
`canSeekBackwards` - `true` if the ability to seek backwards is desired.

SegmentedSeekableStream

```
public SegmentedSeekableStream(SeekableStream stream,
                               long[] segmentPositions,
                               int segmentLength,
                               int totalLength,
                               boolean canSeekBackwards)
```

Constructs a `SegmentedSeekableStream` given a `SeekableStream` as input, a list of the starting positions of the segments of the source stream, the common length of each segment, the total length of the segments and a `boolean` indicating whether the output `SegmentedSeekableStream` should support seeking backwards. If `canSeekBakckwards` is `true`, the source stream must itself support seeking backwards.

This constructor is useful for selecting substreams of sector-oriented file formats in which each segment of the substream (except possibly the final segment) occupies a fixed-length sector.

Parameters:

`stream` - A source `SeekableStream`
`segmentPositions` - An array of `long`s giving the starting positions of the segments in the source stream.
`segmentLength` - The common length of each segment.
`totalLength` - The total length of the source segments.
`canSeekBackwards` - `true` if the ability to seek backwards is desired.

Method Detail

getFilePointer

```
public long getFilePointer()
```

Returns the current offset in this stream.

Returns:

the offset from the beginning of the stream, in bytes, at which the next read occurs.

Overrides:

getFilePointer in class SeekableStream

---

canSeekBackwards

```
public boolean canSeekBackwards()
```

Returns `true` if seeking backwards is supported. Support is determined by the value of the `canSeekBackwards` parameter at construction time.

Overrides:

canSeekBackwards in class SeekableStream

---

seek

```
public void seek(long pos)
          throws java.io.IOException
```

Sets the offset, measured from the beginning of this stream, at which the next read occurs.

If `canSeekBackwards()` returns `false`, then setting `pos` to an offset smaller than the current value of `getFilePointer()` will have no effect.

Parameters:

`pos` - the offset position, measured in bytes from the beginning of the stream, at which to set the stream pointer.

Throws:

java.io.IOException - if `pos` is less than `0` or if an I/O error occurs.

Overrides:

seek in class SeekableStream

---

read

```
public int read()
         throws java.io.IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of data, or `-1` if the end of the stream is reached.

Throws:

java.io.IOException - if an I/O error occurs.

Overrides:

read in class SeekableStream

---

read

```
public int read(byte[] b,
                int off,
                int len)
         throws java.io.IOException
```

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of stream is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of stream, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of stream, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

Parameters:

`b` - the buffer into which the data is read.
`off` - the start offset in array `b` at which the data is written.
`len` - the maximum number of bytes to read.

Returns:

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:
java.io.IOException - if an I/O error occurs.
Overrides:
read in class SeekableStream

com.sun.media.jai.codec

Class StreamSegment

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.StreamSegment
```

public class **StreamSegment**
extends java.lang.Object

A utility class representing a segment within a stream as a `long` starting position and an `int` length.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

startPos

`private long` **`startPos`**

segmentLength

`private int` **`segmentLength`**

Constructor Detail

StreamSegment

`public` **`StreamSegment`**`()`

Constructs a `StreamSegment`. The starting position and length are set to 0.

StreamSegment

```
public StreamSegment(long startPos,
                     int segmentLength)
```

Constructs a `StreamSegment` with a given starting position and length.

Method Detail

getStartPos

`public final long` **`getStartPos`**`()`

Returns the starting position of the segment.

setStartPos

`public final void` **`setStartPos`**`(long startPos)`

Sets the starting position of the segment.

getSegmentLength

`public final int` **`getSegmentLength`**`()`

Returns the length of the segment.

setSegmentLength

`public final void` **`setSegmentLength`**`(int segmentLength)`

Sets the length of the segment.

com.sun.media.jai.codec

Interface StreamSegmentMapper

All Known Implementing Classes:
SectorStreamSegmentMapper, StreamSegmentMapperImpl

---

public abstract interface **StreamSegmentMapper**

An interface for use with the `SegmentedSeekableStream` class. An instance of the `StreamSegmentMapper` interface provides the location and length of a segment of a source `SeekableStream` corresponding to the initial portion of a desired segment of the output stream.

As an example, consider a mapping between a source `SeekableStream src` and a `SegmentedSeekableStream dst` comprising bytes 100-149 and 200-249 of the source stream. The `dst` stream has a reference to an instance `mapper` of `StreamSegmentMapper`.

A call to `dst.seek(0); dst.read(buf, 0, 10)` will result in a call to `mapper.getStreamSegment(0, 10)`, returning a new `StreamSegment` with a starting position of 100 and a length of 10 (or less). This indicates that in order to read bytes 0-9 of the segmented stream, bytes 100-109 of the source stream should be read.

A call to `dst.seek(10); int nbytes = dst.read(buf, 0, 100)` is somewhat more complex, since it will require data from both segments of `src`. The method `mapper.getStreamSegment(10, 100)` will be called. This method will return a new `StreamSegment` with a starting position of 110 and a length of 40 (or less). The length is limited to 40 since a longer value would result in a read past the end of the first segment. The read will stop after the first 40 bytes and an addition read or reads will be required to obtain the data contained in the second segment.

This interface is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Method Detail

getStreamSegment

```
public StreamSegment getStreamSegment(long pos,
                                      int length)
```

Returns a `StreamSegment` object indicating the location of the initial portion of a desired segment in the source stream. The length of the returned `StreamSegment` may be smaller than the desired length.

Parameters:
`pos` - The desired starting position in the `SegmentedSeekableStream`, as a `long`.
`length` - The desired segment length.

---

getStreamSegment

```
public void getStreamSegment(long pos,
                             int length,
                             StreamSegment seg)
```

Sets the values of a `StreamSegment` object indicating the location of the initial portion of a desired segment in the source stream. The length of the returned `StreamSegment` may be smaller than the desired length.

