Horus C++ Reference

Version 1.1 - Jan 2002

Dennis Koelma

And other ISIS members

Intelligent Sensory Information Systems
University of Amsterdam, Faculty of Science
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
koelma@science.uva.nl
http://www.science.uva.nl/~horus/

Contents

Horus C++ Reference

This document

Related documents

- `Horus User Guide`
- `Horus Java Reference`
- `Horus IDL Reference`
- `IDL - C++ Binding Reference`
- `IDL - Java Binding Reference`

Author:
Dennis Koelma

Chapter 1

Class Overview

1.1 Class overview

The most important user level classes:

HxImageRep (p. 313), **HxImageFactory** (p. 298), **HxImageSeq** (p. 333), **HxHistogram** (p. 259), **HxNJet** (p. 627), **HxBSplineCurve** (p. 213), **HxSampledBSplineCurve** (p. 658)

Basic classes and types:

HxScalarInt (p. 696), **HxScalarDouble** (p. 677), **HxVec2Int** (p. 785), **HxVec2Double** (p. 766), **HxVec3Int** (p. 824), and **HxVec3Double** (p. 804).

HxPoint (p. 140), **HxPointInt** (p. 140), **HxPointZ** (p. 644), **HxPointList.h**

HxString (p. 152), **HxStringList.h**

HxMatrix (p. 556), **HxVector** (p. 844)

Image related classes:

HxImageSignature (p. 360), **HxGeoIntType** (p. 91), **HxSizes** (p. 150), **HxTagList** (p. 716), **HxValue** (p. 757), **HxValueList.h**, **HxValueType** (p. 164)

Chapter 2

Pixels

2.1 Pixels

2.2 Unary operations

Unary operations on $\mathbf{x} \in \{\mathbf{Z}^n, \mathbf{R}^n\}$:

dimension	$\dim(\mathbf{x}) = n$	scalar result
negation	$-\mathbf{x} = (-x_1, ..., -x_n)$	
complement	$\sim\mathbf{x} = (\sim x_1, ..., \sim x_n)$	one's complement, integer only
abs	$\lvert\mathbf{x}\rvert = (\lvert x_1\rvert, ..., \lvert x_n\rvert)$	
ceil	$\lceil\mathbf{x}\rceil = (\lceil x_1\rceil, ..., \lceil x_n\rceil)$	
floor	$\lfloor\mathbf{x}\rfloor = (\lfloor x_1\rfloor, ..., \lfloor x_n\rfloor)$	
round	$[\mathbf{x}] = ([x_1], ..., [x_n])$	
projection	$p_i(\mathbf{x}) = x_i$	scalar result
sum	$\sum\mathbf{x} = x_1 + ... + x_n$	scalar result
product	$\prod\mathbf{x} = x_1 ... x_n$	scalar result
minimum	$\wedge\mathbf{x} = x_1 \wedge ... \wedge x_n$	scalar result
maximum	$\vee\mathbf{x} = x_1 \vee ... \vee x_n$	scalar result
L^1 norm	$\lvert\lvert\mathbf{x}\rvert\rvert_1 = \lvert x_1\rvert + ... + \lvert x_n\rvert$	cityblock, scalar result
L^2 norm	$\lvert\lvert\mathbf{x}\rvert\rvert_2 = \sqrt{{x_1}^2 + ... + {x_n}^2}$	Euclidian, floating point scalar
L^∞ norm	$\lvert\lvert\mathbf{x}\rvert\rvert_\infty = \lvert x_1\rvert \vee ... \vee \lvert x_n\rvert$	chessboard, scalar result
square root	$\sqrt{\mathbf{x}} = (\sqrt{x_1}, ..., \sqrt{x_n})$	floating point result
sine	$\sin(\mathbf{x}) = (\sin(x_1), ..., \sin(x_n))$	floating point result
cosine	$\cos(\mathbf{x}) = (\cos(x_1), ..., \cos(x_n))$	floating point result
tangent	$\tan(\mathbf{x}) = (\tan(x_1), ..., \tan(x_n))$	floating point result
arc sine	$\mathrm{asin}(\mathbf{x}) = (\mathrm{asin}(x_1), ..., \mathrm{asin}(x_n))$	floating point result
arc cosine	$\mathrm{acos}(\mathbf{x}) = (\mathrm{acos}(x_1), ..., \mathrm{acos}(x_n))$	floating point result
arc tangent	$\mathrm{atan}(\mathbf{x}) = (\mathrm{atan}(x_1), ..., \mathrm{atan}(x_n))$	floating point result
arc tangent	$\mathrm{atan2}(\mathbf{x}) = \mathrm{atan2}(x_1, x_2)$	n = 2 only, floating point scalar
hyperbolic sin	$\sinh(\mathbf{x}) = (\sinh(x_1), ..., \sinh(x_n))$	floating point result
hyperbolic cos	$\cosh(\mathbf{x}) = (\cosh(x_1), ..., \cosh(x_n))$	floating point result
hyperbolic tan	$\tanh(\mathbf{x}) = (\tanh(x_1), ..., \tanh(x_n))$	floating point result
exponent	$\exp(\mathbf{x}) = (e^{x_1}, ..., e^{x_n})$	floating point result
natural log	$\log(\mathbf{x}) = (\log_e(x_1), ..., \log_e(x_n))$	floating point result
base 10 log	$\log 10(\mathbf{x}) = (\log_{10}(x_1), ..., \log_{10}(x_n))$	floating point result

2.3 Binary operations

Binary operations on $\mathbf{x}, \mathbf{y} \in \{\mathbf{Z}^n, \mathbf{R}^n\}$:

addition	$\mathbf{x} + \mathbf{y} = (x_1 + y_1, ..., x_n + y_n)$	
subtraction	$\mathbf{x} - \mathbf{y} = (x_1 - y_1, ..., x_n - y_n)$	
multiplication	$\mathbf{x} * \mathbf{y} = (x_1 y_1, ..., x_n y_n)$	Hadamard product
division	$\mathbf{x}/\mathbf{y} = (x_1/y_1, ..., x_n/y_n)$	
minimum	$\mathbf{x} \wedge \mathbf{y} = \begin{cases} \mathbf{x} & \text{if } \mathbf{x} < \mathbf{y} \\ \mathbf{y} & \text{otherwise} \end{cases}$	
maximum	$\mathbf{x} \vee \mathbf{y} = \begin{cases} \mathbf{x} & \text{if } \mathbf{x} > \mathbf{y} \\ \mathbf{y} & \text{otherwise} \end{cases}$	
infimum	$\mathbf{x} \wedge \mathbf{y} = (x_1 \wedge y_1, ..., x_n \wedge y_n)$	
supremum	$\mathbf{x} \vee \mathbf{y} = (x_1 \vee y_1, ..., x_n \vee y_n)$	
power	$\mathbf{x}^{\mathbf{y}} = (x_1^{y_1}, ..., x_n^{y_n})$	
modulo	$\mathbf{x} \,\%\, \mathbf{y} = (x_1 \,\%\, y_1, ..., x_n \,\%\, y_n)$	integer only
and	$\mathbf{x} \,\&\, \mathbf{y} = (x_1 \,\&\, y_1, ..., x_n \,\&\, y_n)$	integer only
or	$\mathbf{x} \mid \mathbf{y} = (x_1 \mid y_1, ..., x_n \mid y_n)$	integer only
xor	$\mathbf{x} \,\hat{}\, \mathbf{y} = (x_1 \,\hat{}\, y_1, ..., x_n \,\hat{}\, y_n)$	integer only
left shift	$\mathbf{x} \ll \mathbf{y} = (x_1 \ll y_1, ..., x_n \ll y_n)$	integer only
right shift	$\mathbf{x} \gg \mathbf{y} = (x_1 \gg y_1, ..., x_n \gg y_n)$	integer only
dot	$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + ... + x_n y_n$	scalar result
cross	$\mathbf{x} \times \mathbf{y} = (x_2 y_3 - x_3 y_2, ...)$	n = 3 only
equal	$\mathbf{x} = \mathbf{y}$	scalar result
not equal	$\mathbf{x} \neq \mathbf{y}$	scalar result

The comparison operations given below are for $n > 1$.
For $n = 1$ the standard definition is used.

less than	$\mathbf{x} < \mathbf{y} = \begin{cases} 1 & \text{if } \|\|\mathbf{x}\|\|_1 < \|\|\mathbf{y}\|\|_1 \\ 0 & \text{otherwise} \end{cases}$	scalar result
greater than	$\mathbf{x} > \mathbf{y} = \begin{cases} 1 & \text{if } \|\|\mathbf{x}\|\|_1 > \|\|\mathbf{y}\|\|_1 \\ 0 & \text{otherwise} \end{cases}$	scalar result
less equal	$\mathbf{x} \leq \mathbf{y} = \begin{cases} 1 & \text{if } \|\|\mathbf{x}\|\|_1 \leq \|\|\mathbf{y}\|\|_1 \\ 0 & \text{otherwise} \end{cases}$	scalar result
greater equal	$\mathbf{x} \geq \mathbf{y} = \begin{cases} 1 & \text{if } \|\|\mathbf{x}\|\|_1 \geq \|\|\mathbf{y}\|\|_1 \\ 0 & \text{otherwise} \end{cases}$	scalar result

2.4 Arithmetic types

The standard unary and binary pixel operations are defined for all arithmetic datatypes:

HxScalarInt (p. 696), **HxScalarDouble** (p. 677), **HxVec2Int** (p. 785), **HxVec2Double** (p. 766), **HxVec3Int** (p. 824), and **HxVec3Double** (p. 804).

2.5 Color conversion functions

The supported color models are enumerated in **HxColorModel** (p. 50). Color semantics are defined in **HxColor** (p. 231).

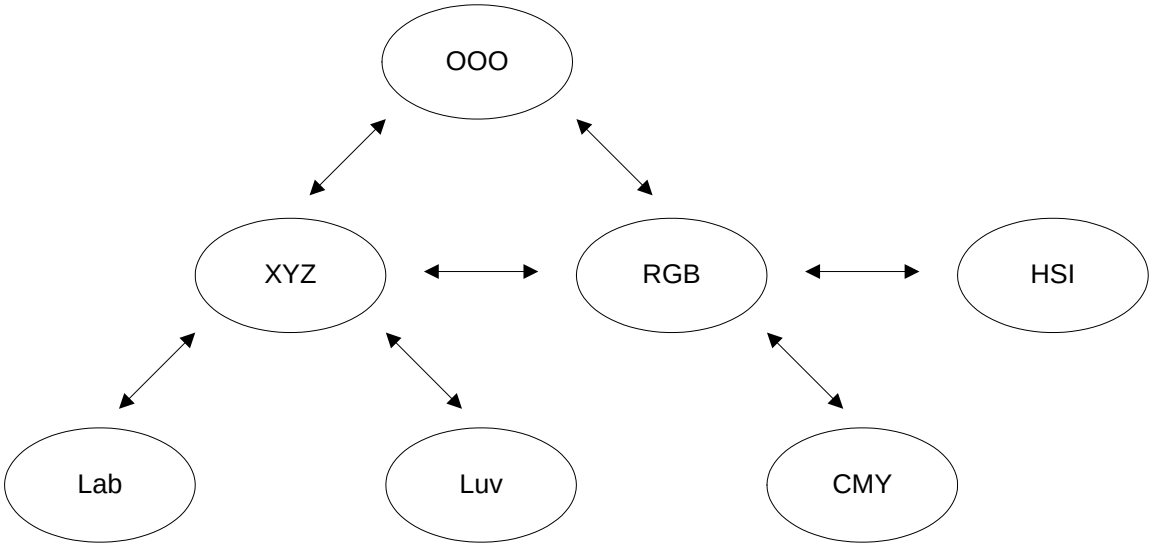


Figure 2.1: Color spaces

Functions for conversion of colors from one color space to another

RGB <-> CMY : **HxColRGB2CMY** (p. 44) and **HxColCMY2RGB** (p. 45).

RGB <-> XYZ : **HxColRGB2XYZ** (p. 45) and **HxColXYZ2RGB** (p. 45).

CMY <-> XYZ : **HxColCMY2XYZ** (p. 45) and **HxColXYZ2CMY** (p. 46).

Lab <-> XYZ : **HxColLab2XYZ** (p. 46) and **HxColXYZ2Lab** (p. 46).

Luv <-> XYZ : **HxColLuv2XYZ** (p. 47) and **HxColXYZ2Luv** (p. 47).

RGB <-> OOO : **HxColRGB2OOO** (p. 47) and **HxColOOO2RGB** (p. 47).

XYZ <-> OOO : **HxColXYZ2OOO** (p. 48) and **HxColOOO2XYZ** (p. 48).

RGB <-> HSI : **HxColRGB2HSI** (p. 48) and **HxColHSI2RGB** (p. 49).

Chapter 3

Images

3.1 Images

Our definition of images and operations on images is given in **Image definition and generic operations** (p. 9).

More down to earth, image data is stored in an **HxImageRep** (p. 313) and processed via functions such as listed in **Global image functions** (p. 11).

If you really want to know what is going on, see **Image data representation** (p. 12) and **The way image operations work** (p. 13).

This chapter:

3.2 Image definition and generic operations

3.3 Image definition

A digital image consists of a set of pixels. Associated with each pixel is a location (point) $\mathbf{x}$ and a (pixel) value $\mathbf{a}(\mathbf{x})$. The set of all points $\mathbf{x}$ is referred to as the domain of the image, and is denoted by $\mathbf{X}$. Usually, $\mathbf{X}$ is an n-dimensional lattice with $n = 1, 2,$ or 3. Also, the point set is bounded in each dimension resulting in a rectangular shape for $n = 2$ and a block shape for $n = 3$. That is, for a 2-dimensional $w \times h$ image $\mathbf{X} = \mathbf{Z}_w \times \mathbf{Z}_h = \{(x_1, x_2) \in \mathbf{Z}^2 : 0 \leq x_1 \leq w - 1, 0 \leq x_2 \leq h - 1\}$ ($\mathbf{Z}_n = \{0, 1, ..., n - 1\}$).

The set of all pixel values $\mathbf{a}(\mathbf{x})$ is referred to as the *range* of an image, and is denoted by $\mathbf{F}$. A pixel value is a scalar value or a vector of n scalar values (usually $n = 2$ or 3). A scalar value is represented by one of the following:

- a k bits integer value (bit, byte, short, int, ...)
- a k bits floating point value (float, double, ...)
- a complex number

For example, for color pixels represented by RGB values $\mathbf{F} = \mathbf{Z}_{2^k} \times \mathbf{Z}_{2^k} \times \mathbf{Z}_{2^k}$ (typically $k = 8$). In summary, an image $\mathbf{a}$ is a shorthand notation for $\{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{x} \in \mathbf{Z}^n (n = 1, 2, 3), \mathbf{a}(\mathbf{x}) \in \{\mathbf{Z}^n, \mathbf{R}^n, \mathbf{C}\} (n = 1, 2, 3)\}$.

In case the pixels in an n-dimensional image contain multiple values (say m values) we often encounter two different ways of addressing the image content. The first way is to regard the image to be an n-dimensional field of m-dimensional vectors, i.e.

$$\mathbf{a} = \{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{x} \in \mathbf{Z}^n, \mathbf{a}(\mathbf{x}) \in \mathbf{F}_1 \times \mathbf{F}_2 \times ... \times \mathbf{F}_m\}$$

The second way is to regard the image to be a set of m n-dimensional fields of scalars:

$$\mathbf{a} = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\} \text{ with } \mathbf{a}_i = \{(\mathbf{x}, \mathbf{a_i}(\mathbf{x})) : \mathbf{x} \in \mathbf{Z}^n, \mathbf{a_i}(\mathbf{x}) \in \mathbf{F}_i\}$$

We take the former way the basic representation of vector images in our system. The latter way is also supported by means of projection functions but then the user has to keep track of the set of scalar images that represent a single vector image.

3.4 Generic image operations

- unary pixel operations: $\mathbf{c} = f(\mathbf{a}) = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = f(\mathbf{a}(\mathbf{x})), \mathbf{x} \in \mathbf{X}\}$, with f a unary operation on $\mathbf{F}$.
- binary pixel operations: $\mathbf{c} = \mathbf{a}\gamma\mathbf{b} = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x})\gamma\mathbf{b}(\mathbf{x})), \mathbf{x} \in \mathbf{X}\}$, with γ a binary operation on $\mathbf{F}$.

  The operand may also be a scalar k: $\mathbf{c} = \mathbf{a}\gamma k = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x})\gamma k), \mathbf{x} \in \mathbf{X}\}$
- reduce operations: $\Gamma\mathbf{a} = \Gamma_{\mathbf{x}\in\mathbf{X}}\mathbf{a}(\mathbf{x}) = \Gamma_{i=1}^{n}\mathbf{a}(x_i) = \mathbf{a}(x_1)\gamma\mathbf{a}(x_2)\gamma...\gamma\mathbf{a}(x_n)$, with γ an associative and commutative binary operation on $\mathbf{F}$.
- generalized convolution: $\mathbf{c} = \mathbf{a} \textcircled{\gamma} \mathbf{t} = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{\Gamma}_{\mathbf{y}\in\mathbf{Y}}\mathbf{a}(\mathbf{x} - \mathbf{y}) \bigcirc \mathbf{t}(\mathbf{y}), \mathbf{x} \in \mathbf{X}\}$, with $\bigcirc$ and γ binary operations on $\mathbf{F}$, and $\mathbf{t} = \{(\mathbf{y}, \mathbf{t}(\mathbf{y})) : \mathbf{y} \in \mathbf{Y}\}$. Here, $\mathbf{Y} = \mathbf{Z}_{\pm w} \times \mathbf{Z}_{\pm h} \times \mathbf{Z}_{\pm d} = \{(x_1, x_2, x_3) \in \mathbf{Z}^3 : -w + 1 \leq x_1 \leq w - 1, -h + 1 \leq x_2 \leq h - 1, -d + 1 \leq x_3 \leq d - 1\}$ ($\mathbf{Z}_{\pm n} = \{-n + 1, ..., -1, 0, 1, ..., n - 1\}$). In order for a template operation to be applied near the edge of an image a frame is added. Pixel values in a frame are set to the zero-element in the computation, by mirroring pixel values across the edge of an image, or by tiling the image data.
- neighbourhood operations: $\mathbf{c} = \mathbf{a} \textcircled{\Lambda} N = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \Lambda_{\mathbf{y}\in N}\mathbf{a}(\mathbf{x} - \mathbf{y}), \mathbf{x} \in \mathbf{X}\}$, with N a neighbourhood ($N : \mathbf{X} \to 2^{\mathbf{X}}$) and Λ a reduce operation on the pixel values in a set ($\Lambda : (\mathbf{X}, \mathbf{F}) \to \mathbf{F}$). Note : Λ is more general than Γ in that Λ need not be defined in terms of a binary operation on $\mathbf{F}$.
- recursive neighbourhood operations: Apply neighbourhood operation repeatedly until no more changes occur. The operation can depend upon the scan direction in which the neighbourhood operation is applied.
- geometric (domain) operations: $\mathbf{c} = \mathbf{a} \circ f = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(f(\mathbf{x})), \mathbf{x} \in \mathbf{X}\}$, with f a unary operation on $\mathbf{X}$.

3.5 Global image functions

Arithmetic **unary** **HxAbs** (p. 35), **HxCeil** (p. 43), **HxComplement** (p. 51), **HxExp** (p. 63), **HxFloor** (p. 69), **HxLog** (p. 111), **HxLog10** (p. 111), **HxNegate** (p. 128), **HxNorm1** (p. 129), **HxNorm2** (p. 130), **HxNorm2Sqr** (p. 131), **HxNormInf** (p. 132), **HxProjectRange** (p. 143), **HxSqrt** (p. 150), **HxRound** (p. 147), **HxUnaryMax** (p. 159), **HxUnaryMin** (p. 160), **HxUnaryProduct** (p. 160), **HxUnarySum** (p. 161)

trigonometric **HxAcos** (p. 36), **HxArg** (p. 39), **HxAsin** (p. 39), **HxAtan** (p. 40), **HxAtan2** (p. 40), **HxConjugate** (p. 52), **HxCos** (p. 57), **HxCosh** (p. 58), **HxSin** (p. 148), **HxSinh** (p. 149), **HxTan** (p. 157), **HxTanh** (p. 158).

binary **HxAdd** (p. 36), **HxAnd** (p. 38), **HxCross** (p. 58), **HxDiv** (p. 60), **HxDot** (p. 61), **HxEqual** (p. 62), **HxGreaterEqual** (p. 92), **HxGreaterThan** (p. 93), **HxInf** (p. 106), **HxInverseProjectRange** (p. 107), **HxLeftShift** (p. 107), **HxLessEqual** (p. 108), **HxLessThan** (p. 109), **HxMax** (p. 124), **HxMin** (p. 125), **HxMod** (p. 126), **HxMul** (p. 127), **HxNotEqual** (p. 132), **HxOr** (p. 134), **HxPow** (p. 142), **HxRightShift** (p. 145), **HxSub** (p. 153), **HxSup** (p. 154), **HxXor** (p. 165).

binary value **HxAddVal** (p. 37), HxAffinePix, **HxAndVal** (p. 38), **HxCrossVal** (p. 59), **HxDivVal** (p. 61), **HxDotVal** (p. 62), **HxEqualVal** (p. 63), **HxGreaterEqualVal** (p. 92), **HxGreaterThanVal** (p. 93), **HxInfVal** (p. 106), **HxLeftShiftVal** (p. 108), **HxLessEqualVal** (p. 109), **HxLessThanVal** (p. 110), **HxMaxVal** (p. 124), **HxMinVal** (p. 126), **HxModVal** (p. 127), **HxMulVal** (p. 128), **HxNotEqualVal** (p. 133), **HxOrVal** (p. 134), **HxPowVal** (p. 142), **HxRightShiftVal** (p. 146), **HxSubVal** (p. 153), **HxSupVal** (p. 154), **HxXorVal** (p. 166).

reduce **HxPixInf** (p. 137), **HxPixMax** (p. 137), **HxPixMin** (p. 138), **HxPixProduct** (p. 138), **HxPixSum** (p. 139), **HxPixSup** (p. 139).

Color **HxColorSpace** (p. 51), HxOpponentColor, **HxRGB2Intensity** (p. 145)

Conversion **HxImageAsByte** (p. 95), **HxImageAsComplex** (p. 95), **HxImageAsDouble** (p. 96), **HxImageAsFloat** (p. 96), **HxImageAsInt** (p. 97), **HxImageAsShort** (p. 97), **HxImageAsVec2Byte** (p. 98), **HxImageAsVec2Double** (p. 98), **HxImageAsVec2Float** (p. 99), **HxImageAsVec2Int** (p. 100), **HxImageAsVec2Short** (p. 100), **HxImageAsVec3Byte** (p. 101), **HxImageAsVec3Double** (p. 101), **HxImageAsVec3Float** (p. 102), **HxImageAsVec3Int** (p. 102), **HxImageAsVec3Short** (p. 103).

Detector HxImageToHistogram, HxGreyEdgeHistogram, HxLabelBlobs

Export **HxExportByteData** (p. 64), **HxExportDoubleData** (p. 65), **HxExportFloatData** (p. 65), **HxExportIntData** (p. 66), **HxExportMatlabPixels** (p. 66), HxExportPpmPixels, **HxExportShortData** (p. 67), **HxExportSi** (p. 68), **HxImagesToFile** (p. 105), HxMatrixConv, **HxWriteFile** (p. 165).

Filter **HxCannyEdgeMap** (p. 41), **HxCannyThreshold** (p. 42), **HxCannyThresholdAlt** (p. 42), **HxConvGauss2d** (p. 53), **HxConvGauss3d** (p. 54), **HxConvKernelSeparated** (p. 55), **HxConvKernelSeparated2d** (p. 55), **HxConvolution** (p. 56), **HxConvolution1d** (p. 56), **HxDistanceTransform** (p. 59), **HxGauss** (p. 88), **HxGaussDerivative2d** (p. 89), **HxGaussDerivative3d** (p. 90), **HxNonMaxSuppressionGradDir** (p. 129), **HxNormalizedCorrelation** (p. 131), **HxParabolicDilation** (p. 135), **HxParabolicErosion** (p. 135), **HxPercentile** (p. 136), HxRecGauss, **HxUniform** (p. 162), **HxUniformNonSep** (p. 162).