Parameters:
`pos` - The desired starting position in the `SegmentedSeekableStream`, as a `long`.
`length` - The desired segment length.
`seg` - A `StreamSegment` object to be overwritten.

com.sun.media.jai.codec

Class StreamSegmentMapperImpl

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.StreamSegmentMapperImpl
```

class **StreamSegmentMapperImpl**
extends java.lang.Object
implements StreamSegmentMapper

An implementation of the `StreamSegmentMapper` interface that requires an explicit list of the starting locations and lengths of the source segments.

Field Detail

segmentPositions

```
private long[] segmentPositions
```

segmentLengths

```
private int[] segmentLengths
```

Constructor Detail

StreamSegmentMapperImpl

```
public StreamSegmentMapperImpl(long[] segmentPositions,
                               int[] segmentLengths)
```

Method Detail

getStreamSegment

```
public StreamSegment getStreamSegment(long position,
                                      int length)
```

Specified by:
getStreamSegment in interface StreamSegmentMapper

getStreamSegment

```
public void getStreamSegment(long position,
                             int length,
                             StreamSegment seg)
```

Specified by:
getStreamSegment in interface StreamSegmentMapper

com.sun.media.jai.codec

Class TIFFDecodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.TIFFDecodeParam
```

public class **TIFFDecodeParam**
extends java.lang.Object
implements ImageDecodeParam

An instance of `ImageDecodeParam` for decoding images in the TIFF format.

To determine the number of images present in a TIFF file, use the `getNumPages()` method on the `ImageDecoder` object that will be used to perform the decoding. The desired page number may be passed as an argument to the `ImageDecoder.decodeAsRaster)()` or `decodeAsRenderedImage()` methods.

For TIFF Palette color images, the colorMap always has entries of short data type, the color Black being represented by 0,0,0 and White by 65536,65536,65536. In order to display these images, the default behavior is to dither the short values down to 8 bits. The dithering is done by calling the `decode16BitsTo8Bits` method for each short value that needs to be dithered. The method has the following implementation: `byte b; short s; s = s & 0xffff; b = (byte)((s >> 8) & 0xff);` If a different algorithm is to be used for the dithering, this class should be subclassed and an appropriate implementation should be provided for the `decode16BitsTo8Bits` method in the subclass.

If the palette contains image data that is signed short, as specified by the SampleFormat tag, the dithering is done by calling `decodeSigned16BitsTo8Bits` instead. The method has the following implementation: `byte b; short s; b = (byte)((s + Short.MIN_VALUE) >> 8);` In order to use a different algorithm for the dithering, this class should be subclassed and the method overridden.

If it is desired that the Palette be decoded such that the output image is of short data type and no dithering is performed, the `setDecodePaletteAsShorts` method should be used.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

See Also:
`TIFFDirectory`

Field Detail

decodePaletteAsShorts

`private boolean` **`decodePaletteAsShorts`**

Constructor Detail

TIFFDecodeParam

`public` **`TIFFDecodeParam`**`()`

Constructs a default instance of `TIFFDecodeParam`.

Method Detail

setDecodePaletteAsShorts

`public void` **`setDecodePaletteAsShorts`**`(boolean decodePaletteAsShorts)`

If set, the entries in the palette will be decoded as shorts and no short to byte lookup will be applied to them.

getDecodePaletteAsShorts

`public boolean` **`getDecodePaletteAsShorts`**`()`

Returns `true` if palette entries will be decoded as shorts, resulting in an output image with short datatype.

decode16BitsTo8Bits

`public byte` **`decode16BitsTo8Bits`**`(int s)`

Returns an unsigned 8 bit value computed by dithering the unsigned 16 bit value. Note that the TIFF specified short datatype is an unsigned value, while Java's `short` datatype is a signed value. Therefore the Java `short` datatype cannot be used to store the TIFF specified short value. A Java `int` is used as input instead to this method. The method deals correctly only with 16 bit unsigned values.

---

decodeSigned16BitsTo8Bits

`public byte` **`decodeSigned16BitsTo8Bits`**`(short s)`

Returns an unsigned 8 bit value computed by dithering the signed 16 bit value. This method deals correctly only with values in the 16 bit signed range.

com.sun.media.jai.codec

Class TIFFDirectory

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.TIFFDirectory
```

public class **TIFFDirectory**
extends java.lang.Object

A class representing an Image File Directory (IFD) from a TIFF 6.0 stream. The TIFF file format is described in more detail in the comments for the TIFFDescriptor class.

A TIFF IFD consists of a set of TIFFField tags. Methods are provided to query the set of tags and to obtain the raw field array. In addition, convenience methods are provided for acquiring the values of tags that contain a single value that fits into a byte, int, long, float, or double.

Every TIFF file is made up of one or more public IFDs that are joined in a linked list, rooted in the file header. A file may also contain so-called private IFDs that are referenced from tag data and do not appear in the main list.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

See Also:
`TIFFDescriptor`, `TIFFField`

Field Detail

stream

`SeekableStream` **`stream`**

The stream being read.

isBigEndian

`boolean` **`isBigEndian`**

A boolean storing the endianness of the stream.

numEntries

`int` **`numEntries`**

The number of entries in the IFD.

fields

`TIFFField[]` **`fields`**

An array of TIFFFields.

fieldIndex

`java.util.Hashtable` **`fieldIndex`**

A Hashtable indexing the fields by tag number.

sizeOfType

`private static final int[]` **`sizeOfType`**

Constructor Detail

TIFFDirectory

```
TIFFDirectory()
```

The default constructor.

TIFFDirectory

```
public TIFFDirectory(SeekableStream stream,
                     int directory)
              throws java.io.IOException
```

Constructs a TIFFDirectory from a SeekableStream. The directory parameter specifies which directory to read from the linked list present in the stream; directory 0 is normally read but it is possible to store multiple images in a single TIFF file by maintaing multiple directories.

Parameters:

`stream` - a SeekableStream to read from.
`directory` - the index of the directory to read.

TIFFDirectory

```
public TIFFDirectory(SeekableStream stream,
                     long ifd_offset)
              throws java.io.IOException
```

Constructs a TIFFDirectory by reading a SeekableStream. The ifd_offset parameter specifies the stream offset from which to begin reading; this mechanism is sometimes used to store private IFDs within a TIFF file that are not part of the normal sequence of IFDs.

Parameters:

`stream` - a SeekableStream to read from.
`ifd_offset` - the long byte offset of the directory.

Method Detail

isValidEndianTag

```
private static boolean isValidEndianTag(int endian)
```

---

initialize

```
private void initialize()
                 throws java.io.IOException
```

---

getNumEntries

```
public int getNumEntries()
```

Returns the number of directory entries.

---

getField

```
public TIFFField getField(int tag)
```

Returns the value of a given tag as a TIFFField, or null if the tag is not present.

---

isTagPresent

```
public boolean isTagPresent(int tag)
```

Returns true if a tag appears in the directory.

---

getTags

```
public int[] getTags()
```

Returns an ordered array of ints indicating the tag values.

---

getFields

```
public TIFFField[] getFields()
```

Returns an array of TIFFFields containing all the fields in this directory.

---

getFieldAsByte

```
public byte getFieldAsByte(int tag,
                           int index)
```

Returns the value of a particular index of a given tag as a byte. The caller is responsible for ensuring that the tag is present and has type TIFFField.TIFF_SBYTE, TIFF_BYTE, or TIFF_UNDEFINED.

---

getFieldAsByte

```
public byte getFieldAsByte(int tag)
```

Returns the value of index 0 of a given tag as a byte. The caller is responsible for ensuring that the tag is present and has type TIFFField.TIFF_SBYTE, TIFF_BYTE, or TIFF_UNDEFINED.

---

getFieldAsLong

```
public long getFieldAsLong(int tag,
                           int index)
```

Returns the value of a particular index of a given tag as a long. The caller is responsible for ensuring that the tag is present and has type TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, TIFF_SLONG or TIFF_LONG.

---

getFieldAsLong

```
public long getFieldAsLong(int tag)
```

Returns the value of index 0 of a given tag as a long. The caller is responsible for ensuring that the tag is present and has type TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, TIFF_SLONG or TIFF_LONG.

---

getFieldAsFloat

```
public float getFieldAsFloat(int tag,
                             int index)
```

Returns the value of a particular index of a given tag as a float. The caller is responsible for ensuring that the tag is present and has numeric type (all but TIFF_UNDEFINED and TIFF_ASCII).

---

getFieldAsFloat

```
public float getFieldAsFloat(int tag)
```

Returns the value of index 0 of a given tag as a float. The caller is responsible for ensuring that the tag is present and has numeric type (all but TIFF_UNDEFINED and TIFF_ASCII).

---

getFieldAsDouble

```
public double getFieldAsDouble(int tag,
                               int index)
```

Returns the value of a particular index of a given tag as a double. The caller is responsible for ensuring that the tag is present and has numeric type (all but TIFF_UNDEFINED and TIFF_ASCII).

---

getFieldAsDouble

```
public double getFieldAsDouble(int tag)
```

Returns the value of index 0 of a given tag as a double. The caller is responsible for ensuring that the tag is present and has numeric type (all but TIFF_UNDEFINED and TIFF_ASCII).

---

readShort

```
private short readShort(SeekableStream stream)
                 throws java.io.IOException
```

---

readUnsignedShort

```
private int readUnsignedShort(SeekableStream stream)
                       throws java.io.IOException
```

---

readInt

```
private int readInt(SeekableStream stream)
             throws java.io.IOException
```

---

readUnsignedInt

```
private long readUnsignedInt(SeekableStream stream)
                      throws java.io.IOException
```

---

readLong

```
private long readLong(SeekableStream stream)
               throws java.io.IOException
```

---

readFloat

```
private float readFloat(SeekableStream stream)
                 throws java.io.IOException
```

---

readDouble

```
private double readDouble(SeekableStream stream)
                   throws java.io.IOException
```

---

readUnsignedShort

```
private static int readUnsignedShort(SeekableStream stream,
                                     boolean isBigEndian)
                              throws java.io.IOException
```

---

readUnsignedInt

```
private static long readUnsignedInt(SeekableStream stream,
                                    boolean isBigEndian)
                             throws java.io.IOException
```

---

getNumDirectories

```
public static int getNumDirectories(SeekableStream stream)
                             throws java.io.IOException
```

Returns the number of image directories (subimages) stored in a given TIFF file, represented by a `SeekableStream`.

---

isBigEndian

```
public boolean isBigEndian()
```

Returns a boolean indicating whether the byte order used in the the TIFF file is big-endian (i.e. whether the byte order is from the most significant to the least significant)

com.sun.media.jai.codec
Class TIFFEncodeParam

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.TIFFEncodeParam
```

public class **TIFFEncodeParam**
extends java.lang.Object
implements ImageEncodeParam

An instance of `ImageEncodeParam` for encoding images in the TIFF format.