Generation **HxImagesFromFile** (p. 105), **HxMakeFrom2Images** (p. 112), **HxMakeFrom3Images** (p. 112), **HxMakeFromByteData** (p. 113), **HxMakeFromDoubleData** (p. 113), **HxMakeFromFile** (p. 114), **HxMakeFromFileSI** (p. 114), **HxMakeFromFloatData** (p. 115), **HxMakeFromGrayValue** (p. 116), **HxMakeFromImage** (p. 117), **HxMakeFromImport** (p. 117), **HxMakeFromIntData** (p. 118), **HxMakeFromJavaRgb** (p. 118), **HxMakeFromMatlab** (p. 119), **HxMakeFromNamedGenerator** (p. 119), HxMakeFromPpmPixels, **HxMakeFromShortData** (p. 120), **HxMakeFromSi** (p. 121), **HxMakeFromSignature** (p. 121), **HxMakeFromValue** (p. 122), **HxMakeGaussian1d** (p. 122), **HxMakeParabola1d** (p. 123)

Geometric **HxExtend** (p. 68), **HxExtendVal** (p. 69), **HxReflect** (p. 144), **HxRestrict** (p. 144), **HxRotate** (p. 146), **HxScale** (p. 148).

Inquiry **HxImageMaxSize** (p. 103), **HxImageMinSize** (p. 104).

Pixel **HxContrastStretch** (p. 52), **HxSquaredDistance** (p. 151)

Sample HxGetPoints, HxGetValues, HxIdentMaskSum, HxIdentMaskMean, HxIdentMaskStDev, HxMaskSum.

Segmentation **HxThreshold** (p. 158), **HxTriStateThreshold** (p. 159).

3.6 Image data representation

The **HxImageRep** (p. 313) and **HxImageData** (p. 288) classes, etc.

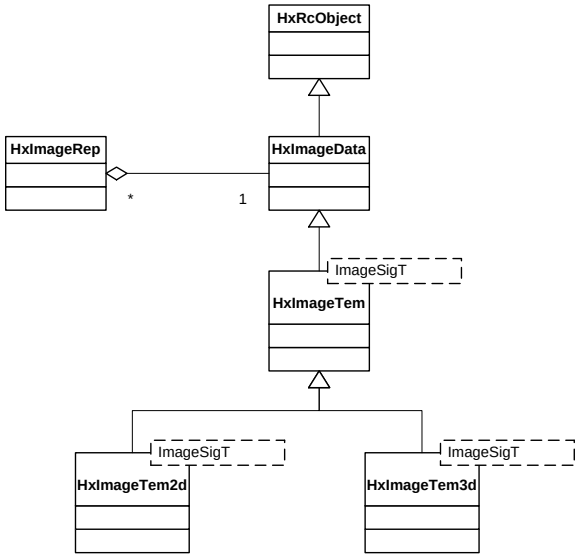


Figure 3.1: Image class hierarchy

HxImageFactory (p. 298), HxImageCreator, and HxDataPtrCreator.

Wich image types are available : **Image signatures** (p. 12) and instantiations.

3.6.1 Image signatures

Base class : **HxImageSignature** (p. 360).

The list of signatures

2D images with pixels represented by a scalar value **HxImageSig2dByte** (p. 344), **HxImageSig2dComplex** (p. 345), **HxImageSig2dDouble** (p. 345), **HxImageSig2dFloat** (p. 346), **HxImageSig2dInt** (p. 347), **HxImageSig2dShort** (p. 348).

2D images with pixels represented by a vector of 2 scalars **HxImageSig2dVec2Byte** (p. 348), **HxImageSig2dVec2Double** (p. 349), **HxImageSig2dVec2Float** (p. 350), **HxImageSig2dVec2Int** (p. 351), **HxImageSig2dVec2Short** (p. 351).

2D images with pixels represented by a vector of 3 scalars **HxImageSig2dVec3Byte** (p. 352), **HxImageSig2dVec3Double** (p. 353), **HxImageSig2dVec3Float** (p. 354), **HxImageSig2dVec3Int** (p. 354), **HxImageSig2dVec3Short** (p. 355).

3D images with pixels represented by a scalar value **HxImageSig3dByte** (p. 356), **HxImageSig3dDouble** (p. 357), **HxImageSig3dFloat** (p. 357), **HxImageSig3dInt** (p. 358), **HxImageSig3dShort** (p. 359).

3.7 The way image operations work

This is a description of the interaction between the various concepts that work together to implement an image processing operation. The description servers as a road map for analysis of the implementation. Also, it shows where the user is actually interacting with the infrastructure when a new image processing operation is being implemented.

To make the description more concrete, we trace the implementation of HxAbs. Note that you can easily trace the implementation of any operation in **Global image functions** (p. 11) via the source code incorporated in the documentation.

(1) Global image functions : (HxAbs (p. 35)**)** Operations on images are invoked via a global function (see **Global image functions** (p. 11) for an overview). Basically, a global function is a wrapper around a member function of **HxImageRep** (p. 313) that may do some prior initialization and translation of parameters.

So, **HxAbs** (p. 35) calls **HxImageRep::unaryPixOp** (p. 322).

(2) HxImageRep member functions (HxImageRep::unaryPixOp (p. 322)**)** The member functions of **HxImageRep** (p. 313) match the generic operations on images and their variations listed in **Image processing patterns and variations** (p. 16). The most important task of the member function itself is to instantiate a method frame (see **Method frames** (p. 14) for an overview) to ...

So, **HxImageRep::unaryPixOp** (p. 322) calls **HxImageData::unaryPixOp** (p. 293).

(3) HxImageData member functions (HxImageData::unaryPixOp (p. 293)**)** The member functions of **HxImageData** (p. 288) try to locate the approriate image functor (see **Image functors** (p. 15) for an overview). Lookup is based on a key. A key is constructed from the signature of the object image (this image) image and the name of the image functor. When present in the function, other image parameters may also be incorporated in the key. Note that the type of the key is ”linked” to the member function. For example, unaryPixOp uses **HxImgFtorUpoKey** (p. 489).

With the key, an image functor is searched for in the functor table and downcast to the level of ”I-nary” functors (see **Image functors** (p. 15)). The functor found is given the task of invoking the actual functor (the ”real” functor, see **Image functors** (p. 15)) via callIt.

So, **HxImageData::unaryPixOp** (p. 293) calls **HxImgFtorI2::callIt** (p. 405).

(4) ”I-nary” level image functors (HxImgFtorI2::callIt (p. 405)**)** This is actually a pure virtual member function. It is implemented one level lower in the tree (the ”cast” level) by a template class.

So, we end up in **HxImgFtorI2Cast::callIt** (p. 410).

(5-1) ”cast” level image functors (HxImgFtorI2Cast::callIt (p. 410)**)** At the ”cast” level, the polymorphic **HxImageData** (p. 288) parameters are converted to statically typed data pointer parameters. The data pointer parameters are passed to the doIt function of the same class.

So, **HxImgFtorI2Cast::callIt** (p. 410) calls **HxImgFtorI2Cast::doIt** (p. 411).

(5-2) "cast" level image functors (HxImgFtorI2Cast::doIt (p. 411)**)** This is actually a pure virtual member function. It is implemented at the lowest level in the tree by the "real functor".

So, we end up in **HxImgFtorUpo::doIt** (p. 488).

(6) "real" image functors (HxImgFtorUpo::doIt (p. 488)**)** The "real" functors introduce additional information for the pattern to work, typically in the form of template parameters. The number and type of the template parameters depend upon the pattern. For example, **HxImgFtorUpo** (p. 486) has only one : UpoT.

The template parameters may use information from the tag list for run-time initialization. To that end, the tag list is passed to the constructor of the template parameter.

Variations on patterns are typically specified through typedefs in the template parameter. Based on these typedefs, the appropriate global template function is selected to do the actual work. For the unary pixel operation this is not (yet) implemented, but see **HxImgFtorInOut::doIt** (p. 441) and **HxFuncInOutPixOp**(DataPtrT,HxSizes,PixOpT&) (p. 86) for an example.

So, **HxImgFtorUpo::doIt** (p. 488) calls **HxFuncUnaryPixOp** (p. 82)

(7-1) global dispatch functions (not implemented for upo) A dispatch function is a global template function that selects the appropriate global template function implementing a variation on a pattern via typedefs in a template parameter.

See **HxFuncInOutPixOp**(DataPtrT,HxSizes,PixOpT&) (p. 86) for an example.

(7-2) global image processing functions (HxFuncUnaryPixOp (p. 82)**)** At last, we have come to the function that actually implements the pattern and does some image processing. That is, the function implements "the loop over all pixels in the image" and calls user defined functors at the pre-defined moments.

Note that two times a virtual function was used to go down in the functor hierarchy. Ensuring the "right" path is enforced by the instantiations of the actual image functors and the keys. See **Abs instantiations** (p. **??**) for the instantiations of our HxAbs example.

Todo:

explain keys, explain link between **HxImageRep** (p. 313) member functions and image functor instantiations

3.7.1 Method frames

- **HxMfBpo** (p. 583)
- **HxMfBroadestSig** (p. 586)
- **HxMfGenConv** (p. 589)
- **HxMfIdentity** (p. 592)
- **HxMfKernelNgb** (p. 594)
- **HxMfMNpo** (p. 596)
- **HxMfMpo** (p. 599)
- **HxMfNgb** (p. 601)
- **HxMfReqIntermediatePixType** (p. 603)
- **HxMfReqKernelPixType** (p. 606)
- **HxMfReqMaskPixType** (p. 608)
- **HxMfReqResultPixType** (p. 610)
- **HxMfResize** (p. 612)
- **HxMfTranspose** (p. 614)
- **HxMfUpo** (p. 616)

3.7.2 Image functors

In general, a functor is a function wrapped in an object. An image functor refers to an object that is used to make the user-specified functional part of a pattern do what the user wants it to do.

Image functors are collected in the **HxImgFtorTable** (p. 481). In general, functors are inserted in the table via declaration of a static variable that instantiates a leaf node image functor from the tree below for a specific image type. More accurately, the constructor of **HxImgFunctor** (p. 490) inserts itself in the table.

In general, functors are retrieved in the **HxImageData** (p. 288) member functions via **HxImgFtorTableTem** (p. 482). The template class provides statically typed access to **HxImgFtorTable** (p. 481). The **HxImageData** (p. 288) member functions access the table at the first level below the root of the tree below, i.e. **HxImgFtorI1** (p. 392), **HxImgFtorI2** (p. 400), etc.

Most important classes:

HxImgFunctor (p. 490), **HxImgFtorTableTem** (p. 482), **HxImgFtorKey** (p. 447), **HxImgFtorDescription** (p. 376), **HxImgFtorRuleBase** (p. 471), **HxImgFtorTable** (p. 481), **HxImgFtorKeyNameTable** (p. 448),

The image functor tree

The tree has four levels:

1. The root level : **HxImgFunctor** (p. 490) (for insertion and retrieval from the table)
2. The ”I-nary” level : I1 stands for ”taking 1 image parameter”, etc.
3. The ”cast” level : the highest template level for static casting of the image data pointers.
4. The ”real functor” level : the leaf nodes are the actual functors that introduce additional, pattern specific template parameters

- **HxImgFunctor** (p. 490)
  - **HxImgFtorI1** (p. 392)
    - **HxImgFtorI1Cast** (p. 394)<ImgSigT>
      - **HxImgFtorInOut** (p. 440)<ImgSigT, InOutT>
      - **HxImgFtorRgb2d** (p. 463)<ImgSigT, RgbT>
      - **HxImgFtorRgb3d** (p. 466)<ImgSigT, RgbT>
      - **HxImgFtorSetBorder2d** (p. 474)<ImgSigT>
      - **HxImgFtorSetBorder3d** (p. 477)<ImgSigT>
  - **HxImgFtorI2** (p. 400)
    - **HxImgFtorI2Cast** (p. 405)<DstImgSigT, SrcImgSigT>
      - **HxImgFtorNgb2d** (p. 456)<DstImgSigT, SrcImgSigT, NgbT>
      - **HxImgFtorRecNgb2d** (p. 459)<ImgSigT, KerImgSigT, PixOpT, RedOpT>
      - **HxImgFtorSet** (p. 472)<DstImgSigT, SrcImgSigT>
      - **HxImgFtorTranspose2d** (p. 483)<DstImgSigT, SrcImgSigT>
      - **HxImgFtorUpo** (p. 486)<DstImgSigT, SrcImgSigT, UpoT>
  - **HxImgFtorI3** (p. 416)
    - **HxImgFtorI3Cast** (p. 419)<DstImgSigT, Src1ImgSigT, Src2ImgSigT>
      - **HxImgFtorBpo** (p. 373)<DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT>
      - **HxImgFtorGenConv2d** (p. 377)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
      - **HxImgFtorGenConv2dK1d** (p. 380)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
      - **HxImgFtorGenConv3d** (p. 384)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
      - **HxImgFtorGenConv3dK1d** (p. 387)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
      - **HxImgFtorKernelNgb2d** (p. 443)<DstImgSigT, SrcImgSigT, KerImgSigT, NgbT>

- **HxImgFtorIM** (p. 427)
  - **HxImgFtorIMCast** (p. 428)<DstImgSigT, SrcImgsSigT>
    - **HxImgFtorMpo** (p. 452)<DstImgSigT, SrcImgsSigT, MpoT>
- **HxImgFtorIMN** (p. 433)
  - **HxImgFtorIMNCast** (p. 435)<DstImgSigT, SrcImgsSigT>
    - **HxImgFtorMNpo** (p. 449)<DstImgSigT, SrcImgsSigT, MNpoT>

3.8 Image processing patterns and variations

The overview reflects the current status of the implementation. Note that the list is not complete. Also, some patterns still have to be re-written to use proper image functors. The "In/Out pixel operation and variations" description is closed to the way it will be in the future.

- Unary pixel operation and variations (instantiated via the **HxImgFtorUpo** (p. 486) functor)
  - **Unary pixel operation** (p. 17)
- Binary pixel operation and variations (instantiated via the **HxImgFtorBpo** (p. 373) functor)
  - **Binary pixel operation.** (p. 17)
- Multi pixel operation and variations (instantiated via the **HxImgFtorMpo** (p. 452) functor)
  - **Multi pixel operation.** (p. 18)
- M output, N input pixel operation and variations (instantiated via the **HxImgFtorMNpo** (p. 449) functor)
  - **M output, N input pixel operation.** (p. 18)
- **In/Out pixel operation and variations.** (p. 19) (instantiated via the **HxImgFtorInOut** (p. 440) functor)
  - **Translation invariant, 1 phase pixel export operation.** (p. 19)
  - **Translation variant, 1 phase pixel export operation.** (p. 20)
  - **Translation invariant, 1 phase pixel import operation.** (p. 20)
  - **Translation variant, 1 phase pixel import operation.** (p. 20)
  - **Translation invariant, n phase pixel export operation.** (p. 21)
  - **Translation variant, n phase pixel export operation.** (p. 22)
  - **Translation invariant, n phase pixel import operation.** (p. 22)
  - **Translation variant, n phase pixel import operation.** (p. 23)
- Display operation and variations
  - **Display operation for 2D images** (p. 23) (instantiated via the **HxImgFtorRgb2d** (p. 463) functor)
  - **Display operation for 3D images** (p. 24) (instantiated via the **HxImgFtorRgb3d** (p. 466) functor)
- General convolution and variations

  On 2D images with 2D kernel functor : **HxImgFtorGenConv2d** (p. 377)

  On 2D images with 1D kernel functor : **HxImgFtorGenConv2dK1d** (p. 380)

  On 3D images with 3D kernel functor : **HxImgFtorGenConv3d** (p. 384)

  On 3D images with 1D kernel functor : **HxImgFtorGenConv3dK1d** (p. 387)
- Neighbourhood operation and variations (instantiated via the **HxImgFtorNgb2d** (p. 456) functor)

  - **2 dimensional, translation invariant, 1 phase, coordinate enumerated neighbourhood operation.** (p. 24)
  - **2 dimensional, translation invariant, 1 phase, loop neighbourhood operation.** (p. 25)
  - **2 dimensional, translation invariant, 2 phase, loop neighbourhood operation.** (p. 25)

  Neighbourhood operations with a kernel on 2D images (instantiated via the **HxImgFtorKernelNgb2d** (p. 443) functor)

  - **2 dimensional, translation invariant, 1 phase, loop neighbourhood operation with kernel.** (p. 26)
  - **2 dimensional, translation invariant, 2 phase, loop neighbourhood operation using a kernel.** (p. 26)

- Recursive neighbourhood operation and variations

  On 2D images functor : **HxImgFtorRecNgb2d** (p. 459)

- Border set operation and variations

  2D images functor : **HxImgFtorSetBorder2d** (p. 474)

  3D images functor : **HxImgFtorSetBorder3d** (p. 477)

3.8.1 Unary pixel operation

Pseudo code of the operation:

```
UnaryPixOp(Out, In, PixOp)
{
    foreach i, o in In, Out
        Out(o) = PixOp(In(i));
}
```

The requirements on the UpoT template parameter expressed as class definition are:

```
template<class DstValT, class SrcValT>
class UpoT
{
public:
                        UpoT(HxTagList&);

    DstValT             doIt(const SrcValT& x);

    static HxString     className();
};
```

The function doIt will be called with 1 parameter of type SrcImgSigT::ArithType, and the result will be stored in a variable of type DstImgSigT::ArithType before being written to the destination image.

3.8.2 Binary pixel operation.

Pseudo code of the operation:

```
BinaryPixOp(Out, In1, In2, PixOp)
{
    foreach i1, i2, o in In1, In2, Out
        Out(o) = PixOp(In1(i1), In2(i2));
}
```

The requirements on the BpoT template parameter expressed as class definition are:

```
template<class DstValT, class Src1ValT, class Src2ValT>
class BpoT
{
public:
                        BpoT(HxTagList&);

    DstValT             doIt(const Src1ValT& x, const Src2ValT& y);

    static HxString     className();
};
```

The function doIt will be called with 2 parameters of type Src1ImgSigT::ArithType and Src2ImgSigT::ArithType, and the result will be stored in a variable of type DstImgSigT::ArithType before being written to the destination image.

3.8.3 Multi pixel operation.

Pseudo code of the operation:

```
MultiPixOp(Out, In[], PixOp)
{
    foreach i, o in In, Out
        Out(o) = PixOp(In[0](i)...In[N](i));
}
```

The requirements on the MpoT template parameter expressed as class definition are:

```
template<class DstValT, class SrcValT>
class MpoT
{
public:
                    MpoT(HxTagList& tags);

    DstValT         doIt(SrcValT const *x);

    static HxString className();
};
```

The function doIt will be called with an array containing the pixels of the source images. On construction, the number of sources can be retrieved from the tag list by tag ”sourceCnt”. The result will be stored in a variable of type DstImgSigT::ArithType before being written to the destination image.

3.8.4 M output, N input pixel operation.

Pseudo code of the operation:

```
MNPixOp(Out, In[], PixOp)
{
    foreach i, o in In, Out
        Out[0](o)...Out[M](o) = PixOp(In[0](i)...In[N](i));
}
```

The requirements on the MNpoT template parameter expressed as class definition are:

```
template<class DstValT, class SrcValT>
class MNpoT
{
public:
                    MNpoT(HxTagList& tags);

    void            doIt(DstValT *y, SrcValT const *x);

    static HxString className();
};
```

The function doIt will be called with 2 arrays, destination pixels and source pixels. On construction, the number of sources can be retrieved from the tag list by tag ”sourceCnt”. The constructor should return the required number of destination images by the tag ”resultCnt”. The results should be stored in the y-array before being written to the destination images. If the operation is not applicable for the given number of source images, the constructor may indicate failure in the pre-conditioning by returning ”preOpIsOk” false in the tag list.

3.8.5 In/Out pixel operation and variations.

The pattern loops (a number of times) over all pixels in the image and ferches/offers each pixel from/to a pixel functor. With the In variation of the pattern the pixel functor is asked to produce a pixel value for each position in the image. With the Out variation of the pattern the pixel functor is offered the pixel value at each position to do something with it.

Tags for variations:

DirectionCategory **HxPixOpInTag** (p. 637) or **HxPixOpOutTag** (p. 639)

TransVarianceCategory **HxPixOpTransVarTag** (p. 640) or **HxPixOpTransInVarTag** (p. 639)

PhaseCategory **HxPixOp1PhaseTag** (p. 636) or **HxPixOp2PhaseTag** (p. 637) or **HxPixOpNPhaseTag** (p. 638)

3.8.6 Translation invariant, 1 phase pixel export operation.

Pseudo code of the operation:

```
OutPixOp(In, PixOp)
{
    foreach i in In
        PixOp(In(i));
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpOutTag           DirectionCategory;
    typedef HxPixOpTransInVarTag    TransVarianceCategory;
    typedef HxPixOp1PhaseTag        PhaseCategory;

                        PixOpT(HxTagList&);
```

```
    void                doIt(const ArithT& v);

    static HxString     className();
};
```

3.8.7 Translation variant, 1 phase pixel export operation.

Pseudo code of the operation:

```
OutPixOp(In, PixOp)
{
    foreach i in In
        PixOp(In(i), i.x, i.y, i.z);
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpOutTag           DirectionCategory;
    typedef HxPixOpTransVarTag      TransVarianceCategory;
    typedef HxPixOp1PhaseTag        PhaseCategory;

                        PixOpT(HxTagList&, int w, int h, int d);

    void                doIt(const ArithT& v, int x, int y, int z);

    static HxString     className();
};
```

3.8.8 Translation invariant, 1 phase pixel import operation.

Pseudo code of the operation:

```
InPixOp(Out, PixOp)
{
    foreach o in Out
        Out(o) = PixOp();
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpInTag            DirectionCategory;
    typedef HxPixOpTransInVarTag    TransVarianceCategory;
    typedef HxPixOp1PhaseTag        PhaseCategory;

                        PixOpT(HxTagList&);

    ArithT              doIt();

    static HxString     className();
};
```

3.8.9 Translation variant, 1 phase pixel import operation.

Pseudo code of the operation:

```
InPixOp(Out, PixOp)
{
    foreach o in Out
        Out(o) = PixOp(o.x, o.y, o.z);
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpInTag            DirectionCategory;
    typedef HxPixOpTransVarTag      TransVarianceCategory;
    typedef HxPixOp1PhaseTag        PhaseCategory;

                        PixOpT(HxTagList&, int w, int h, int d);

    ArithT              doIt(int x, int y, int z);

    static HxString     className();
};
```

3.8.10 Translation invariant, n phase pixel export operation.

Pseudo code of the operation:

```
OutPixOp(In, PixOp)
{
    for p = 1 to PixOp.nrPhases();
    {
        PixOp.init(p)
        foreach i in In
            PixOp(In(i));
        PixOp.done(p)
    }
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpOutTag           DirectionCategory;
    typedef HxPixOpTransInVarTag    TransVarianceCategory;
    typedef HxPixOpNPhaseTag        PhaseCategory;

                        PixOpT(HxTagList&);

    void                doIt(const ArithT& v);

    int                 nrPhases() const;
    void                init(int phase);
    void                done(int phase);
```

```
    static HxString     className();
};
```

3.8.11 Translation variant, n phase pixel export operation.

Pseudo code of the operation:

```
OutPixOp(In, PixOp)
{
    for p = 1 to PixOp.nrPhases();
    {
        PixOp.init(p)
        foreach i in In
            PixOp(In(i), i.x, i.y, i.z);
        PixOp.done(p)
    }
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpOutTag           DirectionCategory;
    typedef HxPixOpTransVarTag      TransVarianceCategory;
    typedef HxPixOpNPhaseTag        PhaseCategory;

                        PixOpT(HxTagList&, int w, int h, int d);

    void                doIt(const ArithT& v, int x, int y, int z);

    int                 nrPhases() const;
    void                init(int phase);
    void                done(int phase);

    static HxString     className();
};
```

3.8.12 Translation invariant, n phase pixel import operation.

Pseudo code of the operation:

```
InPixOp(Out, PixOp)
{
    for p = 1 to PixOp.nrPhases();
    {
        PixOp.init(p)
        foreach o in Out
            Out(o) = PixOp();
        PixOp.done(p)
    }
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpInTag            DirectionCategory;
    typedef HxPixOpTransInVarTag    TransVarianceCategory;
    typedef HxPixOpNPhaseTag        PhaseCategory;

                        PixOpT(HxTagList&);

    ArithT              doIt();

    int                 nrPhases() const;
    void                init(int phase);
    void                done(int phase);

    static HxString     className();
};
```

3.8.13 Translation variant, n phase pixel import operation.

Pseudo code of the operation:

```
InPixOp(Out, PixOp)
{
    for p = 1 to PixOp.nrPhases();
    {
        PixOp.init(p)
        foreach o in Out
            Out(o) = PixOp(i.x, i.y, i.z);
        PixOp.done(p)
    }
}
```

The requirements on the PixOpT template parameter expressed as class definition are:

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxPixOpInTag            DirectionCategory;
    typedef HxPixOpTransVarTag      TransVarianceCategory;
    typedef HxPixOpNPhaseTag        PhaseCategory;

                        PixOpT(HxTagList&, int w, int h, int d);

    ArithT              doIt(int x, int y, int z);

    int                 nrPhases() const;
    void                init(int phase);
    void                done(int phase);

    static HxString     className();
};
```

3.8.14 Display operation for 2D images

The requirements on the RgbT template parameter expressed as a class definition are:

```
template<class ValT, class ValDoubleT>
class RgbT
{
public:
                        RgbT(HxTagList& tags);

    int                 doIt(const ValT& pixV);

    int                 doItDouble(const ValDoubleT& pixV);

    static HxString     className();
};
```

Depending on the display parameters set by the GUI the function doIt will be called with 1 parameter of type ImgSigT::ArithType *or* the doItDouble function will be called with 1 parameter of type ImgSigT::ArithTypeDouble.

3.8.15 Display operation for 3D images

The requirements on the RgbT template parameter expressed as a class definition are:

```
template<class ValT, class ValDoubleT>
class RgbT
{
public:
                        RgbT(HxTagList& tags);

    int                 doIt(const ValT& pixV);

    int                 doItDouble(const ValDoubleT& pixV);

    static HxString     className();
};
```

Depending on the display parameters set by the GUI the function doIt will be called with 1 parameter of type ImgSigT::ArithType OR the doItDouble function will be called with 1 parameter of type ImgSigT::ArithTypeDouble.

3.8.16 2 dimensional, translation invariant, 1 phase, coordinate enumerated neighbourhood operation.

The requirements on the NgbT template parameter expressed as class definition are:

```
template<class ArithT>
class NgbT
{
public:

    typedef HxNgbCnumTag        IteratorCategory;
    typedef HxNgb1PhaseTag      PhaseCategory;
    typedef HxNgbTransInVarTag  TransVarianceCategory;

    typedef HxCnum              CnumType;

                                NgbT(HxTagList& tags);
                                ~NgbT();
```

```
    HxSizes                     size();

    CnumType                    begin();
    CnumType                    end();

    void                        init(ArithT value);
    void                        next(int x, int y, ArithT value);
    ArithT                      result() const;

    static HxString             className();
};
```

The Requirements on CnumType expressed as a class definition are:

```
class CnumType
{
public:
                                CnumType();
                                CnumType(const CnumType& rhs);
    CnumType&                   operator=(const CnumType& rhs);
    int                         x();
    int                         y();
    int                         z();
    void                        inc();
    bool                        operator!=(const CnumType& rhs);
};
```

3.8.17 2 dimensional, translation invariant, 1 phase, loop neighbourhood operation.

The requirements on the NgbT template parameter expressed as class definition are:

```
template<class ArithT>
class NgbT
{
public:

    typedef HxNgbLoopTag        IteratorCategory;
    typedef HxNgb1PhaseTag      PhaseCategory;
    typedef HxNgbTransInVarTag  TransVarianceCategory;

                                NgbT(HxTagList& tags);
                                ~NgbT();

    HxSizes                     size();
    void                        init();

    void                        next(int x, int y, ArithT value);
    ArithT                      result() const;

    static HxString             className();
};
```

3.8.18 2 dimensional, translation invariant, 2 phase, loop neighbourhood operation.

The requirements on the NgbT template parameter expressed as class definition are:

```
template<class ArithT>
class NgbT
{
```

```
public:

    typedef HxNgbLoopTag        IteratorCategory;
    typedef HxNgb2PhaseTag      PhaseCategory;
    typedef HxNgbTransInVarTag  TransVarianceCategory;

                                NgbT(HxTagList& tags);
                                ~NgbT();

    HxSizes                     size();

    void                        init();
    void                        init2();

    void                        next(int x, int y, ArithT value);
    void                        next2(int x, int y, ArithT value);

    ArithT                      result() const;

    static HxString             className();
};
```

3.8.19 2 dimensional, translation invariant, 1 phase, loop neighbourhood operation with kernel.

The requirements on the NgbT template parameter expressed as class definition are:

```
template<class ArithT>
class NgbT
{
public:

    typedef HxNgbLoopTag        IteratorCategory;
    typedef HxNgb1PhaseTag      PhaseCategory;
    typedef HxNgbTransInVarTag  TransVarianceCategory;

                                NgbT(HxTagList& tags);
                                ~NgbT();

    HxSizes                     size();
    void                        init();

    void                        next(int x, int y, ArithT value, ArithT mask);
    ArithT                      result() const;

    static HxString             className();
};
```

3.8.20 2 dimensional, translation invariant, 2 phase, loop neighbourhood operation using a kernel.

```
template<class ArithT>
class NgbT
{
public:

    typedef HxNgbLoopTag        IteratorCategory;
    typedef HxNgb2PhaseTag      PhaseCategory;
    typedef HxNgbTransInVarTag  TransVarianceCategory;
```

```
                              NgbT(HxTagList& tags);
                              ~NgbT();

    HxSizes                   size();

    void                      init();
    void                      init2();

    void                      next(int x, int y, ArithT value, ArithT mask);
    void                      next2(int x, int y, ArithT value, ArithT mask);

    ArithT                    result() const;

    static HxString           className();
};
```

Chapter 4

Geometry

4.1 BSplines

4.1.1 Terminology

4.1.1.1 Parametric and Sampled Curves

Parametric and *sampled* curves represent ordered sets of points in 2-D or 3-D space.

In the case of a *parametric* curve, it is possible to navigate along all its points with a path parameter defined in a continuous interval. We use $C(t)$ to denote a point in such a curve. We call *path parameter domain* the values which can be assumed by the parameter t.

In the case of a *sampled* curve only some points are represented and they are considered as linked with straight lines. We use $c[j]$ to denote a vertex in such curve, j being the *index* in a vector of sampled points.

Parametric and sampled curves may be open or closed.

One type of parametric curve is called piecewise polynomial curve. In this case each segment of the overall curve is given by two functions (or three, in 3D) which are polynomials of degree d of the path parameter t.

Ex: polynomials for a 2D cubic piecewise polynomial parametric curve:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

A list of real values called *knots* define the space for the path parameter t.

The coefficients a, b, c, d are computed from given information (ex: control points) using a specific algorithm (or model).

4.1.1.2 Splines

Splines are parametric piecewise polynomial curves defined with a model (the recipe) and control parameters (the ingredients). They can be implemented with different methods to construct different types of curves. Examples: interpolating spline curves, B-spline curves, Beta-spline curves, NURBS, etc.

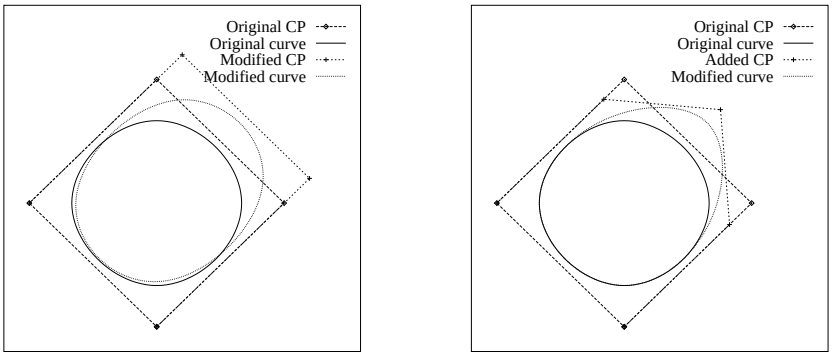


Figure 4.1: Examples of closed cubic B-Spline curves (degree=3). *Left:* Curve with four control points and uniformly spaced knots. Manipulation of one control point affects the whole curve. *Right:* A control point is added (knots become unevenly spaced). Manipulation of one control point affects only a curve interval.

4.1.1.3 B-Spline curves

We are actually interested in *B-spline curves* (see Figure 4.1), which are *splines* implemented as a linear combination of *basis functions* (see Figure 4.2).

We adopt the following terminology and notation:

- degree: degree of the polynomials (= order -1);
- type of curve: determine the usage of control points and knots at ends. In closed curves, knots are used circularly. There are several options for open curves, which we consider different types of curves (ex. with loose ends or with repeated end points);
- knots: ordered list of values that define the path parameter values for navigation along the curve. The values of the knots may be initialized in different ways: uniformly or non-uniformly distributed; generated automatically or explicitly informed by the programmer.
- interval: piece of a curve determined by two path positions t_1, t_2.
- control points (P): geometry information used to instantiate a given B-spline model as a curve in space. The number of control points is dependent on the model, that is, the degree, number of knots, and the curve type.
- basis functions (B): weight of control points for the computation of curve points along the path. Basis functions are completely defined by the knots and the degree of the curve. We refer to these as $B(i)$.
- curve points (C): points in the parametric curve at hand. In this implementation, curve points are never stored in the class, but generated upon request. In continuous curves, we refer to these as $C(t)$. In sampled curves, we refer to curve points as $C(j)$, where j is the index corresponding to a point with fixed t.
- derivatives (d): all derivatives are computed for the path parameter t. We refer to these as $dC(t, order)$, $dB(i, t, order)$, etc.

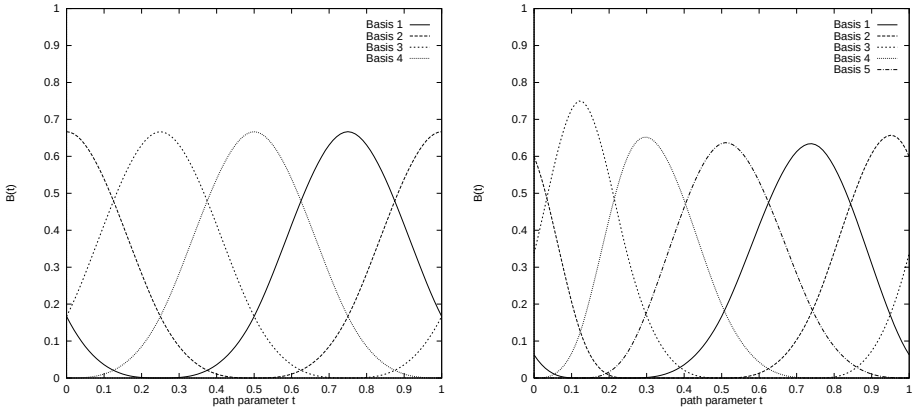


Figure 4.2: Basis functions for the curves in Figure 4.1. *Left:* Uniformly spaced knots. *Right:* Unevenly spaced knots.

4.1.2 Formulae

Introductory sources of information about B-splines are [**?**] and [**?**], which we used as we used as an intermediate step. Later we adopted [**?**] and [**?**]. The code follows the algorithm and notation as indicated in the comments. Below we shortly present the most important formulae used in its implementation.

Keep in mind the following notation:

- d = degree
- o = order $(d + 1)$
- λ_l = l^{th} knot in the sequence
- t = path parameter
- P_i = i^{th} control point
- X_i, Y_i = x and y coordinates of control point P_i
- $C(t)$ = curve point at t, (x, y) coordinates
- $C^{(n)}(t)$ = n^{th} derivative of curve at path path parameter t, (x, y) coordinates
- $P_i^{(n)}(t)$ = contribution of i^{th} control point for the computation of the n^{th} derivative at path parameter t
- $B_{i,o}(t)$ = value of i^{th} basis function of order o at path parameter t
- $B_{i,o}^{(n)}(t)$ = n^{th} derivative of i^{th} basis function of order o at path parameter t

A curve point $C(t)$, $t \in [\lambda_l, \lambda_{l+1})$, is evaluated as follows:

$$x(t) = \sum_{i=l-d}^{l} X_i B_{i,d+1}(t)$$
$$y(t) = \sum_{i=l-d}^{l} Y_i B_{i,d+1}(t) \qquad (4.1)$$

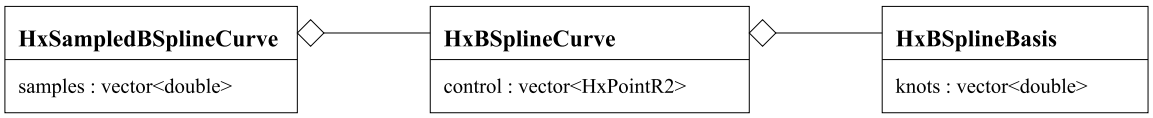


Figure 4.3: BSpline curve classes

The value of a i^{th} basis function of order o at position t - $B_{i,o}(t)$, $t \in [\lambda_i, \lambda_{i+1})$ - is computed with the recursive method proposed by de Boor (see equation 1.24 in [**?**]):

$$\begin{aligned} B_{i,o}(t) &= \frac{t-\lambda_i}{\lambda_{i+o-1}-\lambda_i} B_{i,o-1}(t) + \frac{\lambda_{i+o}-t}{\lambda_{i+o}-\lambda_{i+1}}, B_{i+1,o-1}(t) \\ B_{i,1}(t) &= \begin{cases} 1, & \text{if } t \in [\lambda_i, \lambda_{i+1}), \\ 0, & \text{if } t \notin [\lambda_i, \lambda_{i+1}). \end{cases} \end{aligned} \tag{4.2}$$

The n^{th} derivative of the curve at the path parameter t - $t \in [\lambda_l, \lambda_{l+1})$, is also computed with recursion (see equations 1.39 and 1.40 in [**?**]):

$$C^{(n)}(t) = \prod_{i=1}^{n}(d+1-l) \sum_{i=l-d+n}^{i} P_i^{(n)} B_{i,d+1-n}(t) \tag{4.3}$$

with

$$P_i^{(n)} = \begin{cases} P_i, & \text{if } n = 0, \\ \frac{P_i^{(n-1)} - P_{i-1}^{(n-1)}}{\lambda_{l+d+1-n} - \lambda_l}, & \text{if } n > 0. \end{cases} \tag{4.4}$$

The value of the n^{th} derivative of the i^{th} basis function of order o the path parameter t, $t \in [\lambda_i, \lambda_{i+1})$, is derived from equation 1.25 in [**?**] and computed as follows:

$$\begin{aligned} B_{i,o}^{(n)}(t) &= (o-1)\left\{ \frac{B_{i,o-1}^{(n-1)}(t)}{\lambda_{i+o-1}-\lambda_i} - \frac{B_{i+1,o-1}^{(n-1)}(t)}{\lambda_{i+o}-\lambda_{i+1}} \right\} \\ B_{i,o}^{(0)}(t) &= B_{i,o}(t) \end{aligned} \tag{4.5}$$

4.1.3 Class definition

The class design for the implementation is given in Figure 4.3.

4.1.3.1 `HxBSplineBasis`

Class to represent the B-Spline basis functions, which correspond to the “model.”

Internal Representation:

- degree: integer, ≥ 1.
- type of curve: closed, open (the ends are loose) or openRepeatEndPoints (the end points are repeated, and the curve passes through them).

- range of path parameter: real, interval $t \in [minT, maxT)$.
- knots: vector of increasing real numbers. The knots that define the curve intervals are in the range $[minT, maxT]$. Extra knots are added to cope with end point conditions for closed or open curves.

4.1.3.2 `HxBSplineCurve`

A `HxBSplineCurve` results from the association of a `HxBSplineBasis` with a vector of control points used to instantiate it in 2-D.

Internal Representation:

- `HxBSplineBasis`: determines the curve model.
- vector of control points: instantiate curve in 2-D.

This curve is called *continuous* because its manipulation is valid for any value of $t \in [minT, maxT)$.

4.1.3.3 `HxSampledBSplineCurve`

A `HxSampledBSplineCurve` consists of a `HxBSplineCurve` and a vector of t values indicating the path positions where the continuous curve is sampled. This is meant to facilitate the manipulation of sampled curves. Potentially, the implementation of a sampled curve allows for the pre-computation of values to increase performance (e.g. curve points and basis). The current implementation does not take advantage of this.

Internal Representation:

- `HxBSplineCurve`: the continuous curve.
- vector with sampled path parameter positions.

Chapter 5

Global Image Functions Reference

5.1 HxAbs.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAbs** (**HxImageRep** im)

  Absolute value.

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 HxImageRep L_HXIMAGEREP HxAbs (HxImageRep *im*)

Absolute value.

The function computes the absolute value (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Instantiations : **Abs instantiations** (p. **??**)

(obsolete) **Implementation specifics** : The pixel functor : **HxUpoAbs** (p. 719). The image functor instantiator : **HxInstantiatorAbs** (p. 497).

```
13 {
14     return im.unaryPixOp("abs");
15 }
```

5.2 HxAcos.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAcos** (**HxImageRep** im)

  Arc cosine.

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 HxImageRep L_HXIMAGEREP HxAcos (HxImageRep *im*)

Arc cosine.

The function computes the arc cosine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoAcos** (p. 720). The image functor instantiator : **HxInstantiatorAcos** (p. 498).

```
13 {
14     return im.unaryPixOp("acos");
15 }
```

5.3 HxAdd.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAdd** (**HxImageRep** im1, **HxImageRep** im2)

  Addition.

5.3.1 Detailed Description

5.3.2 Function Documentation

5.3.2.1 HxImageRep L_HXIMAGEREP HxAdd (HxImageRep *im1*, HxImageRep *im2*)

Addition.

The function performs addition (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoAdd** (p. 169). The image functor instantiator : **HxInstantiatorAdd** (p. 498).

```
14 {
15     return im1.binaryPixOp(im2, "add");
16 }
```

5.4 HxAddVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddVal** (**HxImageRep** im, **HxValue** val)
  Addition.

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 HxImageRep L_HXIMAGEREP HxAddVal (HxImageRep *im*, HxValue *val*)

Addition.

The function performs addition (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoAdd** (p. 169). The image functor instantiator : **HxInstantiatorAddV** (p. 499).

```
14 {
15     return im.binaryPixOp(val, "add");
16 }
```

5.5 HxAffinePix.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAffinePix** (**HxImageRep** im, **HxValue** v1, **HxValue** v2, **HxValue** v3)

5.5.1 Detailed Description

5.6 HxAnd.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAnd** (**HxImageRep** im1, **HxImageRep** im2)
  And.

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 HxImageRep L_HXIMAGEREP HxAnd (HxImageRep *im1*, HxImageRep *im2*)

And.

The function performs and (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoAnd** (p. 172). The image functor instantiator : **HxInstantiatorAnd** (p. 500).

```
13 {
14     return im1.binaryPixOp(im2, "and");
15 }
```

5.7 HxAndVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAndVal** (**HxImageRep** im, **HxValue** val)

  And.

5.7.1 Detailed Description

5.7.2 Function Documentation

5.7.2.1 HxImageRep L_HXIMAGEREP HxAndVal (HxImageRep *im*, HxValue *val*)

And.

The function performs and (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoAnd** (p. 172). The image functor instantiator : **HxInstantiatorAndV** (p. 501).

```
13 {
14     return im.binaryPixOp(val, "and");
15 }
```

5.8 HxArg.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxArg** (**HxImageRep** im)

  Argument.

5.8.1 Detailed Description

5.8.2 Function Documentation

5.8.2.1 HxImageRep L_HXIMAGEREP HxArg (HxImageRep *im*)

Argument.

The function computes the complex argument (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoArg** (p. 722). The image functor instantiator : **HxInstantiatorArg** (p. 501).

```
13 {
14     return im.unaryPixOp("arg");
15 }
```

5.9 HxAsin.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAsin** (**HxImageRep** im)

  Arc sine.

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 HxImageRep L_HXIMAGEREP HxAsin (HxImageRep *im*)

Arc sine.

The function computes the arc sine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoAsin** (p. 723). The image functor instantiator : **HxInstantiatorAsin** (p. 502).

```
13 {
14     return im.unaryPixOp("asin");
15 }
```

5.10 HxAtan.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAtan** (**HxImageRep** im)
  Arc tangent.

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 HxImageRep L_HXIMAGEREP HxAtan (HxImageRep *im*)

Arc tangent.

The function computes the arc tangent (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoAtan** (p. 724). The image functor instantiator : **HxInstantiatorAtan** (p. 503).

```
13 {
14     return im.unaryPixOp("atan");
15 }
```

5.11 HxAtan2.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAtan2** (**HxImageRep** im)
  Arc tangent (use first and second pixel dimension).

5.11.1 Detailed Description

5.11.2 Function Documentation

5.11.2.1 HxImageRep L_HXIMAGEREP HxAtan2 (HxImageRep *im*)

Arc tangent (use first and second pixel dimension).

The function computes the arc tangent (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoAtan2** (p. 725). The image functor instantiator : **HxInstantiatorAtan2** (p. 503).

```
13 {
14     return im.unaryPixOp("atan2");
15 }
```

5.12 HxCannyEdgeMap.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCannyEdgeMap** (**HxImageRep** img, double sigma)

  Computes the Canny edge map of a scalar image.

5.12.1 Detailed Description

5.12.2 Function Documentation

5.12.2.1 HxImageRep L_HXIMAGEREP HxCannyEdgeMap (HxImageRep *img*, double *sigma*)

Computes the Canny edge map of a scalar image.

The result is a vector image.

```
16 {
17     int minSize = HxImageMinSize(img);
18
19     HxImageRep gauss0 = HxMakeGaussian1d(sigma, 0, 4.0, minSize);
20     HxImageRep gauss1 = HxMakeGaussian1d(sigma, 1, 4.0, minSize);
21
22     HxImageRep Ix = img.genConvSeparated(1, gauss1, gauss0, "mul", "addAssign",
23                                          HxImageRep::ARITH_PREC);
24     HxImageRep Iy = img.genConvSeparated(1, gauss0, gauss1, "mul", "addAssign",
25                                          HxImageRep::ARITH_PREC);
26
27     return HxMakeFrom2Images(Ix, Iy);
28 }
```

5.13 HxCannyThreshold.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCannyThreshold** (**HxImageRep** img, double sigma, double level)

  Computes the Canny edge map of a scalar image, performs non-maxima suppression, and tresholds the norm of the resulting vector field at the given level.

5.13.1 Detailed Description

5.13.2 Function Documentation

5.13.2.1 HxImageRep L_HXIMAGEREP HxCannyThreshold (HxImageRep *img*, double *sigma*, double *level*)

Computes the Canny edge map of a scalar image, performs non-maxima suppression, and tresholds the norm of the resulting vector field at the given level.

```
17 {
18     HxImageRep edges = HxCannyEdgeMap(img, sigma);
19     HxImageRep nonmax = HxNonMaxSuppressionGradDir(edges);
20     return HxThreshold(HxNorm2Sqr(nonmax), HxValue(level * level));
21 }
```

5.14 HxCannyThresholdAlt.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCannyThresholdAlt** (**HxImageRep** img, double sigma, double level)

  Computes the Canny edge map of a scalar image, performs non-maxima suppression, and tresholds the norm of the resulting vector field at the given level (alternative implementation).

5.14.1 Detailed Description

5.14.2 Function Documentation

5.14.2.1 HxImageRep L_HXIMAGEREP HxCannyThresholdAlt (HxImageRep *img*, double *sigma*, double *level*)

Computes the Canny edge map of a scalar image, performs non-maxima suppression, and tresholds the norm of the resulting vector field at the given level (alternative implementation).

```
14 {
15     HxImageRep edges = HxCannyEdgeMap(img, sigma);
16     HxTagList tags;
17     HxAddTag(tags, "level", HxValue(level));
18     return edges.neighbourhoodOp("isMaxGradDir", tags);
19 }
```

5.15 HxCeil.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCeil** (**HxImageRep** im)
  Ceiling.

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 HxImageRep L_HXIMAGEREP HxCeil (HxImageRep *im*)

Ceiling.

The function computes the ceiling (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoCeil** (p. 726). The image functor instantiator : **HxInstantiatorCeil** (p. 504).

```
13 {
14     return im.unaryPixOp("ceil");
15 }
```

5.16 HxColConvert.h File Reference

```
#include "HxVec3Double.h"
```

Functions

- L_HXBASIS **HxVec3Double HxColRGB2CMY** (const **HxVec3Double** &v)
  Conversion from RGB to CMY.
- L_HXBASIS **HxVec3Double HxColCMY2RGB** (const **HxVec3Double** &v)
  Conversion from CMY to RGB.
- L_HXBASIS **HxVec3Double HxColRGB2XYZ** (const **HxVec3Double** &v)
  Conversion from RGB(Rec709) to XYZ (1931).
- L_HXBASIS **HxVec3Double HxColXYZ2RGB** (const **HxVec3Double** &v)
  Conversion from XYZ (1931) to RGB(Rec709).
- L_HXBASIS **HxVec3Double HxColCMY2XYZ** (const **HxVec3Double** &v)
  Conversion from CMY through RGB(Rec709) to XYZ (1931).
- L_HXBASIS **HxVec3Double HxColXYZ2CMY** (const **HxVec3Double** &v)
  Conversion from XYZ (1931) through RGB(Rec709) to CMY.
- L_HXBASIS **HxVec3Double HxColLab2XYZ** (const **HxVec3Double** &v)
  Conversion from Lab to XYZ (1931), D65 reference white point.

- L_HXBASIS **HxVec3Double HxColXYZ2Lab** (const **HxVec3Double** &v)

  Conversion from XYZ (1931) to Lab, D65 reference white point.

- L_HXBASIS **HxVec3Double HxColLuv2XYZ** (const **HxVec3Double** &v)

  Conversion from Luv to XYZ (1931), D65 reference white point.

- L_HXBASIS **HxVec3Double HxColXYZ2Luv** (const **HxVec3Double** &v)

  Conversion from XYZ (1931) to Luv, D65 reference white point.

- L_HXBASIS **HxVec3Double HxColRGB2OOO** (const **HxVec3Double** &v)

  Conversion from RGB(Rec709) to OOO (Geusebroek Thesis).

- L_HXBASIS **HxVec3Double HxColOOO2RGB** (const **HxVec3Double** &v)

  Conversion from OOO (Geusebroek Thesis) to RGB(Rec709).

- L_HXBASIS **HxVec3Double HxColXYZ2OOO** (const **HxVec3Double** &v)

  Conversion from XYZ(1931) to OOO (Geusebroek Thesis).

- L_HXBASIS **HxVec3Double HxColOOO2XYZ** (const **HxVec3Double** &v)

  Conversion from OOO (Geusebroek Thesis) to XYZ(1931).

- L_HXBASIS **HxVec3Double HxColRGB2HSI** (const **HxVec3Double** &v)

  Conversion from RGB to HSI.

- L_HXBASIS **HxVec3Double HxColHSI2RGB** (const **HxVec3Double** &v)

  Conversion from HSI to RGB.

- L_HXBASIS int **HxColRGB2int** (const **HxVec3Double** &v)

  Convert RGB **HxVec3Double** (p. 804) *to an ARGB integer representation.*

- L_HXBASIS int **HxColRGB2int** (const **HxVec3Int** &v)

  Convert RGB **HxVec3Int** (p. 824) *to an ARGB integer representation.*

5.16.1 Detailed Description

5.16.2 Function Documentation

5.16.2.1 L_HXBASIS HxVec3Double HxColRGB2CMY (const HxVec3Double & *v*)

Conversion from RGB to CMY.

```
20 {
21     return HxVec3Double(1, 1, 1) - v;
22 }
```

5.16.2.2 L_HXBASIS HxVec3Double HxColCMY2RGB (const HxVec3Double & *v*)

Conversion from CMY to RGB.

```
26 {
27     return HxVec3Double(1, 1, 1) - v;
28 }
```

5.16.2.3 L_HXBASIS HxVec3Double HxColRGB2XYZ (const HxVec3Double & *v*)

Conversion from RGB(Rec709) to XYZ (1931).

```
34 {
35     // Using Rec709
36     // X = 100 * (0.412452 R/255 + 0.357580 G/255 + 0.180423 B/255)
37     // Y = 100 * (0.212671 R/255 + 0.715160 G/255 + 0.072169 B/255)
38     // Z = 100 * (0.019334 R/255 + 0.119193 G/255 + 0.950227 B/255)
39
40     double X = v.x() * 0.161746 + v.y() * 0.140227 + v.z() * 0.070754;
41     double Y = v.x() * 0.083400 + v.y() * 0.280455 + v.z() * 0.028301;
42     double Z = v.x() * 0.007582 + v.y() * 0.046742 + v.z() * 0.372638;
43
44     return HxVec3Double(X, Y, Z);
45 }
```

5.16.2.4 L_HXBASIS HxVec3Double HxColXYZ2RGB (const HxVec3Double & *v*)

Conversion from XYZ (1931) to RGB(Rec709).

```
49 {
50     // Using rec709
51     // R = 255 * (3.240479 X/100 - 1.537150 * Y/100 - 0.498535 * Z/100)
52     // G = 255 * (-0.969256 * X/100 + 1.875992 * Y/100 + 0.041556 * Z/100)
53     // B = 255 * (0.055648 * Z/100 - 0.204043 * Y/100 + 1.057311 * Z/100)
54
55     double R = v.x() * 8.25322145 + v.y() * -3.9197325 + v.z() * -1.27126425;
56     double G = v.x() * -2.4716028 + v.y() * 4.7837796 + v.z() * 0.1059678;
57     double B = v.x() * 0.1419024 + v.y() * -0.52030965 + v.z() * 2.69614305;
58
59     return HxVec3Double(R, G, B);
60 }
```

5.16.2.5 L_HXBASIS HxVec3Double HxColCMY2XYZ (const HxVec3Double & *v*)

Conversion from CMY through RGB(Rec709) to XYZ (1931).

```
66 {
67     // Using rec709
68     // R = 1-C, G = 1-M, B = 1-Y
69
70     double X = 95.045385
71         - v.x() * 0.161746 - v.y() * 0.140227 - v.z() * 0.070754;
72     double Y = 100.0
73         - v.x() * 0.083400 - v.y() * 0.280455 - v.z() * 0.028301;
```

```
74     double Z = 108.87531
75         - v.x() * 0.007582 - v.y() * 0.046742 - v.z() * 0.372638;
76
77     return HxVec3Double(X, Y, Z);
78 }
```

5.16.2.6 L_HXBASIS HxVec3Double HxColXYZ2CMY (const HxVec3Double & *v*)

Conversion from XYZ (1931) through RGB(Rec709) to CMY.

```
82 {
83     // Using rec709
84     // C = 1-R, M = 1-G, Y = 1-B
85
86     double C = 255.0
87         - v.x() * 8.25322145 - v.y() * -3.9197325 - v.z() * -1.27126425;
88     double M = 255.0
89         - v.x() * -2.4716028 - v.y() * 4.7837796 - v.z() * 0.1059678;
90     double Y = 255.0
91         - v.x() * 0.1419024 - v.y() * -0.52030965 - v.z() * 2.69614305;
92
93     return HxVec3Double(C, M, Y);
94 }
```

5.16.2.7 L_HXBASIS HxVec3Double HxColLab2XYZ (const HxVec3Double & *v*)

Conversion from Lab to XYZ (1931), D65 reference white point.

```
128 {
129     double Y = L2Y(v.x());
130
131     double Ltmp = (v.x() + 16.0) / 116.0; // == f(Y / Yn)
132     double atmp = Ltmp + v.y() / 500.0;
133     double X = Xn * fInv4ab(atmp, Ltmp);
134     double btmp = Ltmp - v.z() / 200.0;
135     double Z = Zn * fInv4ab(btmp, Ltmp);
136     return HxVec3Double(X, Y, Z);
137 }
```

5.16.2.8 L_HXBASIS HxVec3Double HxColXYZ2Lab (const HxVec3Double & *v*)

Conversion from XYZ (1931) to Lab, D65 reference white point.

```
141 {
142     HxVec3Double n = v / HxVec3Double(Xn, Yn, Zn);
143
144     double L = Y2L(n.y());
145
146     double fnX = f4ab(n.x());
147     double fnY = f4ab(n.y());
148     double fnZ = f4ab(n.z());
149
150     double a = 500.0 * (fnX - fnY);
151     double b = 200.0 * (fnY - fnZ);
152     return HxVec3Double(L, a, b);
153 }
```

5.16.2.9 L_HXBASIS HxVec3Double HxColLuv2XYZ (const HxVec3Double & *v*)

Conversion from Luv to XYZ (1931), D65 reference white point.

```
161 {
162     double Y = L2Y(v.x());
163     double tmp = Xn + 15 * Yn + 3 * Zn;
164     double unp = 4 * Xn / tmp;
165     double vnp = 9 * Yn / tmp;
166     double Q = v.y() / (13 * v.x()) + unp;
167     double R = v.z() / (13 * v.x()) + vnp;
168     double A = 3 * Y * (5 * R - 3);
169     double Z = ((Q - 4) * A - 15 * Q * R * Y) / (12 * R);
170     double X = -(A / R + 3 * Z);
171     return HxVec3Double(X, Y, Z);
172 }
```

5.16.2.10 L_HXBASIS HxVec3Double HxColXYZ2Luv (const HxVec3Double & *v*)

Conversion from XYZ (1931) to Luv, D65 reference white point.

```
176 {
177     double L = Y2L(v.y() / Yn);
178     double tmp = Xn + 15 * Yn + 3 * Zn;
179     double unp = 4 * Xn / tmp;
180     double vnp = 9 * Yn / tmp;
181     tmp = v.x() + 15 * v.y() + 3 * v.z();
182     double up = 4 * v.x() / tmp;
183     double vp = 9 * v.y() / tmp;
184     double us = 13 * L * (up - unp);
185     double vs = 13 * L * (vp - vnp);
186     return HxVec3Double(L, us, vs);
187 }
```

5.16.2.11 L_HXBASIS HxVec3Double HxColRGB2OOO (const HxVec3Double & *v*)

Conversion from RGB(Rec709) to OOO (Geusebroek Thesis).

For transformation from RGB to XYZ, see HxColRGB2XYZ.

```
275 {
276     double E   = v.x() * 0.000233846 + v.y() * 0.00261968 + v.z() * 0.00127135;
277     double El  = v.x() * 0.000726333 + v.y() * 0.000718106+ v.z() * -0.00121377;
278     double Ell = v.x() * 0.000846833 + v.y() * -0.00173932+ v.z() * 0.000221515;
279
280     return HxVec3Double(E, El, Ell);
281 }
```

5.16.2.12 L_HXBASIS HxVec3Double HxColOOO2RGB (const HxVec3Double & *v*)

Conversion from OOO (Geusebroek Thesis) to RGB(Rec709).

For transformation from XYZ to RGB, see HxColXYZ2RGB.

```
287 {
288     double R = v.x() * 328.084 + v.y() * 469.182 + v.z() * 687.853;
289     double G = v.x() * 199.794 + v.y() * 172.242 + v.z() * -202.904;
290     double B = v.x() * 314.533 + v.y() * -441.212 + v.z() * 291.574;
291 
292     return HxVec3Double(R, G, B);
293 }
```

5.16.2.13 L_HXBASIS HxVec3Double HxColXYZ2OOO (const HxVec3Double & *v*)

Conversion from XYZ(1931) to OOO (Geusebroek Thesis).

```
251 {
252     double E   = v.x() *-0.004362 + v.y() * 0.010954 + v.z() * 0.003408;
253     double El  = v.x() * 0.004055 + v.y() * 0.001220 + v.z() *-0.004120;
254     double Ell = v.x() * 0.011328 + v.y() *-0.011755 + v.z() *-0.000664;
255 
256     return HxVec3Double(E, El, Ell);
257 }
```

5.16.2.14 L_HXBASIS HxVec3Double HxColOOO2XYZ (const HxVec3Double & *v*)

Conversion from OOO (Geusebroek Thesis) to XYZ(1931).

```
263 {
264     double X = v.x() * 103.337 + v.y() * 68.824 + v.z() * 103.435;
265     double Y = v.x() * 92.297  + v.y() * 74.949 + v.z() * 8.714;
266     double Z = v.x() * 129.034 + v.y() * -152.804 + v.z() * 104.383;
267 
268     return HxVec3Double(X, Y, Z);
269 }
```

5.16.2.15 L_HXBASIS HxVec3Double HxColRGB2HSI (const HxVec3Double & *v*)

Conversion from RGB to HSI.

No formula yet...

```
200 {
201     double I = v.sum().x() / 3.0;
202     double S = (I == 0.0) ? 1.0 : 1.0 - (v.min().x() / I);
203     double H;
204     if (v.x() == v.y() && v.y() == v.z())
205         H = 0.0;
206     else {
207         double tmp = acos((0.5*(v.x()-v.y()+v.x()-v.z())) /
208                 sqrt((v.x()-v.y())*(v.x()-v.y())+(v.x()-v.z())*(v.y()-v.z())));
209         H = (v.y() > v.z()) ? tmp : TWO_PI - tmp;
210     }
211     return HxVec3Double(H, S, I);
212 }
```

5.16.2.16 L_HXBASIS HxVec3Double HxColHSI2RGB (const HxVec3Double & *v*)

Conversion from HSI to RGB.

No formula yet...

```
216 {
217     double H = v.x();
218     double S = v.y();
219     double I = v.z();
220     double Htmp;
221     double R,B,G;
222 
223     if (H == 0.0)
224         R = G = B = I;
225     else if (H > 0.0 && H < TWO_PI_3 ) {
226         Htmp = 1 / sqrt(3.0) * tan(H - PI_3);
227         B = (1.0 - S) * I;
228         G = (1.5 + 1.5*Htmp ) * I - ((0.5 + 1.5*Htmp) * B);
229         R = 3.0 * I - G - B;
230     }
231     else if (H >= TWO_PI_3 && H < FOUR_PI_3) {
232         Htmp = 1 / sqrt(3.0) * tan(H - PI);
233         R = (1.0 - S) * I;
234         B = (1.5 + 1.5*Htmp ) * I - ((0.5 + 1.5*Htmp) * R);
235         G = 3.0 * I - B - R;
236     }
237     else {
238         Htmp = 1 / sqrt(3.0) * tan(H - FIVE_PI_3);
239         G = (1.0 - S) * I;
240         R = (1.5 + 1.5*Htmp ) * I - ((0.5 + 1.5*Htmp) * G);
241         B = 3.0 * I - R - G;
242     }
243     return HxVec3Double(R, G, B);
244 }
```

5.16.2.17 L_HXBASIS int HxColRGB2int (const HxVec3Double & *v*)

Convert RGB **HxVec3Double** (p. 804) to an ARGB integer representation.

5.16.2.18 L_HXBASIS int HxColRGB2int (const HxVec3Int & *v*)

Convert RGB **HxVec3Int** (p. 824) to an ARGB integer representation.

```
327 {
328     int x = v.x();
329     if (x < 0)
330         x = 0;
331     else {
332         if (x > 255)
333             x = 255;
334     }
335     int y = v.y();
336     if (y < 0)
337         y = 0;
338     else {
339         if (y > 255)
340             y = 255;
341     }
```

```
342     int z = v.z();
343     if (z < 0)
344         z = 0;
345     else {
346         if (z > 255)
347             z = 255;
348     }
349     return (255 << 24) | (x << 16) | (y << 8) | z;
350 }
```

5.17 HxColor.h File Reference

```
#include "HxIoFwd.h"
#include "HxString.h"
#include "HxVec3Double.h"
```

Compounds

- class **HxColor**

  Class definition color semantics.

Enumerations

- enum **HxColorModel** { **RGB**, **CMY**, **XYZ**, **Lab**, **Luv**, **OOO**, **HSI** }

  Supported color spaces.

Functions

- STD_OSTREAM & **operator**<< (STD_OSTREAM &os, const **HxColor** c)

5.17.1 Detailed Description

5.17.2 Enumeration Type Documentation

5.17.2.1 enum HxColorModel

Supported color spaces.

See also : **Color conversion functions** (p. 7).

```
22 { RGB, CMY, XYZ, Lab, Luv, OOO, HSI };
```

5.18 HxColorSpace.h File Reference

```
#include "HxColor.h"
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxColorSpace** (**HxImageRep** im, **HxColorModel** fromColorSpace, **HxColorModel** toColorSpace)

  Color space conversion.

5.18.1 Detailed Description

5.18.2 Function Documentation

5.18.2.1 HxImageRep L_HXIMAGEREP HxColorSpace (HxImageRep *im*, HxColorModel *fromColorSpace*, HxColorModel *toColorSpace*)

Color space conversion.

The function transforms the color model (see **Color conversion functions** (p. 7)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoColSpace** (p. 727). The image functor instantiator : **HxInstantiatorColSpace** (p. 504).

```
14 {
15     HxTagList tags;
16     HxAddTag<HxColorModel>(tags, "fromColorSpace", fromColorSpace);
17     HxAddTag<HxColorModel>(tags, "toColorSpace", toColorSpace);
18     return im.unaryPixOp("colorSpace", tags);
19 }
```

5.19 HxComplement.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxComplement** (**HxImageRep** im)

  One's complement.

5.19.1 Detailed Description

5.19.2 Function Documentation

5.19.2.1 HxImageRep L_HXIMAGEREP HxComplement (HxImageRep *im*)

One's complement.

The function computes the one's complement (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoComplement** (p. 729). The image functor instantiator : **HxInstantiatorComplement** (p. 505).

```
13 {
14     return im.unaryPixOp("complement");
15 }
```

5.20 HxConjugate.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConjugate** (**HxImageRep** im)
  Complex conjugate.

5.20.1 Detailed Description

5.20.2 Function Documentation

5.20.2.1 HxImageRep L_HXIMAGEREP HxConjugate (HxImageRep *im*)

Complex conjugate.

The function computes the complex conjugate (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoConjugate** (p. 730). The image functor instantiator : **HxInstantiatorConjugate** (p. 506).

```
13 {
14     return im.unaryPixOp("conjugate");
15 }
```

5.21 HxContrastStretch.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxContrastStretch** (**HxImageRep** img, double mFactor)
  Contrast stretching.

5.21.1 Detailed Description

5.21.2 Function Documentation

5.21.2.1 HxImageRep L_HXIMAGEREP HxContrastStretch (HxImageRep *img*, double *mFactor*)

Contrast stretching.

Computes mFactor * ((I - Imin) / (Imax - Imin)) with Imin and Imax the minimum and maximum value present in image I.

```
19 {
20     HxScalarDouble min = HxPixMin(img);
21     HxScalarDouble max = HxPixMax(img);
22     return HxMulVal(HxSubVal(img, min), mFactor / (max - min));
23 }
```

5.22 HxConvGauss2d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvGauss2d** (**HxImageRep** img, double sigmax, int orderDerivx, double accuracyx, double sigmay, int orderDerivy, double accuracyy)

  HxConvGauss2d.

5.22.1 Detailed Description

5.22.2 Function Documentation

5.22.2.1 HxImageRep L_HXIMAGEREP HxConvGauss2d (HxImageRep *img*, double *sigmax*, int *orderDerivx*, double *accuracyx*, double *sigmay*, int *orderDerivy*, double *accuracyy*)

HxConvGauss2d.

Equivalent to : img.genConvSeparated(1, gaussx, gaussy, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gaussx, gaussy are the 1d double-precision Gaussian kernels based on the respective sets of sigma, orderDeriv, accuracy parameters.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
18 {
19     HxImageRep gaussx, gaussy;
20 
21     gaussx = HxMakeGaussian1d(sigmax, orderDerivx, accuracyx,
22         img.dimensionSize(1));
23     gaussy = HxMakeGaussian1d(sigmay, orderDerivy, accuracyy,
24         img.dimensionSize(2));
25 
26     return img.genConvSeparated(
27         1, gaussx, gaussy, "mul", "addAssign", HxImageRep::ARITH_PREC);
28 }
```

5.23 HxConvGauss3d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvGauss3d** (**HxImageRep** img, double sigmax, int orderDerivx, double accuracyx, double sigmay, int orderDerivy, double accuracyy, double sigmaz, int orderDerivz, double accuracyz)

  HxConvGauss3d.

5.23.1 Detailed Description

5.23.2 Function Documentation

5.23.2.1 HxImageRep L_HXIMAGEREP HxConvGauss3d (HxImageRep *img*, double *sigmax*, int *orderDerivx*, double *accuracyx*, double *sigmay*, int *orderDerivy*, double *accuracyy*, double *sigmaz*, int *orderDerivz*, double *accuracyz*)

HxConvGauss3d.

Equivalent to (3x): img.generalizedConvolutionK1d(dim, gauss[dim], "mul", "addAssign", HxImageRep::ARITH_PREC)

where gaussx, gaussy, gaussz are the 1d double-precision Gaussian kernels based on the respective sets of sigma, orderDeriv, accuracy parameters.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
19 {
20     HxImageRep gaussx, gaussy, gaussz;
21
22     gaussx = HxMakeGaussian1d(sigmax, orderDerivx, accuracyx,
23         img.dimensionSize(1));
24     gaussy = HxMakeGaussian1d(sigmay, orderDerivy, accuracyy,
25         img.dimensionSize(2));
26     gaussz = HxMakeGaussian1d(sigmaz, orderDerivz, accuracyz,
27         img.dimensionSize(3));
28
29     HxImageRep res = img.generalizedConvolutionK1d(1, gaussx,
30         "mul", "addAssign", HxImageRep::ARITH_PREC);
31     res = res.generalizedConvolutionK1d(2, gaussy,
32         "mul", "addAssign", HxImageRep::ARITH_PREC);
33     return res.generalizedConvolutionK1d(3, gaussz,
34         "mul", "addAssign", HxImageRep::ARITH_PREC);
35 }
```

5.24 HxConvKernelSeparated.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvKernelSeparated** (**HxImageRep** img, **HxImageRep** kernel, **HxImageRep::ResultPrecision** resPrec)

  Convolution with separable kernel.

5.24.1 Detailed Description

5.24.2 Function Documentation

5.24.2.1 HxImageRep L_HXIMAGEREP HxConvKernelSeparated (HxImageRep *img*, HxImageRep *kernel*, HxImageRep::ResultPrecision *resPrec*)

Convolution with separable kernel.

The function performs a convolution on the input image via a generalized convolution operation (see **Images** (p. 9)). The same kernel is applied in each dimension.

Implementation specifics : The image functor instantiator for 2D images : **HxInstMulAddAss2dK1d** (p. 551), and for 3D images : **HxInstMulAddAss3dK1d** (p. 553).

```
15 {
16     return img.genConvSeparated(kernel, "mul", "addAssign", resPrec);
17 }
```

5.25 HxConvKernelSeparated2d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvKernelSeparated2d** (**HxImageRep** img, **HxImageRep** kernelX, **HxImageRep** kernelY, **HxImageRep::ResultPrecision** resPrec=HxImageRep::DEFAULT_PREC)

  Convolution with separable kernel on a 2D image.

- **HxImageRep** L_HXIMAGEREP **HxConvKernelSeparated2d** (**HxImageRep** img, **HxImageRep** kernelX, **HxImageRep** kernelY, **HxImageRep::ResultPrecision** resPrec, **HxTagList** &tags)

5.25.1 Detailed Description

5.25.2 Function Documentation

5.25.2.1 HxImageRep L_HXIMAGEREP HxConvKernelSeparated2d (HxImageRep *img*, HxImageRep *kernelX*, HxImageRep *kernelY*, HxImageRep::ResultPrecision *resPrec* = HxImageRep::DEFAULT_PREC)

Convolution with separable kernel on a 2D image.

The function performs a convolution on the input image via a generalized convolution operation (see **Images** (p. 9)). KernelX is applied in the first dimension, kernelY in the second dimension.

Implementation specifics : The image functor instantiator : **HxInstMulAddAss2dK1d** (p. 551).

5.26 HxConvolution.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvolution** (**HxImageRep** img, **HxImageRep** kernel, **HxImageRep::ResultPrecision** resPrec)

  Convolution.

5.26.1 Detailed Description

5.26.2 Function Documentation

5.26.2.1 HxImageRep L_HXIMAGEREP HxConvolution (HxImageRep *img*, HxImageRep *kernel*, HxImageRep::ResultPrecision *resPrec*)

Convolution.

The function performs a convolution on the input image via a generalized convolution operation (see **Images** (p. 9)).

Implementation specifics : The image functor instantiator for 2D images: **HxInstMulAddAss2d** (p. 551), and for 3D images : **HxInstMulAddAss3d** (p. 552).

```
15 {
16     return img.generalizedConvolution(
17             kernel, "mul", "addAssign", resPrec);
18 }
```

5.27 HxConvolution1d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConvolution1d** (**HxImageRep** img, **HxImageRep** kernel, int dimension, **HxImageRep::ResultPrecision** resPrec)

  Convolution in one dimension.

5.27.1 Detailed Description

5.27.2 Function Documentation

5.27.2.1 HxImageRep L_HXIMAGEREP HxConvolution1d (HxImageRep *img*, HxImageRep *kernel*, int *dimension*, HxImageRep::ResultPrecision *resPrec*)

Convolution in one dimension.

The function performs a convolution on the input image in the given dimension via a generalized convolution operation (see **Images** (p. 9)).

Implementation specifics : The image functor instantiator for 2D images : **HxInstMulAddAss2dK1d** (p. 551), and for 3D images : **HxInstMulAddAss3dK1d** (p. 553).

```
16 {
17     return img.generalizedConvolutionK1d(
18             dimension, kernel, "mul", "addAssign", resPrec);
19 }
```

5.28 HxCos.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCos** (**HxImageRep** im)

  Cosine.

5.28.1 Detailed Description

5.28.2 Function Documentation

5.28.2.1 HxImageRep L_HXIMAGEREP HxCos (HxImageRep *im*)

Cosine.

The function computes the cosine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoCos** (p. 731). The image functor instantiator : **HxInstantiatorCos** (p. 506).

```
13 {
14     return im.unaryPixOp("cos");
15 }
```

5.29 HxCosh.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCosh** (**HxImageRep** im)

  Hyperbolic cosine.

5.29.1 Detailed Description

5.29.2 Function Documentation

5.29.2.1 HxImageRep L_HXIMAGEREP HxCosh (HxImageRep *im*)

Hyperbolic cosine.

The function computes the hyperbolic cosine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoCosh** (p. 732). The image functor instantiator : **HxInstantiatorCosh** (p. 507).

```
13 {
14     return im.unaryPixOp("cosh");
15 }
```

5.30 HxCross.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCross** (**HxImageRep** im1, **HxImageRep** im2)

  Cross product.