This class allows for the specification of encoding parameters. By default, the image is encoded without any compression, and is written out consisting of strips, not tiles. The particular compression scheme to be used can be specified by using the `setCompression` method. The compression scheme specified will be honored only if it is compatible with the type of image being written out. For example, Group3 and Group4 compressions can only be used with Bilevel images. Writing of tiled TIFF images can be enabled by calling the `setWriteTiled` method.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

Field Detail

COMPRESSION_NONE
`public static final int` **`COMPRESSION_NONE`**

COMPRESSION_PACKBITS
`public static final int` **`COMPRESSION_PACKBITS`**

COMPRESSION_GROUP3_1D
`public static final int` **`COMPRESSION_GROUP3_1D`**

COMPRESSION_GROUP3_2D
`public static final int` **`COMPRESSION_GROUP3_2D`**

COMPRESSION_GROUP4
`public static final int` **`COMPRESSION_GROUP4`**

COMPRESSION_LZW
`public static final int` **`COMPRESSION_LZW`**

compression
`private int` **`compression`**

writeTiled
`private boolean` **`writeTiled`**

Constructor Detail

TIFFEncodeParam

`public TIFFEncodeParam()`

Constructs an TIFFEncodeParam object with default values for parameters.

Method Detail

getCompression

`public int getCompression()`

Returns the value of the compression parameter.

---

setCompression

`public void setCompression(int compression)`

Specifies the type of compression to be used. The compression type specified will be honored only if it is compatible with the image being written out.

Parameters:

`compression` - The compression type.

---

getWriteTiled

`public boolean getWriteTiled()`

Returns the value of the writeTiled parameter.

---

setWriteTiled

`public void setWriteTiled(boolean writeTiled)`

If set, the data will be written out in tiled format, instead of in strips.

Parameters:

`writeTiled` - Specifies whether the image data should be wriiten out in tiled format.

com.sun.media.jai.codec

Class TIFFField

```
java.lang.Object
  |
  +--com.sun.media.jai.codec.TIFFField
```

public class **TIFFField**
extends java.lang.Object

A class representing a field in a TIFF 6.0 Image File Directory.

The TIFF file format is described in more detail in the comments for the TIFFDescriptor class.

A field in a TIFF Image File Directory (IFD). A field is defined as a sequence of values of identical data type. TIFF 6.0 defines 12 data types, which are mapped internally onto the Java datatypes byte, int, long, float, and double.

This class is not a committed part of the JAI API. It may be removed or changed in future releases of JAI.

See Also:
`TIFFDescriptor`, `TIFFDirectory`

Field Detail

TIFF_BYTE

`public static final int` **`TIFF_BYTE`**

Flag for 8 bit unsigned integers.

TIFF_ASCII

`public static final int` **`TIFF_ASCII`**

Flag for null-terminated ASCII strings.

TIFF_SHORT

`public static final int` **`TIFF_SHORT`**

Flag for 16 bit unsigned integers.

TIFF_LONG

`public static final int` **`TIFF_LONG`**

Flag for 32 bit unsigned integers.

TIFF_RATIONAL

`public static final int` **`TIFF_RATIONAL`**

Flag for pairs of 32 bit unsigned integers.

TIFF_SBYTE

`public static final int` **`TIFF_SBYTE`**

Flag for 8 bit signed integers.

TIFF_UNDEFINED

`public static final int` **`TIFF_UNDEFINED`**

Flag for 8 bit uninterpreted bytes.

TIFF_SSHORT

`public static final int **TIFF_SSHORT**`

Flag for 16 bit signed integers.

TIFF_SLONG

`public static final int **TIFF_SLONG**`

Flag for 32 bit signed integers.

TIFF_SRATIONAL

`public static final int **TIFF_SRATIONAL**`

Flag for pairs of 32 bit signed integers.

TIFF_FLOAT

`public static final int **TIFF_FLOAT**`

Flag for 32 bit IEEE floats.

TIFF_DOUBLE

`public static final int **TIFF_DOUBLE**`

Flag for 64 bit IEEE doubles.

---

tag

`int **tag**`

The tag number.

---

type

`int **type**`

The tag type.

---

count

`int **count**`

The number of data items present in the field.

---

data

`java.lang.Object **data**`

The field data.

Constructor Detail

TIFFField

`**TIFFField**()`

The default constructor.

TIFFField

```
public TIFFField(int tag,
                 int type,
                 int count,
                 java.lang.Object data)
```

Constructs a TIFFField with arbitrary data. The data parameter must be an array of a Java type appropriate for the type of the TIFF field. Since there is no available 32-bit unsigned datatype, long is used. The mapping between types is as follows:

TIFF type	Java type
`TIFF_BYTE`	`byte`
`TIFF_ASCII`	`String`
`TIFF_SHORT`	`char`
`TIFF_LONG`	`long`
`TIFF_RATIONAL`	`long[2]`
`TIFF_SBYTE`	`byte`
`TIFF_UNDEFINED`	`byte`
`TIFF_SSHORT`	`short`
`TIFF_SLONG`	`int`
`TIFF_SRATIONAL`	`int[2]`
`TIFF_FLOAT`	`float`
`TIFF_DOUBLE`	`double`

Method Detail

getTag

`public int getTag()`

Returns the tag number, between 0 and 65535.

---

getType

`public int getType()`

Returns the type of the data stored in the IFD. For a TIFF6.0 file, the value will equal one of the TIFF_ constants defined in this class. For future revisions of TIFF, higher values are possible.

---

getCount

`public int getCount()`

Returns the number of elements in the IFD.

---

getAsBytes

`public byte[] getAsBytes()`

Returns the data as an uninterpreted array of bytes. The type of the field must be one of TIFF_BYTE, TIFF_SBYTE, or TIFF_UNDEFINED;

For data in TIFF_BYTE format, the application must take care when promoting the data to longer integral types to avoid sign extension.

A ClassCastException will be thrown if the field is not of type TIFF_BYTE, TIFF_SBYTE, or TIFF_UNDEFINED.

---

getAsChars

`public char[] **getAsChars**()`

Returns TIFF_SHORT data as an array of chars (unsigned 16-bit integers).

A ClassCastException will be thrown if the field is not of type TIFF_SHORT.

---

getAsShorts

`public short[] **getAsShorts**()`

Returns TIFF_SSHORT data as an array of shorts (signed 16-bit integers).