5.30.1 Detailed Description

5.30.2 Function Documentation

5.30.2.1 HxImageRep L_HXIMAGEREP HxCross (HxImageRep *im1*, HxImageRep *im2*)

Cross product.

The function performs cross product (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoCross** (p. 173). The image functor instantiator : **HxInstantiatorCross** (p. 507).

```
13 {
14     return im1.binaryPixOp(im2, "cross");
15 }
```

5.31 HxCrossVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxCrossVal** (**HxImageRep** im, **HxValue** val)

  Cross product.

5.31.1 Detailed Description

5.31.2 Function Documentation

5.31.2.1 HxImageRep L_HXIMAGEREP HxCrossVal (HxImageRep *im*, HxValue *val*)

Cross product.

The function performs cross product (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoCross** (p. 173). The image functor instantiator : **HxInstantiatorCrossV** (p. 508).

```
13 {
14     return im.binaryPixOp(val, "cross");
15 }
```

5.32 HxDistanceTransform.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDistanceTransform** (**HxImageRep** img)

  Distance transform replaces each pixel of an object with an estimate of its shortest distance to the background (the distance to the nearest background pixel).

5.32.1 Detailed Description

5.32.2 Function Documentation

5.32.2.1 HxImageRep L_HXIMAGEREP HxDistanceTransform (HxImageRep *img*)

Distance transform replaces each pixel of an object with an estimate of its shortest distance to the background (the distance to the nearest background pixel).

Background is defined as all pixels with value 0.

```
86 {
87     HxImageRep input = BinMap(HxImageAsDouble(img), inf, 0);
88
89     double filterData[] = {
90         inf,    sqrt_5, inf,    sqrt_5, inf,
```

```
91         sqrt_5, sqrt_2, 1,      sqrt_2, sqrt_5,
92         inf,    1,      0,      1,      inf,
93         sqrt_5, sqrt_2, 1,      sqrt_2, sqrt_5,
94         inf,    sqrt_5, inf,    sqrt_5, inf
95     };
96     HxImageRep kernel = HxMakeFromDoubleData(1, 2, HxSizes(5, 5, 1), filterData);
97
98     return input.recursiveNeighOp(kernel, "add", "minAssign");
99 }
```

5.33 HxDiv.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDiv** (**HxImageRep** im1, **HxImageRep** im2)

  Division.

5.33.1 Detailed Description

5.33.2 Function Documentation

5.33.2.1 HxImageRep L_HXIMAGEREP HxDiv (HxImageRep *im1*, HxImageRep *im2*)

Division.

The function performs division (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoDiv** (p. 174). The image functor instantiator : **HxInstantiatorDiv** (p. 509).

```
13 {
14     return im1.binaryPixOp(im2, "div");
15 }
```

5.34 HxDivVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDivVal** (**HxImageRep** im, **HxValue** val)

  Division.

5.34.1 Detailed Description

5.34.2 Function Documentation

5.34.2.1 HxImageRep L_HXIMAGEREP HxDivVal (HxImageRep *im*, HxValue *val*)

Division.

The function performs division (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoDiv** (p. 174). The image functor instantiator : **HxInstantiatorDivV** (p. 509).

```
13 {
14     return im.binaryPixOp(val, "div");
15 }
```

5.35 HxDot.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDot** (**HxImageRep** im1, **HxImageRep** im2)

  Dot product .

5.35.1 Detailed Description

5.35.2 Function Documentation

5.35.2.1 HxImageRep L_HXIMAGEREP HxDot (HxImageRep *im1*, HxImageRep *im2*)

Dot product .

The function performs dot product (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoDot** (p. 175). The image functor instantiator : **HxInstantiatorDot** (p. 510).

```
13 {
14     return im1.binaryPixOp(im2, "dot");
15 }
```

5.36 HxDotVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDotVal** (**HxImageRep** im, **HxValue** val)

  Dot product .

5.36.1 Detailed Description

5.36.2 Function Documentation

5.36.2.1 HxImageRep L_HXIMAGEREP HxDotVal (HxImageRep *im*, HxValue *val*)

Dot product .

The function performs dot product (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoDot** (p. 175). The image functor instantiator : **HxInstantiatorDotV** (p. 511).

```
13 {
14     return im.binaryPixOp(val, "dot");
15 }
```

5.37 HxEqual.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxEqual** (**HxImageRep** im1, **HxImageRep** im2)

  Equal.

5.37.1 Detailed Description

5.37.2 Function Documentation

5.37.2.1 HxImageRep L_HXIMAGEREP HxEqual (HxImageRep *im1*, HxImageRep *im2*)

Equal.

The function performs equal (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoEqual** (p. 176). The image functor instantiator : **HxInstantiatorEqual** (p. 511).

```
13 {
14     return im1.binaryPixOp(im2, "equal");
15 }
```

5.38 HxEqualVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxEqualVal** (**HxImageRep** im, **HxValue** val)
  Equal.

5.38.1 Detailed Description

5.38.2 Function Documentation

5.38.2.1 HxImageRep L_HXIMAGEREP HxEqualVal (HxImageRep *im*, HxValue *val*)

Equal.

The function performs equal (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoEqual** (p. 176). The image functor instantiator : **HxInstantiatorEqualV** (p. 512).

```
13 {
14     return im.binaryPixOp(val, "equal");
15 }
```

5.39 HxExp.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxExp** (**HxImageRep** im)
  Exponent.

5.39.1 Detailed Description

5.39.2 Function Documentation

5.39.2.1 HxImageRep L_HXIMAGEREP HxExp (HxImageRep *im*)

Exponent.

The function computes the exponent (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoExp** (p. 733). The image functor instantiator : **HxInstantiatorExp** (p. 513).

```
13 {
14     return im.unaryPixOp("exp");
15 }
```

5.40 HxExportByteData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportByteData** (**HxImageRep** img, HxByte *data)

  Export image data to array of HxByte.

5.40.1 Detailed Description

5.40.2 Function Documentation

5.40.2.1 void L_HXIMAGEREP HxExportByteData (HxImageRep *img*, HxByte * *data*)

Export image data to array of HxByte.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
14 {
15     HxTagList tags;
16     HxString exportOp
17         = HxString("exportPix<") + HxString(HxClassName<HxByte>()) + ">";
18     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
19     img.exportOp(exportOp, tags);
20 }
```

5.41 HxExportDoubleData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportDoubleData** (**HxImageRep** img, double *data)

  Export image data to array of double.

5.41.1 Detailed Description

5.41.2 Function Documentation

5.41.2.1 void L_HXIMAGEREP HxExportDoubleData (HxImageRep *img*, double * *data*)

Export image data to array of double.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     HxTagList tags;
15     HxString exportOp
16         = HxString("exportPix<") + HxString(HxClassName<double>()) + ">";
17     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
18     img.exportOp(exportOp, tags);
19 }
```

5.42 HxExportFloatData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportFloatData** (**HxImageRep** img, float *data)

  Export image data to array of float.

5.42.1 Detailed Description

5.42.2 Function Documentation

5.42.2.1 void L_HXIMAGEREP HxExportFloatData (HxImageRep *img*, float * *data*)

Export image data to array of float.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     HxTagList tags;
15     HxString exportOp
16         = HxString("exportPix<") + HxString(HxClassName<float>()) + ">";
17     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
18     img.exportOp(exportOp, tags);
19 }
```

5.43 HxExportIntData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportIntData** (**HxImageRep** img, int *data)

  Export image data to array of int.

5.43.1 Detailed Description

5.43.2 Function Documentation

5.43.2.1 void L_HXIMAGEREP HxExportIntData (HxImageRep *img*, int * *data*)

Export image data to array of int.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     HxTagList tags;
15     HxString exportOp
16         = HxString("exportPix<") + HxString(HxClassName<int>()) + ">";
17     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
18     img.exportOp(exportOp, tags);
19 }
```

5.44 HxExportMatlabPixels.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportMatlabPixels** (**HxImageRep** img, double *pixels)

  Export image data to array of int.

5.44.1 Detailed Description

5.44.2 Function Documentation

5.44.2.1 void L_HXIMAGEREP HxExportMatlabPixels (HxImageRep *img*, double * *pixels*)

Export image data to array of int.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     img.getDoublePixels(pixels);
15 }
```

5.45 HxExportPpmPixels.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportPpmPixels** (**HxImageRep** img, const HxByte *pixels)

5.45.1 Detailed Description

5.46 HxExportShortData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxExportShortData** (**HxImageRep** img, short *data)

  Export image data to array of short.

5.46.1 Detailed Description

5.46.2 Function Documentation

5.46.2.1 void L_HXIMAGEREP HxExportShortData (HxImageRep *img*, short * *data*)

Export image data to array of short.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     HxTagList tags;
15     HxString exportOp
16         = HxString("exportPix<") + HxString(HxClassName<short>()) + ">";
17     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
18     img.exportOp(exportOp, tags);
19 }
```

5.47 HxExportSi.h File Reference

```
#include "HxImageRep.h"
```

Functions

- L_HXIMAGEREP IMAGE * **HxExportSi** (**HxImageRep** img)

  Export image data to a SciImage image.

5.47.1 Detailed Description

5.47.2 Function Documentation

5.47.2.1 L_HXIMAGEREP IMAGE* HxExportSi (HxImageRep *img*)

Export image data to a SciImage image.

```
13 {
14     return img.toImageSi();
15 }
```

5.48 HxExtend.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxExtend** (**HxImageRep** img, **HxImageRep** background, **HxPoint** begin)

  Extension of domain.

5.48.1 Detailed Description

5.48.2 Function Documentation

5.48.2.1 HxImageRep L_HXIMAGEREP HxExtend (HxImageRep *img*, HxImageRep *background*, HxPoint *begin*)

Extension of domain.

Extend the domain of the image to the size of the background image. The image is put at the position indicated by begin. Points are treated as pixel coordinates (integers).

```
13 {
14     return img.extend(background, begin);
15 }
```

5.49 HxExtendVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxExtendVal** (**HxImageRep** img, **HxSizes** newSize, **HxValue** background, **HxPoint** begin)

  Extension of domain.

5.49.1 Detailed Description

5.49.2 Function Documentation

5.49.2.1 HxImageRep L_HXIMAGEREP HxExtendVal (HxImageRep *img*, HxSizes *newSize*, HxValue *background*, HxPoint *begin*)

Extension of domain.

Extend the domain of the image to the given size. The image is put at the position indicated by begin. Points are treated as pixel coordinates (integers).

```
14 {
15     return img.extend(newSize, background, begin);
16 }
```

5.50 HxFloor.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxFloor** (**HxImageRep** im)

  Floor.

5.50.1 Detailed Description

5.50.2 Function Documentation

5.50.2.1 HxImageRep L_HXIMAGEREP HxFloor (HxImageRep *im*)

Floor.

The function computes the floor (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoFloor** (p. 734). The image functor instantiator : **HxInstantiatorFloor** (p. 513).

```
13 {
14     return im.unaryPixOp("floor");
15 }
```

5.51 HxFuncBorderOp.h File Reference

```
#include "HxBorderType.h"
```

Functions

- template<class DataPtrT> void **HxFuncBorderMirror2d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize)

  Set the border of a 2D image by mirroring.

- template<class DataPtrT> void **HxFuncBorderMirror3d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize)

  Set the border of a 3D image by mirroring.

- template<class DataPtrT, class ArithT> void **HxFuncBorderConstant2d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize, ArithT value)

  Set the border of a 2D image to a constant value.

- template<class DataPtrT, class ArithT> void **HxFuncBorderConstant3d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize, ArithT value)

  Set the border of a 3D image to a constant value.

- template<class DataPtrT> void **HxFuncBorderPropagate2d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize)

  Set the border of a 2D image by propagating the "last" value.

- template<class DataPtrT> void **HxFuncBorderPropagate3d** (DataPtrT imgPtr, **HxSizes** imgSize, **HxSizes** borderSize)

  Set the border of a 3D image by propagating the "last" value.

5.51.1 Detailed Description

5.51.2 Function Documentation

5.51.2.1 template<class DataPtrT> void HxFuncBorderMirror2d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*)

Set the border of a 2D image by mirroring.

```
19 {
20     int     borderWidth = borderSize.x();
21     int     borderHeight = borderSize.y();
22     int     imgWidth = imgSize.x() - borderWidth * 2;
23     int     imgHeight = imgSize.y() - borderHeight * 2;
24     int     x, y;
25 
26     // mirror top part
27     for (y=0 ; y<borderHeight ; y++) {
28         DataPtrT srcPtr = imgPtr;
29         srcPtr.incXYZ(borderWidth, borderHeight + y, 0);
30         DataPtrT dstPtr = imgPtr;
31         dstPtr.incXYZ(borderWidth, borderHeight - 1 - y, 0);
32         for (x=0 ; x<imgWidth ; x++) {
33             dstPtr.write(srcPtr.read());
34             srcPtr.incX();
35             dstPtr.incX();
36         }
```

```
37     }
38
39     // mirror bottom part
40     for (y=0 ; y<borderHeight ; y++) {
41         DataPtrT srcPtr = imgPtr;
42         srcPtr.incXYZ(borderWidth, borderHeight + imgHeight - 1 - y, 0);
43         DataPtrT dstPtr = imgPtr;
44         dstPtr.incXYZ(borderWidth, borderHeight + imgHeight + y, 0);
45         for (x=0 ; x<imgWidth ; x++) {
46             dstPtr.write(srcPtr.read());
47             srcPtr.incX();
48             dstPtr.incX();
49         }
50     }
51
52     // mirror left part including upper and lower "corner"
53     int totalHeight = imgHeight + 2*borderHeight;
54     for (y=0 ; y<totalHeight ; y++) {
55         DataPtrT srcPtr = imgPtr;
56         srcPtr.incXYZ(borderWidth, y, 0);
57         DataPtrT dstPtr = imgPtr;
58         dstPtr.incXYZ(borderWidth - 1, y, 0);
59         for (x=0 ; x<borderWidth ; x++) {
60             dstPtr.write(srcPtr.read());
61             srcPtr.incX();
62             dstPtr.decX();
63         }
64     }
65
66     // mirror right part including upper and lower "corner"
67     for (y=0 ; y<totalHeight ; y++) {
68         DataPtrT srcPtr = imgPtr;
69         srcPtr.incXYZ(borderWidth + imgWidth - 1, y, 0);
70         DataPtrT dstPtr = imgPtr;
71         dstPtr.incXYZ(borderWidth + imgWidth, y, 0);
72         for (x=0 ; x<borderWidth ; x++) {
73             dstPtr.write(srcPtr.read());
74             srcPtr.decX();
75             dstPtr.incX();
76         }
77     }
78 }
```

5.51.2.2 template<class DataPtrT> void HxFuncBorderMirror3d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*)

Set the border of a 3D image by mirroring.

```
84 {
85     int     borderWidth = borderSize.x();
86     int     borderHeight = borderSize.y();
87     int     borderDepth = borderSize.z();
88     int     imgWidth = imgSize.x() - borderWidth * 2;
89     int     imgHeight = imgSize.y() - borderHeight * 2;
90     int     imgDepth = imgSize.z() - borderDepth * 2;
91     int     x, y, z;
92
93     // mirror top part of each XY plane
94     for (z=0 ; z<imgDepth ; z++) {
95         for (y=0 ; y<borderHeight ; y++) {
96             DataPtrT srcPtr = imgPtr;
97             srcPtr.incXYZ(borderWidth, borderHeight + y, borderDepth + z);
```

```
98              DataPtrT dstPtr = imgPtr;
99              dstPtr.incXYZ(borderWidth, borderHeight - 1 - y, borderDepth + z);
100              for (x=0 ; x<imgWidth ; x++) {
101                  dstPtr.write(srcPtr.read());
102                  srcPtr.incX();
103                  dstPtr.incX();
104              }
105          }
106      }
107      // mirror bottom part of each XY plane
108      for (z=0 ; z<imgDepth ; z++) {
109          for (y=0 ; y<borderHeight ; y++) {
110              DataPtrT srcPtr = imgPtr;
111              srcPtr.incXYZ(
112                  borderWidth, borderHeight + imgHeight - 1 - y, borderDepth + z);
113              DataPtrT dstPtr = imgPtr;
114              dstPtr.incXYZ(
115                  borderWidth, borderHeight + imgHeight + y, borderDepth + z);
116              for (x=0 ; x<imgWidth ; x++) {
117                  dstPtr.write(srcPtr.read());
118                  srcPtr.incX();
119                  dstPtr.incX();
120              }
121          }
122      }
123      // mirror left part of each XY plane including upper and lower "corner"
124      int totalHeight = imgHeight + 2*borderHeight;
125      for (z=0 ; z<imgDepth ; z++) {
126          for (y=0 ; y<totalHeight ; y++) {
127              DataPtrT srcPtr = imgPtr;
128              srcPtr.incXYZ(borderWidth, y, borderDepth + z);
129              DataPtrT dstPtr = imgPtr;
130              dstPtr.incXYZ(borderWidth - 1, y, borderDepth + z);
131              for (x=0 ; x<borderWidth ; x++) {
132                  dstPtr.write(srcPtr.read());
133                  srcPtr.incX();
134                  dstPtr.decX();
135              }
136          }
137      }
138      // mirror right part of each XY plane including upper and lower "corner"
139      for (z=0 ; z<imgDepth ; z++) {
140          for (y=0 ; y<totalHeight ; y++) {
141              DataPtrT srcPtr = imgPtr;
142              srcPtr.incXYZ(borderWidth + imgWidth - 1, y, borderDepth + z);
143              DataPtrT dstPtr = imgPtr;
144              dstPtr.incXYZ(borderWidth + imgWidth, y, borderDepth + z);
145              for (x=0 ; x<borderWidth ; x++) {
146                  dstPtr.write(srcPtr.read());
147                  srcPtr.decX();
148                  dstPtr.incX();
149              }
150          }
151      }
152      int totalPlane = (imgWidth + 2*borderWidth) * (imgHeight + 2*borderHeight);
153      // mirror front planes
154      for (z=0 ; z<borderDepth ; z++) {
155          DataPtrT srcPtr = imgPtr;
156          srcPtr.incXYZ(0, 0, borderDepth + z);
157          DataPtrT dstPtr = imgPtr;
158          dstPtr.incXYZ(0, 0, borderDepth - 1 - z);
159          for (x=0 ; x<totalPlane ; x++) {
160              dstPtr.write(srcPtr.read());
161              srcPtr.incX();
162              dstPtr.incX();
```

```
163         }
164     }
165     // mirror back planes
166     for (z=0 ; z<borderDepth ; z++) {
167         DataPtrT srcPtr = imgPtr;
168         srcPtr.incXYZ(0, 0, borderDepth + imgDepth - 1 - z);
169         DataPtrT dstPtr = imgPtr;
170         dstPtr.incXYZ(0, 0, borderDepth + imgDepth + z);
171         for (x=0 ; x<totalPlane ; x++) {
172             dstPtr.write(srcPtr.read());
173             srcPtr.incX();
174             dstPtr.incX();
175         }
176     }
177 }
```

5.51.2.3 template<class DataPtrT, class ArithT> void HxFuncBorderConstant2d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*, ArithT *value*)

Set the border of a 2D image to a constant value.

```
183 {
184     int     borderWidth = borderSize.x();
185     int     borderHeight = borderSize.y();
186     int     imgWidth = imgSize.x() - borderWidth * 2;
187     int     imgHeight = imgSize.y() - borderHeight * 2;
188     int     x, y, totalWidth = imgWidth + 2*borderWidth;
189 
190     DataPtrT dstPtr = imgPtr;
191 
192     // set left and right part
193     for (y=0 ; y<imgHeight ; y++) {
194         DataPtrT dstPtr = imgPtr;
195         dstPtr.incY(borderHeight + y);
196         for (x=0 ; x<borderWidth ; x++)
197             dstPtr.writeIncX(value);
198         dstPtr.incX(imgWidth);
199         for (x=0 ; x<borderWidth ; x++)
200             dstPtr.writeIncX(value);
201     }
202 
203     // set top and bottom part including left and right "corners"
204     for (y=0 ; y<borderHeight ; y++) {
205         DataPtrT dstPtr = imgPtr;
206         dstPtr.incY(y);
207         for (x=0 ; x<totalWidth ; x++)
208             dstPtr.writeIncX(value);
209         dstPtr.incXYZ(-totalWidth, imgHeight + borderHeight, 0);
210         for (x=0 ; x<totalWidth ; x++)
211             dstPtr.writeIncX(value);
212     }
213 }
```

5.51.2.4 template<class DataPtrT, class ArithT> void HxFuncBorderConstant3d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*, ArithT *value*)

Set the border of a 3D image to a constant value.

```
219 {
```

```
220     int     borderWidth = borderSize.x();
221     int     borderHeight = borderSize.y();
222     int     borderDepth = borderSize.z();
223     int     imgWidth = imgSize.x() - borderWidth * 2;
224     int     imgHeight = imgSize.y() - borderHeight * 2;
225     int     imgDepth = imgSize.z() - borderDepth * 2;
226     int     totalWidth = imgWidth + 2*borderWidth;
227     int     totalHeight = imgHeight + 2*borderHeight;
228     int     totalDepth = imgDepth + 2*borderDepth;
229     int     x, y, z;
230
231     DataPtrT basePtr = imgPtr;
232     imgPtr.incZ(imgDepth);
233
234     for (z=0 ; z<imgDepth ; z++)
235     {
236
237         // set left and right part
238         for (y=0 ; y<imgHeight ; y++) {
239             DataPtrT dstPtr = imgPtr;
240             dstPtr.incY(borderHeight + y);
241             for (x=0 ; x<borderWidth ; x++)
242                 dstPtr.writeIncX(value);
243             dstPtr.incX(imgWidth);
244             for (x=0 ; x<borderWidth ; x++)
245                 dstPtr.writeIncX(value);
246         }
247
248         // set top and bottom part including left and right "corners"
249         for (y=0 ; y<borderHeight ; y++) {
250             DataPtrT dstPtr = imgPtr;
251             dstPtr.incY(y);
252             for (x=0 ; x<totalWidth ; x++)
253                 dstPtr.writeIncX(value);
254             dstPtr.incXYZ(-totalWidth, imgHeight + borderHeight, 0);
255             for (x=0 ; x<totalWidth ; x++)
256                 dstPtr.writeIncX(value);
257         }
258
259         imgPtr.incZ();
260     }
261
262     // set front and back planes
263
264     for (int p=0; p<totalDepth; p += imgDepth + borderDepth)
265     {
266         imgPtr = basePtr;
267         imgPtr.incZ(p);
268         for (z=0; z<borderDepth; z++)
269         {
270             for (y=0; y<totalHeight; y++)
271             {
272                 for (x=0; x<totalWidth; x++)
273                 {
274                     imgPtr.writeIncX(value);
275                 }
276                 imgPtr.decX(totalWidth);
277                 imgPtr.incY();
278             }
279             imgPtr.decY(totalHeight);
280             imgPtr.incZ();
281         }
282     }
283 }
```

5.51.2.5 template<class DataPtrT> void HxFuncBorderPropagate2d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*)

Set the border of a 2D image by propagating the ”last” value.

```
289 {
290     int     borderWidth = borderSize.x();
291     int     borderHeight = borderSize.y();
292     int     imgWidth = imgSize.x() - borderWidth * 2;
293     int     imgHeight = imgSize.y() - borderHeight * 2;
294     int     x, y, totalWidth = imgWidth + 2*borderWidth;
295 
296     DataPtrT srcPtr = imgPtr;
297     DataPtrT dstPtr = imgPtr;
298 
299     // propagate left and right part
300     for (y=0 ; y<imgHeight ; y++) {
301         dstPtr = srcPtr = imgPtr;
302         srcPtr.incXYZ(borderWidth, borderHeight + y, 0);
303         dstPtr.incY(borderHeight + y);
304         for (x=0 ; x<borderWidth ; x++)
305             dstPtr.writeIncX(srcPtr.read());
306 
307         dstPtr = srcPtr = imgPtr;
308         srcPtr.incXYZ(borderWidth + imgWidth - 1, borderHeight + y, 0);
309         dstPtr.incXYZ(borderWidth + imgWidth, borderHeight + y, 0);
310         for (x=0 ; x<borderWidth ; x++)
311             dstPtr.writeIncX(srcPtr.read());
312     }
313 
314     // propagate top and bottom part including left and right "corners"
315     for (y=0 ; y<borderHeight ; y++) {
316         dstPtr = srcPtr = imgPtr;
317         srcPtr.incY(borderHeight);
318         dstPtr.incY(y);
319         for (x=0 ; x<totalWidth ; x++)
320             dstPtr.writeIncX(srcPtr.readIncX());
321 
322         dstPtr = srcPtr = imgPtr;
323         srcPtr.incY(borderHeight + imgHeight - 1);
324         dstPtr.incY(borderHeight + imgHeight + y);
325         for (x=0 ; x<totalWidth ; x++)
326             dstPtr.writeIncX(srcPtr.readIncX());
327     }
328 
329 }
```