A ClassCastException will be thrown if the field is not of type TIFF_SSHORT.

---

getAsInts

`public int[] **getAsInts**()`

Returns TIFF_SLONG data as an array of ints (signed 32-bit integers).

A ClassCastException will be thrown if the field is not of type TIFF_SLONG.

---

getAsLongs

`public long[] **getAsLongs**()`

Returns TIFF_LONG data as an array of longs (signed 64-bit integers).

A ClassCastException will be thrown if the field is not of type TIFF_LONG.

---

getAsFloats

`public float[] **getAsFloats**()`

Returns TIFF_FLOAT data as an array of floats.

A ClassCastException will be thrown if the field is not of type TIFF_FLOAT.

---

getAsDoubles

`public double[] **getAsDoubles**()`

Returns TIFF_DOUBLE data as an array of doubles.

A ClassCastException will be thrown if the field is not of type TIFF_DOUBLE.

---

getAsSRationals

`public int[][] **getAsSRationals**()`

Returns TIFF_SRATIONAL data as an array of 2-element arrays of ints.

A ClassCastException will be thrown if the field is not of type TIFF_SRATIONAL.

---

getAsRationals

`public long[][] **getAsRationals**()`

Returns TIFF_RATIONAL data as an array of 2-element arrays of longs.

A ClassCastException will be thrown if the field is not of type TIFF_RATTIONAL.

---

getAsInt

`public int **getAsInt**(int index)`

Returns data in TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, or TIFF_SLONG format as an int.

TIFF_BYTE and TIFF_UNDEFINED data are treated as unsigned; that is, no sign extension will take place and the returned value will be in the range [0, 255]. TIFF_SBYTE data will be returned in the range [-128, 127].

A ClassCastException will be thrown if the field is not of type TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, or TIFF_SLONG.

getAsLong

`public long **getAsLong**(int index)`

Returns data in TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, TIFF_SLONG, or TIFF_LONG format as a long.

TIFF_BYTE and TIFF_UNDEFINED data are treated as unsigned; that is, no sign extension will take place and the returned value will be in the range [0, 255]. TIFF_SBYTE data will be returned in the range [-128, 127].

A ClassCastException will be thrown if the field is not of type TIFF_BYTE, TIFF_SBYTE, TIFF_UNDEFINED, TIFF_SHORT, TIFF_SSHORT, TIFF_SLONG, or TIFF_LONG.

getAsFloat

`public float **getAsFloat**(int index)`

Returns data in any numerical format as a float. Data in TIFF_SRATIONAL or TIFF_RATIONAL format are evaluated by dividing the numerator into the denominator using double-precision arithmetic and then truncating to single precision. Data in TIFF_SLONG, TIFF_LONG, or TIFF_DOUBLE format may suffer from truncation.

A ClassCastException will be thrown if the field is of type TIFF_UNDEFINED or TIFF_ASCII.

getAsDouble

`public double **getAsDouble**(int index)`

Returns data in any numerical format as a float. Data in TIFF_SRATIONAL or TIFF_RATIONAL format are evaluated by dividing the numerator into the denominator using double-precision arithmetic.

A ClassCastException will be thrown if the field is of type TIFF_UNDEFINED or TIFF_ASCII.

getAsString

`public java.lang.String **getAsString**(int index)`

Returns a TIFF_ASCII data item as a String.

A ClassCastException will be thrown if the field is not of type TIFF_ASCII.

getAsSRational

`public int[] **getAsSRational**(int index)`

Returns a TIFF_SRATIONAL data item as a two-element array of ints.

A ClassCastException will be thrown if the field is not of type TIFF_SRATIONAL.

getAsRational

`public long[] **getAsRational**(int index)`

Returns a TIFF_RATIONAL data item as a two-element array of ints.

A ClassCastException will be thrown if the field is not of type TIFF_RATIONAL.

DataBufferDouble - class javax.media.jai.DataBufferDouble.
An extension of `DataBuffer` that stores data internally in `double` form.
DataBufferDouble(double[][], int) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with the specified data arrays.
DataBufferDouble(double[][], int, int[]) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with the specified data arrays, size, and per-bank offsets.
DataBufferDouble(double[], int) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with the specified data array.
DataBufferDouble(double[], int, int) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with the specified data array.
DataBufferDouble(int) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with a specified size.
DataBufferDouble(int, int) - Constructor for class javax.media.jai.DataBufferDouble
Constructs a `double`-based `DataBuffer` with a specified number of banks, all of which are of a specified size.
DataBufferFloat - class javax.media.jai.DataBufferFloat.
An extension of DataBuffer that stores data internally in `float` form.
DataBufferFloat(float[][], int) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with the specified data arrays.
DataBufferFloat(float[][], int, int[]) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with the specified data arrays, size, and per-bank offsets.
DataBufferFloat(float[], int) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with the specified data array.
DataBufferFloat(float[], int, int) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with the specified data array.
DataBufferFloat(int) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with a specified size.
DataBufferFloat(int, int) - Constructor for class javax.media.jai.DataBufferFloat
Constructs a `float`-based `DataBuffer` with a specified number of banks, all of which are of a specified size.
dataH - Variable in class javax.media.jai.KernelJAI
The horizontal data for a separable kernel
dataHd - Variable in class javax.media.jai.InterpolationTable
The horizontal coefficient data in double format.
dataHelper(int) - Static method in class javax.media.jai.InterpolationBicubic2

dataHelper(int) - Static method in class javax.media.jai.InterpolationBicubic

dataHf - Variable in class javax.media.jai.InterpolationTable
The horizontal coefficient data in floating-point format.
dataHi - Variable in class javax.media.jai.InterpolationTable
The horizontal coefficient data in fixed-point format.
dataType - Variable in class javax.media.jai.ColorCube
The data type cached to accelerate findNearestEntry().
DATATYPE_MASK - Static variable in class javax.media.jai.RasterAccessor
The bits of a FormatTagID associated with pixel datatype.
dataV - Variable in class javax.media.jai.KernelJAI
The vertical data for a separable kernel
dataVd - Variable in class javax.media.jai.InterpolationTable
The vertical coefficient data in double format.
dataVf - Variable in class javax.media.jai.InterpolationTable
The vertical coefficient data in floating-point format.
dataVi - Variable in class javax.media.jai.InterpolationTable
The vertical coefficient data in fixed-point format.
DCTDescriptor - class javax.media.jai.operator.DCTDescriptor.
An `OperationDescriptor` describing the "DCT" operation.
DCTDescriptor() - Constructor for class javax.media.jai.operator.DCTDescriptor
Constructor.
decode16BitsTo8Bits(int) - Method in class com.sun.media.jai.codec.TIFFDecodeParam
Returns an unsigned 8 bit value computed by dithering the unsigned 16 bit value.
decodeAsRaster() - Method in class com.sun.media.jai.codec.ImageDecoderImpl
Returns a `Raster` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`.
decodeAsRaster() - Method in interface com.sun.media.jai.codec.ImageDecoder
Returns a Raster that contains the decoded contents of the SeekableStream associated with this ImageDecoder.
decodeAsRaster(int) - Method in class com.sun.media.jai.codec.ImageDecoderImpl
Returns a `Raster` that contains the decoded contents of the `SeekableStream` associated with this `ImageDecoder`.
decodeAsRaster(int) - Method in interface com.sun.media.jai.codec.ImageDecoder
Returns a Raster that contains the decoded contents of the SeekableStream associated with this ImageDecoder.

getOperationName() - Method in class javax.media.jai.RenderedOp
Returns the name of the operation this node represents as a `String`.
getOperationName() - Method in class javax.media.jai.RenderableOp
Returns the name of the operation this node represents as a `String`.
getOperationName() - Method in class javax.media.jai.CollectionOp
Returns the name of the operation this node represents as a `String`.
getOperationNames() - Method in class javax.media.jai.OperationRegistry
Returns a list of names under which all the OperationDescriptors in the registry are registered.
getOperationRegistry() - Method in class javax.media.jai.JAI
Returns the OperationRegistry being used by this JAI instance.
getOrderedCIFList(String, String) - Method in class javax.media.jai.OperationRegistry
Returns a list of the CIFs of a product registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set.
getOrderedOperationList() - Method in class javax.media.jai.ProductOperationGraph
Performs a topological sort on the set of RIFs.
getOrderedOperationList(String) - Method in class javax.media.jai.OperationGraph
Returns an ordered list of the specified imageFactory
getOrderedProductList(String) - Method in class javax.media.jai.OperationRegistry
Returns a list of the products registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set.
getOrderedRIFList(String, String) - Method in class javax.media.jai.OperationRegistry
Returns a list of the RIFs of a product registered under a particular OperationDescriptor, in an ordering that satisfies all of the pairwise preferences that have been set.
getOutput8BitGray() - Method in class com.sun.media.jai.codec.PNGDecodeParam
Returns the current value of the 8-bit gray output parameter.
getOutputStream() - Method in class com.sun.media.jai.codec.ImageEncoderImpl
Returns the OutputStream associated with this ImageEncoder.
getOutputStream() - Method in interface com.sun.media.jai.codec.ImageEncoder
Returns the OutputStream associated with this ImageEncoder.
getPaint() - Method in class javax.media.jai.GraphicsJAI
See comments in java.awt.Graphics2D.
getPaint() - Method in class javax.media.jai.TiledImageGraphics

getPaint() - Method in class javax.media.jai.RenderableGraphics

getPalette() - Method in class com.sun.media.jai.codec.PNGEncodeParam.Palette
Returns the current RGB palette.
getPaletteHistogram() - Method in class com.sun.media.jai.codec.PNGEncodeParam
Returns the palette histogram to be stored with this image.
getPaletteTransparency() - Method in class com.sun.media.jai.codec.PNGEncodeParam.Palette
Returns the alpha values associated with each palette entry.
getParam() - Method in class com.sun.media.jai.codec.ImageDecoderImpl
Returns the current parameters as an instance of the `ImageDecodeParam` interface.
getParam() - Method in class com.sun.media.jai.codec.ImageEncoderImpl
Returns the current parameters as an instance of the ImageEncodeParam interface.
getParam() - Method in interface com.sun.media.jai.codec.ImageDecoder
Returns the current parameters as an instance of the ImageDecodeParam interface.
getParam() - Method in interface com.sun.media.jai.codec.ImageEncoder
Returns the current parameters as an instance of the ImageEncodeParam interface.
getParamClasses() - Method in interface javax.media.jai.OperationDescriptor
Returns an array of `Class`es that describe the types of parameters required by this operation.
getParamClasses() - Method in class javax.media.jai.OperationDescriptorImpl
Returns the parameter class types of this operation.
getParamDefaults() - Method in interface javax.media.jai.OperationDescriptor
Returns an array of `Object`s that define the default values of the parameters for this operation.
getParamDefaults() - Method in class javax.media.jai.OperationDescriptorImpl
Returns the default values of the parameters for this operation.
getParamDefaultValue(int) - Method in interface javax.media.jai.OperationDescriptor
Returns the default value of a specified parameter.
getParamDefaultValue(int) - Method in class javax.media.jai.OperationDescriptorImpl
Returns the default value of specified parameter.
getParameterBlock() - Method in class javax.media.jai.RenderedOp
Returns the `ParameterBlock` of this node.
getParameterBlock() - Method in class javax.media.jai.RenderableOp
Returns the `ParameterBlock` of this node.
getParameterBlock() - Method in class javax.media.jai.CollectionOp
Returns the `ParameterBlock` of this node.