5.51.2.6 template<class DataPtrT> void HxFuncBorderPropagate3d (DataPtrT *imgPtr*, HxSizes *imgSize*, HxSizes *borderSize*)

Set the border of a 3D image by propagating the ”last” value.

```
335 {
336     int     borderWidth = borderSize.x();
337     int     borderHeight = borderSize.y();
338     int     borderDepth = borderSize.z();
339     int     imgWidth = imgSize.x() - borderWidth * 2;
340     int     imgHeight = imgSize.y() - borderHeight * 2;
341     int     imgDepth = imgSize.y() - borderDepth * 2;
342     int     x, y, z;
```

```
343     int     totalWidth = imgWidth + 2*borderWidth;
344     int     totalHeight = imgHeight + 2*borderHeight;
345     int     totalDepth = imgDepth + 2*borderDepth;
346
347     DataPtrT basePtr = imgPtr;
348     DataPtrT srcPtr = imgPtr;
349     DataPtrT dstPtr = imgPtr;
350
351     imgPtr.incZ(borderDepth);
352
353     for (z=0; z<imgDepth; z++)
354     {
355         // propagate left and right part
356         for (y=0 ; y<imgHeight ; y++) {
357             dstPtr = srcPtr = imgPtr;
358             srcPtr.incXYZ(borderWidth, borderHeight + y, 0);
359             dstPtr.incY(borderHeight + y);
360             for (x=0 ; x<borderWidth ; x++)
361                 dstPtr.writeIncX(srcPtr.read());
362
363             dstPtr = srcPtr = imgPtr;
364             srcPtr.incXYZ(borderWidth + imgWidth - 1, borderHeight + y, 0);
365             dstPtr.incXYZ(borderWidth + imgWidth, borderHeight + y, 0);
366             for (x=0 ; x<borderWidth ; x++)
367                 dstPtr.writeIncX(srcPtr.read());
368         }
369
370         // propagate top and bottom part including left and right "corners"
371         for (y=0 ; y<borderHeight ; y++) {
372             dstPtr = srcPtr = imgPtr;
373             srcPtr.incY(borderHeight);
374             dstPtr.incY(y);
375             for (x=0 ; x<totalWidth ; x++)
376                 dstPtr.writeIncX(srcPtr.readIncX());
377
378             dstPtr = srcPtr = imgPtr;
379             srcPtr.incY(borderHeight + imgHeight - 1);
380             dstPtr.incY(borderHeight + imgHeight + y);
381             for (x=0 ; x<totalWidth ; x++)
382                 dstPtr.writeIncX(srcPtr.readIncX());
383         }
384         imgPtr.incZ();
385     }
386
387     imgPtr = basePtr;
388
389     // propagate front planes
390
391     for (z=0; z<borderDepth; z++)
392     {
393         srcPtr = imgPtr;
394         srcPtr.incZ(borderDepth);
395         dstPtr = imgPtr;
396         dstPtr.incZ(z);
397         for (y=0; y<totalHeight; y++)
398         {
399             for (x=0; x<totalWidth; x++)
400             {
401                 dstPtr.writeIncX(srcPtr.readIncX());
402             }
403             dstPtr.decX(totalWidth);
404             dstPtr.incY();
405             srcPtr.decX(totalWidth);
406             srcPtr.incY();
407         }
```

```
408     }
409
410     // propagate back planes
411
412     for (z=0; z<borderDepth; z++)
413     {
414         srcPtr = imgPtr;
415         srcPtr.incZ(imgDepth + borderDepth - 1);
416         dstPtr = imgPtr;
417         dstPtr.incZ(imgDepth + borderDepth + z);
418         for (y=0; y<totalHeight; y++)
419         {
420             for (x=0; x<totalWidth; x++)
421             {
422                 dstPtr.writeIncX(srcPtr.readIncX());
423             }
424             dstPtr.decX(totalWidth);
425             dstPtr.incY();
426             srcPtr.decX(totalWidth);
427             srcPtr.incY();
428         }
429     }
430 }
```

5.52 HxFuncKernelNgbOp2d.h File Reference

```
#include "HxNgbOpCategory.h"
```

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, KernelT &kernel, const HxNgbLoopTag, const HxNgb1PhaseTag, const HxNgbTransInVarTag)

  2 dimensional, translation invariant, 1 phase, loop neighbourhood operation with kernel.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, KernelT &kernel, const HxNgbLoopTag, const HxNgb2PhaseTag, const HxNgbTransInVarTag)

  2 dimensional, translation invariant, 2 phase, loop neighbourhood operation using a kernel.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, KernelT &kernel)

  Dispatch function for HxFuncKernelNgbOp2d.

5.52.1 Detailed Description

5.52.2 Function Documentation

5.52.2.1 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, KernelT & *kernel*, const HxNgbLoopTag, const HxNgb1PhaseTag, const HxNgbTransInVarTag)

2 dimensional, translation invariant, 1 phase, loop neighbourhood operation with kernel.

5.52.2.2 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, KernelT & *kernel*, const *HxNgbLoopTag*, const *HxNgb2PhaseTag*, const *HxNgbTransInVarTag*)

2 dimensional, translation invariant, 2 phase, loop neighbourhood operation using a kernel.

5.52.2.3 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, KernelT & *kernel*) [inline]

Dispatch function for HxFuncKernelNgbOp2d.

Dispatch is based on the categories defined in NgbT.

```
62 {
63     HxFuncKernelNgbOp2d(
64         dstPtr, srcPtr, dstSize, ngb, kernel,
65         typename NgbT::IteratorCategory(),
66         typename NgbT::PhaseCategory(),
67         typename NgbT::TransVarianceCategory());
68 }
```

5.53 HxFuncNgbOp2d.h File Reference

```
#include "HxNgbOpCategory.h"
```

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, const HxNgbCnumTag, const HxNgb1PhaseTag, const HxNgbTransInVarTag)

  2 dimensional, translation invariant, 1 phase, coordinate enumerated neighbourhood operation.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, const HxNgbLoopTag, const HxNgb1PhaseTag, const HxNgbTransInVarTag)

  2 dimensional, translation invariant, 1 phase, loop neighbourhood operation.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, const HxNgbLoopTag, const HxNgb2PhaseTag, const HxNgbTransInVarTag)

  2 dimensional, translation invariant, 2 phase, loop neighbourhood operation.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb)

  Dispatch function for HxFuncNgbOp2d.

5.53.1 Detailed Description

5.53.2 Function Documentation

5.53.2.1 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, const HxNgbCnumTag, const HxNgb1PhaseTag, const HxNgbTransInVarTag)

2 dimensional, translation invariant, 1 phase, coordinate enumerated neighbourhood operation.

5.53.2.2 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, const *HxNgbLoopTag*, const *HxNgb1PhaseTag*, const *HxNgbTransInVarTag*)

2 dimensional, translation invariant, 1 phase, loop neighbourhood operation.

5.53.2.3 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, const *HxNgbLoopTag*, const *HxNgb2PhaseTag*, const *HxNgbTransInVarTag*)

2 dimensional, translation invariant, 2 phase, loop neighbourhood operation.

5.53.2.4 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*) [inline]

Dispatch function for HxFuncNgbOp2d.

Dispatch is based on the categories defined in NgbT.

```
58 {
59     HxFuncNgbOp2d(
60         dstPtr, srcPtr, dstSize, ngb,
61         typename NgbT::IteratorCategory(),
62         typename NgbT::PhaseCategory(),
63         typename NgbT::TransVarianceCategory());
64 }
```

5.54 HxFuncPixOp.h File Reference

```
#include "HxPixOpCategory.h"
```

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void **HxFuncUnaryPixOp** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int nPix, UpoT &upo)

  Unary pixel operation.

- template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void **HxFuncBinaryPixOp** (DstDataPtrT dstPtr, Src1DataPtrT src1Ptr, Src2DataPtrT src2Ptr, int nPix, BpoT &bpo)

  Binary pixel operation.

- template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void **HxFuncMultiPixOp** (DstDataPtrT dstPtr, SrcDataPtrArray &srcPtrs, int nPix, MpoT &mpo)

  Multi pixel operation.

- template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void **HxFuncMNPixOp** (DstDataPtrArray &dstPtr, SrcDataPtrArray &srcPtrs, int nPix, MNpoT &mpo)

  M output, N input pixel operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpOutTag** dummy1, **HxPixOpTransInVarTag** dummy2, **HxPixOp1PhaseTag** dummy3)

  Translation invariant, 1 phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpOutTag** dummy1, **HxPixOpTransVarTag** dummy2, **HxPixOp1PhaseTag** dummy3)

  Translation variant, 1 phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpInTag** dummy1, **HxPixOpTransInVarTag** dummy2, **HxPixOp1PhaseTag** dummy3)

  Translation invariant, 1 phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpInTag** dummy1, **HxPixOpTransVarTag** dummy2, **HxPixOp1PhaseTag** dummy3)

  Translation variant, 1 phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpOutTag** dummy1, **HxPixOpTransInVarTag** dummy2, **HxPixOpNPhaseTag** dummy3)

  Translation invariant, n phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpOutTag** dummy1, **HxPixOpTransVarTag** dummy2, **HxPixOpNPhaseTag** dummy3)

  Translation variant, n phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpInTag** dummy1, **HxPixOpTransInVarTag** dummy2, **HxPixOpNPhaseTag** dummy3)

Translation invariant, n phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxPixOpInTag** dummy1, **HxPixOpTransVarTag** dummy2, **HxPixOpNPhaseTag** dummy3)

  Translation variant, n phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOp** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp)

  Dispatch function for HxFuncInOutPixOp.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes**, **HxTagList** &tags, const **HxPixOpOutTag**, const **HxPixOpTransInVarTag**, const **HxPixOp1PhaseTag**)

  Initialization for translation invariant, 1 phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxPixOpOutTag**, const **HxPixOpTransVarTag**, const **HxPixOp1PhaseTag**)

  Initialization for translation variant, 1 phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes**, **HxTagList** &tags, const **HxPixOpInTag**, const **HxPixOpTransInVarTag**, const **HxPixOp1PhaseTag**)

  Initialization for translation invariant, 1 phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxPixOpInTag**, const **HxPixOpTransVarTag**, const **HxPixOp1PhaseTag**)

  Initialization for translation variant, 1 phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes**, **HxTagList** &tags, const **HxPixOpOutTag**, const **HxPixOpTransInVarTag**, const **HxPixOpNPhaseTag**)

  Initialization for translation invariant, n phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxPixOpOutTag**, const **HxPixOpTransVarTag**, const **HxPixOpNPhaseTag**)

  Initialization for translation variant, n phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes**, **HxTagList** &tags, const **HxPixOpInTag**, const **HxPixOpTransInVarTag**, const **HxPixOpNPhaseTag**)

  Initialization for translation invariant, n phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutPixOpInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxPixOpInTag**, const **HxPixOpTransVarTag**, const **HxPixOpNPhaseTag**)

  Initialization for translation variant, n phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOutPixOpInit** (**HxSizes** sizes, **HxTagList** &tags)

  Dispatch function for HxFuncInOutPixOpInit.

5.54.1 Detailed Description

5.54.2 Function Documentation

5.54.2.1 template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void HxFuncUnaryPixOp (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *nPix*, UpoT & *upo*)

Unary pixel operation.

```
22 {
23     while (--nPix >= 0)
24         dstPtr.writeIncX(upo.doIt(srcPtr.readIncX()));
25 }
```

5.54.2.2 template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void HxFuncBinaryPixOp (DstDataPtrT *dstPtr*, Src1DataPtrT *src1Ptr*, Src2DataPtrT *src2Ptr*, int *nPix*, BpoT & *bpo*)

Binary pixel operation.

5.54.2.3 template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void HxFuncMultiPixOp (DstDataPtrT *dstPtr*, SrcDataPtrArray & *srcPtrs*, int *nPix*, MpoT & *mpo*)

Multi pixel operation.

```
42 {
43     int n = srcPtrs.size();
44     typename SrcDataPtrArray::ArithType *src =
45     new typename SrcDataPtrArray::ArithType [n];
46 
47     while (--nPix >= 0) {
48         for (int i=0; i < n; i++)
49             src[i] = srcPtrs[i].readIncX();
50 
51         dstPtr.writeIncX(mpo.doIt(src));
52     }
53 
54     delete [] src;
55 }
```

5.54.2.4 template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void HxFuncMNPixOp (DstDataPtrArray & *dstPtr*, SrcDataPtrArray & *srcPtrs*, int *nPix*, MNpoT & *mpo*)

M output, N input pixel operation.

```
62 {
63     int n = srcPtrs.size();
64     int m = dstPtrs.size();
65     typename SrcDataPtrArray::ArithType *src =
66     new typename SrcDataPtrArray::ArithType [n];
67     typename DstDataPtrArray::ArithType *dst =
```

```
68     new typename DstDataPtrArray::ArithType [m];
69
70     while (--nPix >= 0) {
71     int i;
72         for (i=0; i < n; i++)
73             src[i] = srcPtrs[i].readIncX();
74
75         mpo.doIt(dst, src);
76
77         for (i=0; i < m; i++)
78             dstPtrs[i].writeIncX(dst[i]);
79     }
80
81     delete [] dst;
82     delete [] src;
83 }
```

5.54.2.5 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpOutTag *dummy1*, HxPixOpTransInVarTag *dummy2*, HxPixOp1PhaseTag *dummy3*)

Translation invariant, 1 phase pixel export operation.

```
155 {
156     for (int z=0; z<sizes.z(); z++)
157     {
158         for (int y=0; y<sizes.y(); y++)
159         {
160             HxFuncInOutPixOp(
161                 ptr, sizes.x(), pixOp, HxPixOpOutTag());
162             ptr.incY();
163         }
164         ptr.decY(sizes.y());
165         ptr.incZ();
166     }
167 }
```

5.54.2.6 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpOutTag *dummy1*, HxPixOpTransVarTag *dummy2*, HxPixOp1PhaseTag *dummy3*)

Translation variant, 1 phase pixel export operation.

```
193 {
194     for (int z=0; z<sizes.z(); z++)
195     {
196         for (int y=0; y<sizes.y(); y++)
197         {
198             HxFuncInOutPixOp(
199                 ptr, sizes.x(), pixOp, 0, y, z, HxPixOpOutTag());
200             ptr.incY();
201         }
202         ptr.decY(sizes.y());
203         ptr.incZ();
204     }
205 }
```

5.54.2.7 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpInTag *dummy1*, HxPixOpTransInVarTag *dummy2*, HxPixOp1PhaseTag *dummy3*)

Translation invariant, 1 phase pixel import operation.

5.54.2.8 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpInTag *dummy1*, HxPixOpTransVarTag *dummy2*, HxPixOp1PhaseTag *dummy3*)

Translation variant, 1 phase pixel import operation.

```
174 {
175     for (int z=0; z<sizes.z(); z++)
176     {
177         for (int y=0; y<sizes.y(); y++)
178         {
179             HxFuncInOutPixOp(
180                 ptr, sizes.x(), pixOp, 0, y, z, HxPixOpInTag());
181             ptr.incY();
182         }
183         ptr.decY(sizes.y());
184         ptr.incZ();
185     }
186 }
```

5.54.2.9 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpOutTag *dummy1*, HxPixOpTransInVarTag *dummy2*, HxPixOpNPhaseTag *dummy3*)

Translation invariant, n phase pixel export operation.

```
235 {
236     for (int phase=0; phase<pixOp.nrPhases(); phase++) {
237         pixOp.init(phase);
238         for (int z=0; z<sizes.z(); z++)
239         {
240             for (int y=0; y<sizes.y(); y++)
241             {
242                 HxFuncInOutPixOp(
243                     ptr, sizes.x(), pixOp, HxPixOpOutTag());
244                 ptr.incY();
245             }
246             ptr.decY(sizes.y());
247             ptr.incZ();
248         }
249         pixOp.done(phase);
250     }
251 }
```

5.54.2.10 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpOutTag *dummy1*, HxPixOpTransVarTag *dummy2*, HxPixOpNPhaseTag *dummy3*)

Translation variant, n phase pixel export operation.

```
281 {
282     for (int phase=0; phase<pixOp.nrPhases(); phase++) {
283         pixOp.init(phase);
284         for (int z=0; z<sizes.z(); z++)
285         {
286             for (int y=0; y<sizes.y(); y++)
287             {
288                 HxFuncInOutPixOp(
289                     ptr, sizes.x(), pixOp, 0, y, z, HxPixOpOutTag());
290                 ptr.incY();
291             }
292             ptr.decY(sizes.y());
293             ptr.incZ();
294         }
295         pixOp.done(phase);
296     }
297 }
```

5.54.2.11 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpInTag *dummy1*, HxPixOpTransInVarTag *dummy2*, HxPixOpNPhaseTag *dummy3*)

Translation invariant, n phase pixel import operation.

```
212 {
213     for (int phase=0; phase<pixOp.nrPhases(); phase++) {
214         pixOp.init(phase);
215         for (int z=0; z<sizes.z(); z++)
216         {
217             for (int y=0; y<sizes.y(); y++)
218             {
219                 HxFuncInOutPixOp(
220                     ptr, sizes.x(), pixOp, HxPixOpInTag());
221                 ptr.incY();
222             }
223             ptr.decY(sizes.y());
224             ptr.incZ();
225         }
226         pixOp.done(phase);
227     }
228 }
```

5.54.2.12 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxPixOpInTag *dummy1*, HxPixOpTransVarTag *dummy2*, HxPixOpNPhaseTag *dummy3*)

Translation variant, n phase pixel import operation.

```
258 {
259     for (int phase=0; phase<pixOp.nrPhases(); phase++) {
260         pixOp.init(phase);
261         for (int z=0; z<sizes.z(); z++)
262         {
263             for (int y=0; y<sizes.y(); y++)
264             {
265                 HxFuncInOutPixOp(
266                     ptr, sizes.x(), pixOp, 0, y, z, HxPixOpInTag());
267                 ptr.incY();
```

```
268             }
269             ptr.decY(sizes.y());
270             ptr.incZ();
271         }
272         pixOp.done(phase);
273     }
274 }
```

5.54.2.13 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOp (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*) [inline]

Dispatch function for HxFuncInOutPixOp.

Dispatch is based on the categories defined in PixOpT.

```
141 {
142     HxFuncInOutPixOp(
143         ptr, sizes, pixOp,
144         typename PixOpT::DirectionCategory(),
145         typename PixOpT::TransVarianceCategory(),
146         typename PixOpT::PhaseCategory());
147 }
```

5.54.2.14 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes, HxTagList & *tags*, const *HxPixOpOutTag*, const *HxPixOpTransInVarTag*, const *HxPixOp1PhaseTag*) [inline]

Initialization for translation invariant, 1 phase pixel export operation.

```
161 {
162     PixOpT pixOp(tags);
163     return pixOp;
164 }
```

5.54.2.15 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes *sizes*, HxTagList & *tags*, const *HxPixOpOutTag*, const *HxPixOpTransVarTag*, const *HxPixOp1PhaseTag*) [inline]

Initialization for translation variant, 1 phase pixel export operation.

```
173 {
174     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
175     return pixOp;
176 }
```

5.54.2.16 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes, HxTagList & *tags*, const *HxPixOpInTag*, const *HxPixOpTransInVarTag*, const *HxPixOp1PhaseTag*) [inline]

Initialization for translation invariant, 1 phase pixel import operation.

```
185 {
186     PixOpT pixOp(tags);
187     return pixOp;
188 }
```

5.54.2.17 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes *sizes*, HxTagList & *tags*, const *HxPixOpInTag*, const *HxPixOpTransVarTag*, const *HxPixOp1PhaseTag*) [inline]

Initialization for translation variant, 1 phase pixel import operation.

```
197 {
198     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
199     return pixOp;
200 }
```

5.54.2.18 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes, HxTagList & *tags*, const *HxPixOpOutTag*, const *HxPixOpTransInVarTag*, const *HxPixOpNPhaseTag*) [inline]

Initialization for translation invariant, n phase pixel export operation.

```
209 {
210     PixOpT pixOp(tags);
211     return pixOp;
212 }
```

5.54.2.19 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes *sizes*, HxTagList & *tags*, const *HxPixOpOutTag*, const *HxPixOpTransVarTag*, const *HxPixOpNPhaseTag*) [inline]

Initialization for translation variant, n phase pixel export operation.

```
221 {
222     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
223     return pixOp;
224 }
```

5.54.2.20 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes, HxTagList & *tags*, const *HxPixOpInTag*, const *HxPixOpTransInVarTag*, const *HxPixOpNPhaseTag*) [inline]

Initialization for translation invariant, n phase pixel import operation.

```
233 {
234     PixOpT pixOp(tags);
235     return pixOp;
236 }
```

5.54.2.21 template<class PixOpT> PixOpT HxFuncInOutPixOpInit (HxSizes *sizes*, HxTagList & *tags*, const *HxPixOpInTag*, const *HxPixOpTransVarTag*, const *HxPixOpNPhaseTag*) [inline]

Initialization for translation variant, n phase pixel import operation.

```
245 {
246     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
247     return pixOp;
248 }
```

5.54.2.22 template<class DataPtrT, class PixOpT> void HxFuncInOutPixOpInit (HxSizes *sizes*, HxTagList & *tags*) [inline]

Dispatch function for HxFuncInOutPixOpInit.

Dispatch is based on the categories defined in PixOpT.

```
257 {
258     HxFuncInOutPixOp(
259         sizes, tags,
260         typename PixOpT::DirectionCategory(),
261         typename PixOpT::TransVarianceCategory(),
262         typename PixOpT::PhaseCategory());
263 }
```

5.55 HxGauss.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGauss** (**HxImageRep** img, double sigma, double accuracy=3.0)

  Convolution Gaussian.

5.55.1 Detailed Description

5.55.2 Function Documentation

5.55.2.1 HxImageRep L_HXIMAGEREP HxGauss (HxImageRep *img*, double *sigma*, double *accuracy* = 3.0)

Convolution Gaussian.

Equivalent to : img.genConvSeparated(gauss, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gauss is the 1d double-precision Gaussian kernel based on sigma and accuracy.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
18 {
19     int minSize = HxImageMinSize(img);
20     HxImageRep gauss = HxMakeGaussian1d(sigma, 0, accuracy, minSize);
21     return img.genConvSeparated(
22         gauss, "mul", "addAssign", HxImageRep::ARITH_PREC);
23 }
```

5.56 HxGaussDerivative2d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGaussDerivative2d** (**HxImageRep** img, double sigma, int orderDerivx, int orderDerivy, double accuracy=3.0)

  Convolution Gaussian.

5.56.1 Detailed Description

5.56.2 Function Documentation

5.56.2.1 HxImageRep L_HXIMAGEREP HxGaussDerivative2d (HxImageRep *img*, double *sigma*, int *orderDerivx*, int *orderDerivy*, double *accuracy* = 3.0)

Convolution Gaussian.

Equivalent to : img.genConvSeparated(gauss, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gauss is the 1d double-precision Gaussian kernel based on sigma, orderDeriv, accuracy, for each dimension.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
18 {
19     HxImageRep gaussx, gaussy;
20 
21     gaussx = HxMakeGaussian1d(sigma, orderDerivx, accuracy,
22         img.dimensionSize(1));
23     gaussy = HxMakeGaussian1d(sigma, orderDerivy, accuracy,
24         img.dimensionSize(2));
25 
26     return img.genConvSeparated(
27         1, gaussx, gaussy, "mul", "addAssign", HxImageRep::ARITH_PREC);
28 }
```

5.57 HxGaussDerivative3d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGaussDerivative3d** (**HxImageRep** img, double sigma, int orderDerivx, int orderDerivy, int orderDerivz, double accuracy=3.0)

  Convolution Gaussian.

5.57.1 Detailed Description

5.57.2 Function Documentation

5.57.2.1 HxImageRep L_HXIMAGEREP HxGaussDerivative3d (HxImageRep *img*, double *sigma*, int *orderDerivx*, int *orderDerivy*, int *orderDerivz*, double *accuracy* = 3.0)

Convolution Gaussian.