lookup(Raster, WritableRaster, Rectangle) - Method in class javax.media.jai.LookupTableJAI
Performs table lookup on a source Raster, writing the result into a supplied WritableRaster.
lookup(WritableRaster) - Method in class javax.media.jai.LookupTableJAI
Performs table lookup in place on a given WritableRaster.
lookupCIF(CollectionImageFactory) - Method in class javax.media.jai.OperationGraph
Locates a CIF within the vector of PartialOrderNodes.
LookupDescriptor - class javax.media.jai.operator.LookupDescriptor.
An `OperationDescriptor` describing the "Lookup" operation.
LookupDescriptor() - Constructor for class javax.media.jai.operator.LookupDescriptor
Constructor.
lookupDouble(int, int) - Method in class javax.media.jai.LookupTableJAI
Performs lookup on a given value belonging to a given source band, and returns the result as a double.
lookupFloat(int, int) - Method in class javax.media.jai.LookupTableJAI
Performs lookup on a given value belonging to a given source band, and returns the result as a float.
lookupOp(String) - Method in class javax.media.jai.ProductOperationGraph
Locates a product from within the vector of PartialOrderNodes using the productName provided.
lookupRIF(RenderedImageFactory) - Method in class javax.media.jai.OperationGraph
Locates a RIF within the vector of PartialOrderNodes.
LookupTableJAI - class javax.media.jai.LookupTableJAI.
A lookup table object associated with the "Lookup" operation.
LookupTableJAI(byte[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded byte lookup table.
LookupTableJAI(byte[][]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded byte lookup table.
LookupTableJAI(byte[][], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded byte lookup table where all bands have the same index offset.
LookupTableJAI(byte[][], int[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded byte lookup table where each band has a different index offset.
LookupTableJAI(byte[], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded byte lookup table with an index offset.
LookupTableJAI(double[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded double lookup table.
LookupTableJAI(double[][]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded double lookup table.
LookupTableJAI(double[][], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded double lookup table where all bands have the same index offset.
LookupTableJAI(double[][], int[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded double lookup table where each band has a different index offset.
LookupTableJAI(double[], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded double lookup table with an index offset.
LookupTableJAI(float[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded float lookup table.
LookupTableJAI(float[][]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded float lookup table.
LookupTableJAI(float[][], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded float lookup table where all bands have the same index offset.
LookupTableJAI(float[][], int[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded float lookup table where each band has a different index offset.
LookupTableJAI(float[], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded float lookup table with an index offset.
LookupTableJAI(int[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded int lookup table.
LookupTableJAI(int[][]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded int lookup table.
LookupTableJAI(int[][], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded int lookup table where all bands have the same index offset.
LookupTableJAI(int[][], int[]) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded int lookup table where each band has a different index offset.
LookupTableJAI(int[], int) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a single-banded int lookup table with an index offset.
LookupTableJAI(short[][], boolean) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded short or unsigned short lookup table.
LookupTableJAI(short[][], int[], boolean) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded short or unsigned short lookup table where each band has a different index offset.
LookupTableJAI(short[][], int, boolean) - Constructor for class javax.media.jai.LookupTableJAI
Constructs a multi-banded short or unsigned short lookup table where all bands have the same index offset.

removeSinks() - Method in class javax.media.jai.PlanarImage
Clears the list of sinks.
removeSource(PlanarImage) - Method in class javax.media.jai.PlanarImage
Removes a `PlanarImage` source from the list of sources.
removeSource(PlanarImage) - Method in class javax.media.jai.RenderedOp
Renders the node if it has not already been rendered, and removes a `PlanarImage` source from the list of sources of the rendered image.
removeSources() - Method in class javax.media.jai.PlanarImage
Clears the list of sources.
removeSources() - Method in class javax.media.jai.RenderedOp
Removes all the sources stored in the `ParameterBlock` of this node.
removeSuppressedProps(PropertyGenerator) - Method in class javax.media.jai.PropertySourceImpl