Equivalent to : img.genConvSeparated(gauss, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gauss is the 1d double-precision Gaussian kernel based on sigma, orderDeriv, accuracy, for each dimension.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
18 {
19     HxImageRep gaussx, gaussy, gaussz;
20
21     gaussx = HxMakeGaussian1d(sigma, orderDerivx, accuracy,
22         img.dimensionSize(1));
23     gaussy = HxMakeGaussian1d(sigma, orderDerivy, accuracy,
24         img.dimensionSize(2));
25     gaussz = HxMakeGaussian1d(sigma, orderDerivz, accuracy,
26         img.dimensionSize(3));
27
28     HxImageRep res = img.generalizedConvolutionK1d(1, gaussx,
29         "mul", "addAssign", HxImageRep::ARITH_PREC);
30     res = res.generalizedConvolutionK1d(2, gaussy,
31         "mul", "addAssign", HxImageRep::ARITH_PREC);
32     return res.generalizedConvolutionK1d(3, gaussz,
33         "mul", "addAssign", HxImageRep::ARITH_PREC);
34 }
```

5.58 HxGeoIntType.h File Reference

```
#include "HxIoFwd.h"
#include "HxString.h"
```

Enumerations

- enum **HxGeoIntType** { **NEAREST**, **LINEAR** }

  Enumeration of geometric interpolation types.

Functions

- std::ostream & **HxGeoIntType_put** (std::ostream &os, **HxGeoIntType** val)
- std::ostream & **operator<<** (std::ostream &os, **HxGeoIntType** val)
- **HxString makeString** (**HxGeoIntType** val)

5.58.1 Detailed Description

5.58.2 Enumeration Type Documentation

5.58.2.1 enum HxGeoIntType

Enumeration of geometric interpolation types.

Currently: NEAREST, LINEAR

```
23 { NEAREST, LINEAR };
```

5.59 HxGetPoints.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxGetPoints** (**HxImageRep** img, **HxPointListBackInserter** ptPtr)

5.59.1 Detailed Description

5.60 HxGetValues.h File Reference

```
#include "HxImageRep.h"
```

Functions

- void L_HXIMAGEREP **HxGetValues** (**HxImageRep** img, **HxPointListConstIter** first, **HxPointList-ConstIter** last, **HxValueListBackInserter** valPtr)

5.60.1 Detailed Description

5.61 HxGreaterEqual.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGreaterEqual** (**HxImageRep** im1, **HxImageRep** im2)

  Greater equal.

5.61.1 Detailed Description

5.61.2 Function Documentation

5.61.2.1 HxImageRep L_HXIMAGEREP HxGreaterEqual (HxImageRep *im1*, HxImageRep *im2*)

Greater equal.

The function performs greater equal (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoGreaterEqual** (p. 177). The image functor instantiator : **HxInstantiatorGreaterEqual** (p. 514).

```
13 {
14     return im1.binaryPixOp(im2, "greaterEqual");
15 }
```

5.62 HxGreaterEqualVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGreaterEqualVal** (**HxImageRep** im, **HxValue** val)

  Greater equal.

5.62.1 Detailed Description

5.62.2 Function Documentation

5.62.2.1 HxImageRep L_HXIMAGEREP HxGreaterEqualVal (HxImageRep *im*, HxValue *val*)

Greater equal.

The function performs greater equal (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoGreaterEqual** (p. 177). The image functor instantiator : **HxInstantiatorGreaterEqualV** (p. 515).

```
13 {
14     return im.binaryPixOp(val, "greaterEqual");
15 }
```

5.63 HxGreaterThan.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGreaterThan** (**HxImageRep** im1, **HxImageRep** im2)
  Greater than.

5.63.1 Detailed Description

5.63.2 Function Documentation

5.63.2.1 HxImageRep L_HXIMAGEREP HxGreaterThan (HxImageRep *im1*, HxImageRep *im2*)

Greater than.

The function performs greater than (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoGreaterThan** (p. 179). The image functor instantiator : **HxInstantiatorGreaterThan** (p. 515).

```
13 {
14     return im1.binaryPixOp(im2, "greaterThan");
15 }
```

5.64 HxGreaterThanVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGreaterThanVal** (**HxImageRep** im, **HxValue** val)
  Greater than.

5.64.1 Detailed Description

5.64.2 Function Documentation

5.64.2.1 HxImageRep L_HXIMAGEREP HxGreaterThanVal (HxImageRep *im*, HxValue *val*)

Greater than.

The function performs greater than (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoGreaterThan** (p. 179). The image functor instantiator : **HxInstantiatorGreaterThanV** (p. 516).

```
13 {
14     return im.binaryPixOp(val, "greaterThan");
15 }
```

5.65 HxIdentMaskMean.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxIdentMaskMean** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p, **HxSizes** size, int label)

5.65.1 Detailed Description

5.66 HxIdentMaskStDev.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxIdentMaskStDev** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p, **HxSizes** size, int label)

5.66.1 Detailed Description

5.67 HxIdentMaskSum.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxIdentMaskSum** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p, **HxSizes** size, int label)

5.67.1 Detailed Description

5.68 HxImageAsByte.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsByte** (**HxImageRep** img)

Convert the pixel representation to HxByte.

5.68.1 Detailed Description

5.68.2 Function Documentation

5.68.2.1 HxImageRep L_HXIMAGEREP HxImageAsByte (HxImageRep *img*)

Convert the pixel representation to HxByte.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DBYTE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.69 HxImageAsComplex.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsComplex** (**HxImageRep** img)

  Convert the pixel representation to **HxComplex** (p. 239).

5.69.1 Detailed Description

5.69.2 Function Documentation

5.69.2.1 HxImageRep L_HXIMAGEREP HxImageAsComplex (HxImageRep *img*)

Convert the pixel representation to **HxComplex** (p. 239).

Conversion is done via a cast.

```
15 {
16     HxImageSignature signature(HXIMAGESIG2DCOMPLEX);
17     signature.setImageDimensionality(img.dimensionality());
18     return img.signature() == signature ?
19             img : HxImageFactory::instance().fromImage(signature, img);
20 }
```

5.70 HxImageAsDouble.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsDouble** (**HxImageRep** img)

  Convert the pixel representation to double.

5.70.1 Detailed Description

5.70.2 Function Documentation

5.70.2.1 HxImageRep L_HXIMAGEREP HxImageAsDouble (HxImageRep *img*)

Convert the pixel representation to double.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DDOUBLE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.71 HxImageAsFloat.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsFloat** (**HxImageRep** img)

  Convert the pixel representation to float.

5.71.1 Detailed Description

5.71.2 Function Documentation

5.71.2.1 HxImageRep L_HXIMAGEREP HxImageAsFloat (HxImageRep *img*)

Convert the pixel representation to float.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DFLOAT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.72 HxImageAsInt.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsInt** (**HxImageRep** img)

  Convert the pixel representation to int.

5.72.1 Detailed Description

5.72.2 Function Documentation

5.72.2.1 HxImageRep L_HXIMAGEREP HxImageAsInt (HxImageRep *img*)

Convert the pixel representation to int.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DINT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.73 HxImageAsShort.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsShort** (**HxImageRep** img)

  Convert the pixel representation to short.

5.73.1 Detailed Description

5.73.2 Function Documentation

5.73.2.1 HxImageRep L_HXIMAGEREP HxImageAsShort (HxImageRep *img*)

Convert the pixel representation to short.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DSHORT);
16     signature.setImageDimensionality(img.dimensionality());
```

```
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.74 HxImageAsVec2Byte.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Byte** (**HxImageRep** img)

  Convert the pixel representation to HxVec2Byte.

5.74.1 Detailed Description

5.74.2 Function Documentation

5.74.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Byte (HxImageRep *img*)

Convert the pixel representation to HxVec2Byte.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC2BYTE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.75 HxImageAsVec2Double.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Double** (**HxImageRep** img)

  Convert the pixel representation to **HxVec2Double** (p. 766).

5.75.1 Detailed Description

5.75.2 Function Documentation

5.75.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Double (HxImageRep *img*)

Convert the pixel representation to **HxVec2Double** (p. 766).

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC2DOUBLE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.76 HxImageAsVec2Float.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Float** (**HxImageRep** img)

  Convert the pixel representation to HxVec2Float.

5.76.1 Detailed Description

5.76.2 Function Documentation

5.76.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Float (HxImageRep *img*)

Convert the pixel representation to HxVec2Float.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC2FLOAT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.77 HxImageAsVec2Int.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Int** (**HxImageRep** img)

  Convert the pixel representation to **HxVec2Int** (p. 785).

5.77.1 Detailed Description

5.77.2 Function Documentation

5.77.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Int (HxImageRep *img*)

Convert the pixel representation to **HxVec2Int** (p. 785).

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC2INT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.78 HxImageAsVec2Short.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Short** (**HxImageRep** img)

  Convert the pixel representation to HxVec2Short.

5.78.1 Detailed Description

5.78.2 Function Documentation

5.78.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Short (HxImageRep *img*)

Convert the pixel representation to HxVec2Short.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC2SHORT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.79 HxImageAsVec3Byte.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Byte** (**HxImageRep** img)

  Convert the pixel representation to HxVec3Byte.

5.79.1 Detailed Description

5.79.2 Function Documentation

5.79.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Byte (HxImageRep *img*)

Convert the pixel representation to HxVec3Byte.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC3BYTE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.80 HxImageAsVec3Double.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Double** (**HxImageRep** img)

  Convert the pixel representation to **HxVec3Double** (p. 804).

5.80.1 Detailed Description

5.80.2 Function Documentation

5.80.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Double (HxImageRep *img*)

Convert the pixel representation to **HxVec3Double** (p. 804).

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC3DOUBLE);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.81 HxImageAsVec3Float.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Float** (**HxImageRep** img)

  Convert the pixel representation to HxVec3Float.

5.81.1 Detailed Description

5.81.2 Function Documentation

5.81.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Float (HxImageRep *img*)

Convert the pixel representation to HxVec3Float.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC3FLOAT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.82 HxImageAsVec3Int.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Int** (**HxImageRep** img)

  Convert the pixel representation to **HxVec3Int** (p. 824).

5.82.1 Detailed Description

5.82.2 Function Documentation

5.82.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Int (HxImageRep *img*)

Convert the pixel representation to **HxVec3Int** (p. 824).

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC3INT);
16     signature.setImageDimensionality(img.dimensionality());
```

```
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.83 HxImageAsVec3Short.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Short** (**HxImageRep** img)

  Convert the pixel representation to HxVec3Short.

5.83.1 Detailed Description

5.83.2 Function Documentation

5.83.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Short (HxImageRep *img*)

Convert the pixel representation to HxVec3Short.

Conversion is done via a cast.

```
14 {
15     HxImageSignature signature(HXIMAGESIG2DVEC3SHORT);
16     signature.setImageDimensionality(img.dimensionality());
17     return img.signature() == signature ?
18             img : HxImageFactory::instance().fromImage(signature, img);
19 }
```

5.84 HxImageMaxSize.h File Reference

```
#include "HxImageRep.h"
```

Functions

- int L_HXIMAGEREP **HxImageMaxSize** (**HxImageRep** img)

  Maximum (one dimensional) size.

5.84.1 Detailed Description

5.84.2 Function Documentation

5.84.2.1 int L_HXIMAGEREP HxImageMaxSize (HxImageRep *img*)

Maximum (one dimensional) size.

Returns Maximum(img.dimensionSize(1), ..., img.dimensionSize(n)) where n == img.dimensionality().

```
13 {
14     int i, d = img.dimensionality(), s = img.dimensionSize(1);
15
16     for (i=2; i<=d; i++)
17         if (img.dimensionSize(i) > s)
18             s = img.dimensionSize(i);
19
20     return s;
21 }
```

5.85 HxImageMinSize.h File Reference

```
#include "HxImageRep.h"
```

Functions

- int L_HXIMAGEREP **HxImageMinSize** (**HxImageRep** img)

  Minimum (one dimensional) size.

5.85.1 Detailed Description

5.85.2 Function Documentation

5.85.2.1 int L_HXIMAGEREP HxImageMinSize (HxImageRep *img*)

Minimum (one dimensional) size.

Returns minimum(img.dimensionSize(1), ..., img.dimensionSize(n)) where n == img.dimensionality().

```
13 {
14     int i, d = img.dimensionality(), s = img.dimensionSize(1);
15
16     for (i=2; i<=d; i++)
17         if (img.dimensionSize(i) < s)
18             s = img.dimensionSize(i);
19
20     return s;
21 }
```

5.86 HxImagesFromFile.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageList** L_HXIMAGEREP **HxImagesFromFile** (**HxString** fileName)

  Write images to file using HxImgFileIo.

5.86.1 Detailed Description

5.86.2 Function Documentation

5.86.2.1 HxImageList L_HXIMAGEREP HxImagesFromFile (HxString *fileName*)

Write images to file using HxImgFileIo.

```
15 {
16     HxTagList tags;
17     return HxImageFactory::instance().imagesFromFile(fileName, tags);
18 }
```

5.87 HxImagesToFile.h File Reference

```
#include "HxImageRep.h"
```

Functions

- bool L_HXIMAGEREP **HxImagesToFile** (**HxImageList** ims, **HxString** fileName)

  Write images to file using HxImgFileIo.

5.87.1 Detailed Description

5.87.2 Function Documentation

5.87.2.1 bool L_HXIMAGEREP HxImagesToFile (HxImageList *ims*, HxString *fileName*)

Write images to file using HxImgFileIo.

```
15 {
16     HxTagList tags;
17     return HxImageFactory::instance().imagesToFile(ims, fileName, tags);
18 }
```

5.88 HxInf.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxInf** (**HxImageRep** im1, **HxImageRep** im2)

  Inifimum.

5.88.1 Detailed Description

5.88.2 Function Documentation

5.88.2.1 HxImageRep L_HXIMAGEREP HxInf (HxImageRep *im1*, HxImageRep *im2*)

Inifimum.

The function performs infimum (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoInf** (p. 180). The image functor instantiator : **HxInstantiatorInf** (p. 516).

```
14 {
15     return im1.binaryPixOp(im2, "inf");
16 }
```

5.89 HxInfVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxInfVal** (**HxImageRep** im, **HxValue** val)

  Inifimum.

5.89.1 Detailed Description

5.89.2 Function Documentation

5.89.2.1 HxImageRep L_HXIMAGEREP HxInfVal (HxImageRep *im*, HxValue *val*)

Inifimum.

The function performs infimum (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoInf** (p. 180). The image functor instantiator : **HxInstantiatorInfV** (p. 518).

```
13 {
14     return im.binaryPixOp(val, "inf");
15 }
```

5.90 HxInverseProjectRange.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxInverseProjectRange** (**HxImageRep** im, int dimension, **HxImageRep** arg)

  Inverse projection of the pixel range.

5.90.1 Detailed Description

5.90.2 Function Documentation

5.90.2.1 HxImageRep L_HXIMAGEREP HxInverseProjectRange (HxImageRep *im*, int *dimension*, HxImageRep *arg*)

Inverse projection of the pixel range.

The function projects (see **Pixels** (p. 5)) all pixels of image im on the given dimension of image arg via a unary pixel operation (see **Images** (p. 9)). Dimension starts at 1.

```
13 {
14     return im.inverseProjectRange(dimension, arg);
15 }
```

5.91 HxLeftShift.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLeftShift** (**HxImageRep** im1, **HxImageRep** im2)

  Left shift.

5.91.1 Detailed Description

5.91.2 Function Documentation

5.91.2.1 HxImageRep L_HXIMAGEREP HxLeftShift (HxImageRep *im1*, HxImageRep *im2*)

Left shift.

The function performs left shift (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLeftShift** (p. 182). The image functor instantiator : **HxInstantiatorLeftShift** (p. 518).

```
13 {
14     return im1.binaryPixOp(im2, "leftShift");
15 }
```

5.92 HxLeftShiftVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLeftShiftVal** (**HxImageRep** im, **HxValue** val)
  Left shift.

5.92.1 Detailed Description

5.92.2 Function Documentation

5.92.2.1 HxImageRep L_HXIMAGEREP HxLeftShiftVal (HxImageRep *im*, HxValue *val*)

Left shift.

The function performs left shift (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLeftShift** (p. 182). The image functor instantiator : **HxInstantiatorLeftShiftV** (p. 519).

```
13 {
14     return im.binaryPixOp(val, "leftShift");
15 }
```

5.93 HxLessEqual.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLessEqual** (**HxImageRep** im1, **HxImageRep** im2)
  Less equal.

5.93.1 Detailed Description

5.93.2 Function Documentation

5.93.2.1 HxImageRep L_HXIMAGEREP HxLessEqual (HxImageRep *im1*, HxImageRep *im2*)

Less equal.

The function performs less equal (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLessEqual** (p. 183). The image functor instantiator : **HxInstantiatorLessEqual** (p. 520).

```
13 {
14     return im1.binaryPixOp(im2, "lessEqual");
15 }
```

5.94 HxLessEqualVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLessEqualVal** (**HxImageRep** im, **HxValue** val)
  Less equal.

5.94.1 Detailed Description

5.94.2 Function Documentation

5.94.2.1 HxImageRep L_HXIMAGEREP HxLessEqualVal (HxImageRep *im*, HxValue *val*)

Less equal.

The function performs less equal (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLessEqual** (p. 183). The image functor instantiator : **HxInstantiatorLessEqualV** (p. 520).

```
13 {
14     return im.binaryPixOp(val, "lessEqual");
15 }
```

5.95 HxLessThan.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLessThan** (**HxImageRep** im1, **HxImageRep** im2)
  Less than.

5.95.1 Detailed Description

5.95.2 Function Documentation

5.95.2.1 HxImageRep L_HXIMAGEREP HxLessThan (HxImageRep *im1*, HxImageRep *im2*)

Less than.

The function performs less than (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLessThan** (p. 185). The image functor instantiator : **HxInstantiatorLessThan** (p. 521).

```
13 {
14     return im1.binaryPixOp(im2, "lessThan");
15 }
```

5.96 HxLessThanVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLessThanVal** (**HxImageRep** im, **HxValue** val)

  Less than.

5.96.1 Detailed Description

5.96.2 Function Documentation

5.96.2.1 HxImageRep L_HXIMAGEREP HxLessThanVal (HxImageRep *im*, HxValue *val*)

Less than.

The function performs less than (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoLessThan** (p. 185). The image functor instantiator : **HxInstantiatorLessThanV** (p. 521).

```
13 {
14     return im.binaryPixOp(val, "lessThan");
15 }
```

5.97 HxLog.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLog** (**HxImageRep** im)

  Natural logarithm.

5.97.1 Detailed Description

5.97.2 Function Documentation

5.97.2.1 HxImageRep L_HXIMAGEREP HxLog (HxImageRep *im*)

Natural logarithm.

The function computes the natural logarithm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoLog** (p. 736). The image functor instantiator : **HxInstantiatorLog** (p. 522).

```
13 {
14     return im.unaryPixOp("log");
15 }
```

5.98 HxLog10.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLog10** (**HxImageRep** im)

  Base 10 logarithm.

5.98.1 Detailed Description

5.98.2 Function Documentation

5.98.2.1 HxImageRep L_HXIMAGEREP HxLog10 (HxImageRep *im*)

Base 10 logarithm.

The function computes the base 10 logarithm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoLog10** (p. 737). The image functor instantiator : **HxInstantiatorLog10** (p. 523).

```
13 {
14     return im.unaryPixOp("log10");
15 }
```

5.99 HxMakeFrom2Images.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFrom2Images** (**HxImageRep** i1, **HxImageRep** i2)

  Make a new image with pixel values "stacked" from given arguments.

5.99.1 Detailed Description

5.99.2 Function Documentation

5.99.2.1 HxImageRep L_HXIMAGEREP HxMakeFrom2Images (HxImageRep *i1*, HxImageRep *i2*)

Make a new image with pixel values "stacked" from given arguments.

For example, if i1 and i2 are scalar images the pixel values in the new image are 2-vectors. Result may need exceed highest pixel dimensionality.

```
14 {
15     return HxImageFactory::instance().from2Images(i1, i2);
16 }
```

5.100 HxMakeFrom3Images.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFrom3Images** (**HxImageRep** i1, **HxImageRep** i2, **HxImageRep** i3)

  Make a new image with pixel values "stacked" from given arguments.

5.100.1 Detailed Description

5.100.2 Function Documentation

5.100.2.1 HxImageRep L_HXIMAGEREP HxMakeFrom3Images (HxImageRep *i1*, HxImageRep *i2*, HxImageRep *i3*)

Make a new image with pixel values "stacked" from given arguments.

For example, if i1, i2, and i3 are scalar images the pixel values in the new image are 3-vectors. Result may need exceed highest pixel dimensionality.

```
15 {
16     return HxImageFactory::instance().from3Images(i1, i2, i3);
17 }
```

5.101 HxMakeFromByteData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromByteData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, HxByte *data)

  Make a new image with given signature and sizes.

5.101.1 Detailed Description

5.101.2 Function Documentation

5.101.2.1 HxImageRep L_HXIMAGEREP HxMakeFromByteData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, HxByte * *data*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromByteData(pixelDimensionality,
17                 dimensions, sizes, data);
18 }
```

5.102 HxMakeFromDoubleData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromDoubleData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, double *data)

  Make a new image with given signature and sizes.

5.102.1 Detailed Description

5.102.2 Function Documentation

5.102.2.1 HxImageRep L_HXIMAGEREP HxMakeFromDoubleData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, double * *data*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromDoubleData(pixelDimensionality,
17                 dimensions, sizes, data);
18 }
```

5.103 HxMakeFromFile.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromFile** (**HxString** fileName)
  New image read from file using HxImgFileIo lib.

5.103.1 Detailed Description

5.103.2 Function Documentation

5.103.2.1 HxImageRep L_HXIMAGEREP HxMakeFromFile (HxString *fileName*)

New image read from file using HxImgFileIo lib.

```
14 {
15     HxTagList tags;
16     return HxImageFactory::instance().fromFile(fileName, tags);
17 }
```

5.104 HxMakeFromFileSI.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromFileSI** (**HxString** fileName)
  New image read from file using scil-image.

5.104.1 Detailed Description

5.104.2 Function Documentation

5.104.2.1 HxImageRep L_HXIMAGEREP HxMakeFromFileSI (HxString *fileName*)

New image read from file using scil-image.

```
14 {
15     return HxImageFactory::instance().fromFile(fileName);
16 }
```

5.105 HxMakeFromFloatData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromFloatData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, float *data)

  Make a new image with given signature and sizes.

5.105.1 Detailed Description

5.105.2 Function Documentation

5.105.2.1 HxImageRep L_HXIMAGEREP HxMakeFromFloatData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, float * *data*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromFloatData(pixelDimensionality,
17                 dimensions, sizes, data);
18 }
```

5.106 HxMakeFromGenerator.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromGenerator** (const **HxImageSignature** &signature, const HxImageGenerator *imgGenerator)

  Make a new image with given signature.

5.106.1 Detailed Description

5.106.2 Function Documentation

5.106.2.1 HxImageRep L_HXIMAGEREP HxMakeFromGenerator (const HxImageSignature & *signature*, const HxImageGenerator * *imgGenerator*)

Make a new image with given signature.

Pixel data and size are determined by image generator.

```
15 {
16     return HxImageFactory::instance().fromGenerator(signature, imgGenerator);
17 }
```

5.107 HxMakeFromGrayValue.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromGrayValue** (const **HxImageSignature** &signature, **HxSizes** sizes, HxByte *pixels)

  Make a new image with given signature and sizes.

5.107.1 Detailed Description

5.107.2 Function Documentation

5.107.2.1 HxImageRep L_HXIMAGEREP HxMakeFromGrayValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxByte * *pixels*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromGrayValue(signature, sizes, pixels);
17 }
```

5.108 HxMakeFromImage.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromImage** (const **HxImageSignature** &signature, **HxImageRep** src)

  Make a new image with given signature.

5.108.1 Detailed Description

5.108.2 Function Documentation

5.108.2.1 HxImageRep L_HXIMAGEREP HxMakeFromImage (const HxImageSignature & *signature*, HxImageRep *src*)

Make a new image with given signature.

Sizes and data are taken from given image.

```
14 {
15     return HxImageFactory::instance().fromImage(signature, src);
16 }
```

5.109 HxMakeFromImport.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromImport** (const **HxImageSignature** &signature, **HxSizes** sizes, **HxString** importOp, **HxTagList** &tags)

  Make a new image with given signature and sizes.

5.109.1 Detailed Description

5.109.2 Function Documentation

5.109.2.1 HxImageRep L_HXIMAGEREP HxMakeFromImport (const HxImageSignature & *signature*, HxSizes *sizes*, HxString *importOp*, HxTagList & *tags*)

Make a new image with given signature and sizes.

Pixel values are initialized by specified import operator.

```
15 {
16     return HxImageFactory::instance().fromImport(signature, sizes, importOp, tags);
17 }
```

5.110 HxMakeFromIntData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromIntData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, int *data)

Make a new image with given signature and sizes.

5.110.1 Detailed Description

5.110.2 Function Documentation

5.110.2.1 HxImageRep L_HXIMAGEREP HxMakeFromIntData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, int * *data*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromIntData(
17             pixelDimensionality, dimensions, sizes, data);
18 }
```

5.111 HxMakeFromJavaRgb.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromJavaRgb** (const **HxImageSignature** &signature, **HxSizes** sizes, int *pixels)

  Make a new image with given signature and sizes.

5.111.1 Detailed Description

5.111.2 Function Documentation

5.111.2.1 HxImageRep L_HXIMAGEREP HxMakeFromJavaRgb (const HxImageSignature & *signature*, HxSizes *sizes*, int * *pixels*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values. The given values are stored in Java RGB format.

```
15 {
16     return HxImageFactory::instance().fromJavaRgb(signature, sizes, pixels);
17 }
```

5.112 HxMakeFromMatlab.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromMatlab** (const **HxImageSignature** &signature, **HxSizes** sizes, double *pixels)

  Make a new image with given signature and sizes.