removeTileObserver(TileObserver) - Method in class javax.media.jai.WritableRenderedImageAdapter
Remove an observer.
removeTileObserver(TileObserver) - Method in class javax.media.jai.TiledImage
Informs this `TiledImage` that a particular TileObserver no longer wishes to receive updates on tile writability status.
removeTiles(RenderedImage) - Method in interface javax.media.jai.TileCache
Advises the cache that all tiles associated with a given image are no longer needed.
removeUnsafeToCopyPrivateChunks() - Method in class com.sun.media.jai.codec.PNGEncodeParam
Remove all private chunks associated with this parameter instance whose ’safe-to-copy’ bit is not set.
removeViewportListener(ViewportListener) - Method in class javax.media.jai.widget.ScrollingImagePanel
Removes the specified ViewportListener
RenderableDescriptor - class javax.media.jai.operator.RenderableDescriptor.
An `OperationDescriptor` describing the "Renderable" operation.
RenderableDescriptor() - Constructor for class javax.media.jai.operator.RenderableDescriptor
Constructor.
RenderableGraphics - class javax.media.jai.RenderableGraphics.
An implementation of `Graphics2D` with `RenderableImage` semantics.
RenderableGraphics(Rectangle2D) - Constructor for class javax.media.jai.RenderableGraphics
Constructs a `RenderableGraphics` given a bounding `Rectangle2D`.
RenderableGraphics(Rectangle2D, LinkedList, Point, Graphics2D) - Constructor for class javax.media.jai.RenderableGraphics
Constructs a `RenderableGraphics` given a bounding `Rectangle2D`, an origin, and a `Graphics2D` object from which to initialize the `RenderableGraphics` state.
RenderableImageAdapter - class javax.media.jai.RenderableImageAdapter.
An adapter class for externally-generated RenderableImages.
RenderableImageAdapter(RenderableImage) - Constructor for class javax.media.jai.RenderableImageAdapter
Constructs a RenderableImageAdapter from a RenderableImage.
RenderableOp - class javax.media.jai.RenderableOp.
A JAI version of RenderableImageOp.
RenderableOp(OperationRegistry, String, ParameterBlock) - Constructor for class javax.media.jai.RenderableOp
Constructs a RenderableOp given the name of the operation to be performed and a ParameterBlock containing RenderableImage sources and other parameters.
RenderableOp(String, ParameterBlock) - Constructor for class javax.media.jai.RenderableOp
Constructs a RenderableOp given the name of the operation to be performed and a ParameterBlock containing RenderableImage sources and other parameters.
renderableSourceClasses - Variable in class javax.media.jai.OperationDescriptorImpl
An array of `Class`es that describe the types of sources required by this operation in the renderable mode.
RenderedImageAdapter - class javax.media.jai.RenderedImageAdapter.
A PlanarImage wrapper for a non-writable RenderedImage.
RenderedImageAdapter(RenderedImage) - Constructor for class javax.media.jai.RenderedImageAdapter
Constructs a RenderedImageAdapter.
RenderedOp - class javax.media.jai.RenderedOp.
A node in a rendered imaging chain.
RenderedOp(OperationRegistry, String, ParameterBlock, RenderingHints) - Constructor for class javax.media.jai.RenderedOp
Constructs a `RenderedOp` that will be used to instantiate a particular rendered operation from a given operation registry, an operation name, a `ParameterBlock`, and a set of rendering hints.
RenderedOp(String, ParameterBlock, RenderingHints) - Constructor for class javax.media.jai.RenderedOp
Constructs a `RenderedOp` that will be used to instantiate a particular rendered operation from a given operation registry, an operation name, a `ParameterBlock`, and a set of rendering hints.
renderedSource - Variable in class javax.media.jai.MultiResolutionRenderableImage
An array of RenderedImage sources.
renderHints - Variable in class javax.media.jai.RenderedOp
The rendering hints to use for this operation.

thePropertySource - Variable in class javax.media.jai.RenderedOp
The `PropertySource` containing the combined properties of all of the node's sources.
thePropertySource - Variable in class javax.media.jai.RenderableOp

theRegistry - Variable in class javax.media.jai.RenderedOp
The `OperationRegistry` that is used to render this node.
theRegistry - Variable in class javax.media.jai.RenderableOp
The `OperationRegistry` that is used to render this node.
theShape - Variable in class javax.media.jai.ROIShape
The internal Shape that defines this mask.
threshold - Variable in class javax.media.jai.ROI
The inclusion/exclusion threshold of the ROI.
ThresholdDescriptor - class javax.media.jai.operator.ThresholdDescriptor.
An `OperationDescriptor` describing the "Threshold" operation.
ThresholdDescriptor() - Constructor for class javax.media.jai.operator.ThresholdDescriptor
Constructor.
TIFF_ASCII - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for null-terminated ASCII strings.
TIFF_BYTE - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 8 bit unsigned integers.
TIFF_DOUBLE - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 64 bit IEEE doubles.
TIFF_FLOAT - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 32 bit IEEE floats.
TIFF_LONG - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 32 bit unsigned integers.
TIFF_RATIONAL - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for pairs of 32 bit unsigned integers.
TIFF_SBYTE - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 8 bit signed integers.
TIFF_SHORT - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 16 bit unsigned integers.
TIFF_SLONG - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 32 bit signed integers.
TIFF_SRATIONAL - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for pairs of 32 bit signed integers.
TIFF_SSHORT - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 16 bit signed integers.
TIFF_UNDEFINED - Static variable in class com.sun.media.jai.codec.TIFFField
Flag for 8 bit uninterpreted bytes.
TIFFDecodeParam - class com.sun.media.jai.codec.TIFFDecodeParam.
An instance of `ImageDecodeParam` for decoding images in the TIFF format.
TIFFDecodeParam() - Constructor for class com.sun.media.jai.codec.TIFFDecodeParam
Constructs a default instance of `TIFFDecodeParam`.
TIFFDescriptor - class javax.media.jai.operator.TIFFDescriptor.
An `OperationDescriptor` describing the "TIFF" operation.
TIFFDescriptor() - Constructor for class javax.media.jai.operator.TIFFDescriptor
Constructor.
TIFFDirectory - class com.sun.media.jai.codec.TIFFDirectory.
A class representing an Image File Directory (IFD) from a TIFF 6.0 stream.
TIFFDirectory() - Constructor for class com.sun.media.jai.codec.TIFFDirectory
The default constructor.
TIFFDirectory(SeekableStream, int) - Constructor for class com.sun.media.jai.codec.TIFFDirectory
Constructs a TIFFDirectory from a SeekableStream.
TIFFDirectory(SeekableStream, long) - Constructor for class com.sun.media.jai.codec.TIFFDirectory
Constructs a TIFFDirectory by reading a SeekableStream.
TIFFEncodeParam - class com.sun.media.jai.codec.TIFFEncodeParam.
An instance of `ImageEncodeParam` for encoding images in the TIFF format.
TIFFEncodeParam() - Constructor for class com.sun.media.jai.codec.TIFFEncodeParam
Constructs an TIFFEncodeParam object with default values for parameters.
TIFFField - class com.sun.media.jai.codec.TIFFField.
A class representing a field in a TIFF 6.0 Image File Directory.
TIFFField() - Constructor for class com.sun.media.jai.codec.TIFFField
The default constructor.
TIFFField(int, int, int, Object) - Constructor for class com.sun.media.jai.codec.TIFFField
Constructs a TIFFField with arbitrary data.