5.112.1 Detailed Description

5.112.2 Function Documentation

5.112.2.1 HxImageRep L_HXIMAGEREP HxMakeFromMatlab (const HxImageSignature & *signature*, HxSizes *sizes*, double * *pixels*)

Make a new image with given signature and sizes.

Pixel values are initialized from given values. The given values are stored in Matlab format.

```
15 {
16     return HxImageFactory::instance().fromMatlab(signature, sizes, pixels);
17 }
```

5.113 HxMakeFromNamedGenerator.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromNamedGenerator** (const **HxImageSignature** &signature, **HxString** generatorName, **HxTagList** &tags)

  Make a new image with given signature.

5.113.1 Detailed Description

5.113.2 Function Documentation

5.113.2.1 HxImageRep L_HXIMAGEREP HxMakeFromNamedGenerator (const HxImageSignature & *signature*, HxString *generatorName*, HxTagList & *tags*)

Make a new image with given signature.

Pixel data and size are determined by image generator.

```
15 {
16     return HxImageFactory::instance().fromNamedGenerator(signature,
17                 generatorName, tags);
18 }
```

5.114 HxMakeFromPpmPixels.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromPpmPixels** (const **HxImageSignature** &signature, **HxSizes** sizes, const HxByte *pixels)

5.114.1 Detailed Description

5.115 HxMakeFromShortData.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromShortData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, short *data)

  Make a new image with given signature and sizes.

5.115.1 Detailed Description

5.115.2 Function Documentation

5.115.2.1 HxImageRep L_HXIMAGEREP HxMakeFromShortData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, short * *data*)

Make a new image with given signature and sizes.

Pixel data is initialized from given values.

```
15 {
16     return HxImageFactory::instance().fromShortData(pixelDimensionality,
17                 dimensions, sizes, data);
18 }
```

5.116 HxMakeFromSi.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromSi** (IMAGE *im)

  Make an **HxImageRep** (p. 313) *with from the given image in SciImage format.*

5.116.1 Detailed Description

5.116.2 Function Documentation

5.116.2.1 HxImageRep L_HXIMAGEREP HxMakeFromSi (IMAGE * *im*)

Make an **HxImageRep** (p. 313) with from the given image in ScilImage format.

```
13 {
14     return HxImageRep(im);
15 }
```

5.117 HxMakeFromSignature.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromSignature** (const **HxImageSignature** &signature, **HxSizes** sizes)

  Make an uninitialized image with given signature and sizes.

5.117.1 Detailed Description

5.117.2 Function Documentation

5.117.2.1 HxImageRep L_HXIMAGEREP HxMakeFromSignature (const HxImageSignature & *signature*, HxSizes *sizes*)

Make an uninitialized image with given signature and sizes.

```
14 {
15     return HxImageFactory::instance().fromSignature(signature, sizes);
16 }
```

5.118 HxMakeFromValue.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromValue** (const **HxImageSignature** &signature, **HxSizes** sizes, **HxValue** val)

  Make a new image with given signature and sizes.

5.118.1 Detailed Description

5.118.2 Function Documentation

5.118.2.1 HxImageRep L_HXIMAGEREP HxMakeFromValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxValue *val*)

Make a new image with given signature and sizes.

Pixel data is initialized with given value.

```
14 {
15     return HxImageFactory::instance().fromValue(signature, sizes, val);
16 }
```

5.119 HxMakeGaussian1d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeGaussian1d** (double sigma, int deri, double acc, int maxfsize, int fsize=-1)

  Generate a kernel image resembling a Gaussian.

5.119.1 Detailed Description

5.119.2 Function Documentation

5.119.2.1 HxImageRep L_HXIMAGEREP HxMakeGaussian1d (double *sigma*, int *deri*, double *acc*, int *maxfsize*, int *fsize* = -1)

Generate a kernel image resembling a Gaussian.

```
15 {
16     HxTagList   tags;
17
18     HxAddTag(tags, "sigma", sigma);
19     HxAddTag(tags, "derivative", deri);
20     HxAddTag(tags, "accuracy", acc);
21     HxAddTag(tags, "size", fsize);
22     HxAddTag(tags, "maxSize", maxfsize);
23
24     return HxImageFactory::instance().fromNamedGenerator(
25             HXIMAGESIG2DDOUBLE, "gauss1d", tags);
26 }
```

5.120 HxMakeParabola1d.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeParabola1d** (double rho, double accuracy, int maxfsize, int fsize=-1)

  Generate a kernel image resembling a parabola.

5.120.1 Detailed Description

5.120.2 Function Documentation

5.120.2.1 HxImageRep L_HXIMAGEREP HxMakeParabola1d (double *rho*, double *accuracy*, int *maxfsize*, int *fsize* = -1)

Generate a kernel image resembling a parabola.

```
15 {
16     HxTagList   tags;
17
18     HxAddTag(tags, "rho", HxVec3Double(rho,rho,rho));
19     HxAddTag(tags, "accuracy", acc);
20     HxAddTag(tags, "size", fsize);
21     HxAddTag(tags, "maxSize", maxfsize);
22
23     return HxImageFactory::instance().fromNamedGenerator(
24             HXIMAGESIG2DDOUBLE, "parabola1d", tags);
25 }
```

5.121 HxMaskSum.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxMaskSum** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p)

5.121.1 Detailed Description

5.122 HxMatrixConv.h File Reference

```
#include "HxMatrix.h"
```

Functions

- **HxMatrix** L_HXIMAGEREP **HxImageRepToMatrix** (**HxImageRep** im)
- **HxVector** L_HXIMAGEREP **HxImageRepToVector** (**HxImageRep** im)
- **HxImageRep** L_HXIMAGEREP **HxMatrixToImageRep** (**HxMatrix** &m)
- **HxImageRep** L_HXIMAGEREP **HxVectorToImageRep** (**HxVector** &v)

5.122.1 Detailed Description

5.123 HxMax.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMax** (**HxImageRep** im1, **HxImageRep** im2)

  Maximum.

5.123.1 Detailed Description

5.123.2 Function Documentation

5.123.2.1 HxImageRep L_HXIMAGEREP HxMax (HxImageRep *im1*, HxImageRep *im2*)

Maximum.

The function performs maximum (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMax** (p. 186). The image functor instantiator : **HxInstantiatorMax** (p. 523).

```
14 {
15     return im1.binaryPixOp(im2, "max");
16 }
```

5.124 HxMaxVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMaxVal** (**HxImageRep** im, **HxValue** val)

  Maximum.

5.124.1 Detailed Description

5.124.2 Function Documentation

5.124.2.1 HxImageRep L_HXIMAGEREP HxMaxVal (HxImageRep *im*, HxValue *val*)

Maximum.

The function performs maximum (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMax** (p. 186). The image functor instantiator : **HxInstantiatorMaxV** (p. 525).

```
13 {
14     return im.binaryPixOp(val, "max");
15 }
```

5.125 HxMin.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMin** (**HxImageRep** im1, **HxImageRep** im2)

  Minimum.

5.125.1 Detailed Description

5.125.2 Function Documentation

5.125.2.1 HxImageRep L_HXIMAGEREP HxMin (HxImageRep *im1*, HxImageRep *im2*)

Minimum.

The function performs minimum (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMin** (p. 188). The image functor instantiator : **HxInstantiatorMin** (p. 525).

```
14 {
15     return im1.binaryPixOp(im2, "min");
16 }
```

5.126 HxMinVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMinVal** (**HxImageRep** im, **HxValue** val)

  Minimum.

5.126.1 Detailed Description

5.126.2 Function Documentation

5.126.2.1 HxImageRep L_HXIMAGEREP HxMinVal (HxImageRep *im*, HxValue *val*)

Minimum.

The function performs minimum (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMin** (p. 188). The image functor instantiator : **HxInstantiatorMinV** (p. 526).

```
13 {
14     return im.binaryPixOp(val, "min");
15 }
```

5.127 HxMod.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMod** (**HxImageRep** im1, **HxImageRep** im2)

  Modulo.

5.127.1 Detailed Description

5.127.2 Function Documentation

5.127.2.1 HxImageRep L_HXIMAGEREP HxMod (HxImageRep *im1*, HxImageRep *im2*)

Modulo.

The function performs modulo (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMod** (p. 191). The image functor instantiator : **HxInstantiatorMod** (p. 527).

```
13 {
14     return im1.binaryPixOp(im2, "mod");
15 }
```

5.128 HxModVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxModVal** (**HxImageRep** im, **HxValue** val)

  Modulo.

5.128.1 Detailed Description

5.128.2 Function Documentation

5.128.2.1 HxImageRep L_HXIMAGEREP HxModVal (HxImageRep *im*, HxValue *val*)

Modulo.

The function performs modulo (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMod** (p. 191). The image functor instantiator : **HxInstantiatorModV** (p. 528).

```
13 {
14     return im.binaryPixOp(val, "mod");
15 }
```

5.129 HxMul.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMul** (**HxImageRep** im1, **HxImageRep** im2)

  Multiplication.

5.129.1 Detailed Description

5.129.2 Function Documentation

5.129.2.1 HxImageRep L_HXIMAGEREP HxMul (HxImageRep *im1*, HxImageRep *im2*)

Multiplication.

The function performs multiplication (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMul** (p. 192). The image functor instantiator : **HxInstantiatorMul** (p. 528).

```
13 {
14     return im1.binaryPixOp(im2, "mul");
15 }
```

5.130 HxMulVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMulVal** (**HxImageRep** im, **HxValue** val)
  Multiplication.

5.130.1 Detailed Description

5.130.2 Function Documentation

5.130.2.1 HxImageRep L_HXIMAGEREP HxMulVal (HxImageRep *im*, HxValue *val*)

Multiplication.

The function performs multiplication (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMul** (p. 192). The image functor instantiator : **HxInstantiatorMulV** (p. 530).

```
13 {
14     return im.binaryPixOp(val, "mul");
15 }
```

5.131 HxNegate.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNegate** (**HxImageRep** im)
  Negation.

5.131.1 Detailed Description

5.131.2 Function Documentation

5.131.2.1 HxImageRep L_HXIMAGEREP HxNegate (HxImageRep *im*)

Negation.

The function computes the negation (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoNegate** (p. 740). The image functor instantiator : **HxInstantiatorNegate** (p. 530).

```
13 {
14     return im.unaryPixOp("negate");
15 }
```

5.132 HxNonMaxSuppressionGradDir.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNonMaxSuppressionGradDir** (**HxImageRep** img)

  Non maxima suppression in the direction of the gradient.

5.132.1 Detailed Description

5.132.2 Function Documentation

5.132.2.1 HxImageRep L_HXIMAGEREP HxNonMaxSuppressionGradDir (HxImageRep *img*)

Non maxima suppression in the direction of the gradient.

Implementation specifics : The neighbourhood functor : **HxNgbNonMaxSuppression2d** (p. 621). The image functor instantiator : **HxNgbNonMaxSuppression2dInst** (p. 623).

```
13 {
14     return img.neighbourhoodOp("nonMaxSuppression");
15 }
```

5.133 HxNorm1.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm1** (**HxImageRep** im)

  L1 norm (city block distance).

5.133.1 Detailed Description

5.133.2 Function Documentation

5.133.2.1 HxImageRep L_HXIMAGEREP HxNorm1 (HxImageRep *im*)

L1 norm (city block distance).

The function computes the L1 norm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoNorm1** (p. 741). The image functor instantiator : **HxInstantiatorNorm1** (p. 531).

```
13 {
14     return im.unaryPixOp("norm1");
15 }
```

5.134 HxNorm2.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm2** (**HxImageRep** im)

  L2 norm (Euclidain norm).

5.134.1 Detailed Description

5.134.2 Function Documentation

5.134.2.1 HxImageRep L_HXIMAGEREP HxNorm2 (HxImageRep *im*)

L2 norm (Euclidain norm).

The function computes the L2 norm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoNorm2** (p. 743). The image functor instantiator : **HxInstantiatorNorm2** (p. 531).

```
13 {
14     return im.unaryPixOp("norm2");
15 }
```

5.135 HxNorm2Sqr.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm2Sqr** (**HxImageRep** im)

  Squared L2 norm (squared Euclidain norm).

5.135.1 Detailed Description

5.135.2 Function Documentation

5.135.2.1 HxImageRep L_HXIMAGEREP HxNorm2Sqr (HxImageRep *im*)

Squared L2 norm (squared Euclidain norm).

The function computes the squared L2 norm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoNorm2Sqr** (p. 744). The image functor instantiator : **HxInstantiatorNorm2Sqr** (p. 532).

```
13 {
14     return im.unaryPixOp("norm2sqr");
15 }
```

5.136 HxNormalizedCorrelation.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNormalizedCorrelation** (**HxImageRep** img, **HxImageRep** kernel)

  Normalized (cross) correlation.

5.136.1 Detailed Description

5.136.2 Function Documentation

5.136.2.1 HxImageRep L_HXIMAGEREP HxNormalizedCorrelation (HxImageRep *img*, HxImageRep *kernel*)

Normalized (cross) correlation.

Implementation specifics : The neighbourhood functor : **HxNgbNormCorrelation** (p. 623). The image functor instantiator : **HxNgbNc2dInst** (p. 621).

```
13 {
14     return img.neighbourhoodOp(kernel, "normalizedCorrelation");
15 }
```

5.137 HxNormInf.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNormInf** (**HxImageRep** im)

  L infinity norm (chessboard).

5.137.1 Detailed Description

5.137.2 Function Documentation

5.137.2.1 HxImageRep L_HXIMAGEREP HxNormInf (HxImageRep *im*)

L infinity norm (chessboard).

The function computes the L infinity norm (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoNormInf** (p. 745). The image functor instantiator : **HxInstantiatorNormInf** (p. 533).

```
13 {
14     return im.unaryPixOp("normInf");
15 }
```

5.138 HxNotEqual.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNotEqual** (**HxImageRep** im1, **HxImageRep** im2)

  Not equal.

5.138.1 Detailed Description

5.138.2 Function Documentation

5.138.2.1 HxImageRep L_HXIMAGEREP HxNotEqual (HxImageRep *im1*, HxImageRep *im2*)

Not equal.

The function performs not equal (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoNotEqual** (p. 194). The image functor instantiator : **HxInstantiatorNotEqual** (p. 533).

```
13 {
14     return im1.binaryPixOp(im2, "notEqual");
15 }
```

5.139 HxNotEqualVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxNotEqualVal** (**HxImageRep** im, **HxValue** val)

  Not equal.

5.139.1 Detailed Description

5.139.2 Function Documentation

5.139.2.1 HxImageRep L_HXIMAGEREP HxNotEqualVal (HxImageRep *im*, HxValue *val*)

Not equal.

The function performs not equal (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoNotEqual** (p. 194). The image functor instantiator : **HxInstantiatorNotEqualV** (p. 534).

```
13 {
14     return im.binaryPixOp(val, "notEqual");
15 }
```

5.140 HxOpponentColor.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpponentColor** (**HxImageRep** im)

5.140.1 Detailed Description

5.141 HxOr.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOr** (**HxImageRep** im1, **HxImageRep** im2)

  Or.

5.141.1 Detailed Description

5.141.2 Function Documentation

5.141.2.1 HxImageRep L_HXIMAGEREP HxOr (HxImageRep *im1*, HxImageRep *im2*)

Or.

The function performs or (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoOr** (p. 196). The image functor instantiator : **HxInstantiatorOr** (p. 535).

```
13 {
14     return im1.binaryPixOp(im2, "or");
15 }
```

5.142 HxOrVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOrVal** (**HxImageRep** im, **HxValue** val)

  Or.

5.142.1 Detailed Description

5.142.2 Function Documentation

5.142.2.1 HxImageRep L_HXIMAGEREP HxOrVal (HxImageRep *im*, HxValue *val*)

Or.

The function performs or (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoOr** (p. 196). The image functor instantiator : **HxInstantiatorOrV** (p. 535).

```
13 {
14     return im.binaryPixOp(val, "or");
15 }
```

5.143 HxParabolicDilation.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxParabolicDilation** (**HxImageRep** img, double rho, double accuracy=3.0)

  Parabolic dilation.

5.143.1 Detailed Description

5.143.2 Function Documentation

5.143.2.1 HxImageRep L_HXIMAGEREP HxParabolicDilation (HxImageRep *img*, double *rho*, double *accuracy* = 3.0)

Parabolic dilation.

Equivalent to : img.genConvSeparated(parabola, ”add”, ”maxAssign”, HxImageRep::ARITH_PREC)

where parabola is the 1d double-precision parabolic kernel based on rho and accuracy.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
17 {
18     int minSize = HxImageMinSize(img);
19     HxImageRep parabola = HxMakeParabola1d(rho, accuracy, minSize);
20     return img.genConvSeparated(
21         parabola, "add", "maxAssign", HxImageRep::ARITH_PREC);
22 }
```

5.144 HxParabolicErosion.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxParabolicErosion** (**HxImageRep** img, double rho, double accuracy=3.0)

  Parabolic erosion.

5.144.1 Detailed Description

5.144.2 Function Documentation

5.144.2.1 HxImageRep L_HXIMAGEREP HxParabolicErosion (HxImageRep *img*, double *rho*, double *accuracy* = 3.0)

Parabolic erosion.

Equivalent to : img.genConvSeparated(parabola, ”add”, ”minAssign”, HxImageRep::ARITH_PREC)

where parabola is the 1d double-precision parabolic kernel based on rho and accuracy.

Notice that the kernel is applied to every dimension of the image separately and that the result image has a double-precision pixel type.

```
17 {
18     int minSize = HxImageMinSize(img);
19     HxImageRep parabola = HxMakeParabola1d(-rho, accuracy, minSize);
20     return img.genConvSeparated(
21         parabola, "add", "minAssign", HxImageRep::ARITH_PREC);
22 }
```

5.145 HxPercentile.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxPercentile** (**HxImageRep** im, int neighSize, double perc)

  Percentile filter.

5.145.1 Detailed Description

5.145.2 Function Documentation

5.145.2.1 HxImageRep L_HXIMAGEREP HxPercentile (HxImageRep *im*, int *neighSize*, double *perc*)

Percentile filter.

Implementation specifics : The neighbourhood functor : **HxNgbPercentile2d** (p. 625). The image functor instantiator : **HxNgbPercentile2dInst** (p. 627).

```
14 {
15     HxTagList tags;
16     HxAddTag(tags, "size", neighSize);
17     HxAddTag(tags, "percentile", perc);
18 
19     return im.neighbourhoodOp("percentile", tags);
20 }
```

5.146 HxPixInf.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixInf** (**HxImageRep** im)

  Pixel infimum.

5.146.1 Detailed Description

5.146.2 Function Documentation

5.146.2.1 HxValue L_HXIMAGEREP HxPixInf (HxImageRep *im*)

Pixel infimum.

The function computes the infimum (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoInfAssign** (p. 181). The image functor instantiator : **HxInstantiatorInfReduce** (p. 517).

```
13 {
14     return im.reduceOp("infAssign");
15 }
```

5.147 HxPixMax.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixMax** (**HxImageRep** im)

  Pixel maximum.

5.147.1 Detailed Description

5.147.2 Function Documentation

5.147.2.1 HxValue L_HXIMAGEREP HxPixMax (HxImageRep *im*)

Pixel maximum.

The function computes the maximum (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMaxAssign** (p. 187). The image functor instantiator : **HxInstantiatorMaxReduce** (p. 524).

```
13 {
14     return im.reduceOp("maxAssign");
15 }
```

5.148 HxPixMin.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixMin** (**HxImageRep** im)

  Pixel minimum.

5.148.1 Detailed Description

5.148.2 Function Documentation

5.148.2.1 HxValue L_HXIMAGEREP HxPixMin (HxImageRep *im*)

Pixel minimum.

The function computes the minimum (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMinAssign** (p. 189). The image functor instantiator : **HxInstantiatorMinReduce** (p. 526).

```
13 {
14     return im.reduceOp("minAssign");
15 }
```

5.149 HxPixProduct.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixProduct** (**HxImageRep** im)

  Pixel product.

5.149.1 Detailed Description

5.149.2 Function Documentation

5.149.2.1 HxValue L_HXIMAGEREP HxPixProduct (HxImageRep *im*)

Pixel product.

The function computes the product (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoMulAssign** (p. 193). The image functor instantiator : **HxInstantiatorMulReduce** (p. 529).

```
13 {
14     return im.reduceOp("mulAssign");
15 }
```

5.150 HxPixSum.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixSum** (**HxImageRep** im)
  Pixel sum.

5.150.1 Detailed Description

5.150.2 Function Documentation

5.150.2.1 HxValue L_HXIMAGEREP HxPixSum (HxImageRep *im*)

Pixel sum.

The function computes the sum (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoAddAssign** (p. 170). The image functor instantiator : **HxInstantiatorAddReduce** (p. 499).

```
13 {
14     return im.reduceOp("addAssign");
15 }
```

5.151 HxPixSup.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxValue** L_HXIMAGEREP **HxPixSup** (**HxImageRep** im)
  Pixel supremum.

5.151.1 Detailed Description

5.151.2 Function Documentation

5.151.2.1 HxValue L_HXIMAGEREP HxPixSup (HxImageRep *im*)

Pixel supremum.

The function computes the supremum (see **Pixels** (p. 5)) of all pixels in the input image via a reduce operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoSupAssign** (p. 204). The image functor instantiator : **HxInstantiatorSupReduce** (p. 545).

```
13 {
14     return im.reduceOp("supAssign");
15 }
```

5.152 HxPoint.h File Reference

```
#include "HxVec3Double.h"
```

Typedefs

- typedef **HxVec3Double HxPoint**

  Definition of a point in 3D space.

5.152.1 Detailed Description

5.152.2 Typedef Documentation

5.152.2.1 typedef HxVec3Double HxPoint

Definition of a point in 3D space.

Uses floating point (double) coordinates.

5.153 HxPointInt.h File Reference

```
#include "HxVec3Int.h"
```

Typedefs

- typedef **HxVec3Int HxPointInt**

  Definition of a point in 3D space using integer coordinates.

5.153.1 Detailed Description

5.153.2 Typedef Documentation

5.153.2.1 typedef HxVec3Int HxPointInt

Definition of a point in 3D space using integer coordinates.

HxPointInt is used primarily to specify the position of a pixel in an image.

5.154 HxPointList.h File Reference

```
#include "HxPoint.h"
#include <list>
```

Compounds

- class **HxPointList**

  Class definition for list of HxPoint's.

Typedefs

- typedef HxPointList::iterator **HxPointListIter**

  Iterator for **HxPointList** (p. 640).

- typedef HxPointList::const_iterator **HxPointListConstIter**

  Const iterator for **HxPointList** (p. 640).

- typedef **HxPointList::back_insert_iterator HxPointListBackInserter**

  Back inserter for **HxPointList** (p. 640).

5.154.1 Detailed Description

5.154.2 Typedef Documentation

5.154.2.1 typedef HxPointList::iterator HxPointListIter

Iterator for **HxPointList** (p. 640).

5.154.2.2 typedef HxPointList::const_iterator HxPointListConstIter

Const iterator for **HxPointList** (p. 640).

5.154.2.3 typedef HxPointList::back_insert_iterator HxPointListBackInserter

Back inserter for **HxPointList** (p. 640).

5.155 HxPow.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxPow** (**HxImageRep** im1, **HxImageRep** im2)

  Power.

5.155.1 Detailed Description

5.155.2 Function Documentation

5.155.2.1 HxImageRep L_HXIMAGEREP HxPow (HxImageRep *im1*, HxImageRep *im2*)

Power.

The function performs power (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoPow** (p. 197). The image functor instantiator : **HxInstantiatorPow** (p. 536).

```
13 {
14     return im1.binaryPixOp(im2, "pow");
15 }
```

5.156 HxPowVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxPowVal** (**HxImageRep** im, **HxValue** val)

  Power.

5.156.1 Detailed Description

5.156.2 Function Documentation

5.156.2.1 HxImageRep L_HXIMAGEREP HxPowVal (HxImageRep *im*, HxValue *val*)

Power.

The function performs power (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoPow** (p. 197). The image functor instantiator : **HxInstantiatorPowV** (p. 536).

```
13 {
14     return im.binaryPixOp(val, "pow");
15 }
```

5.157 HxProjectRange.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxProjectRange** (**HxImageRep** im, int dimension)

  Projection of the pixel range.

5.157.1 Detailed Description

5.157.2 Function Documentation

5.157.2.1 HxImageRep L_HXIMAGEREP HxProjectRange (HxImageRep *im*, int *dimension*)

Projection of the pixel range.

The function computes the projection (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)). Dimension starts at 1.

```
13 {
14     return im.projectRange(dimension);
15 }
```

5.158 HxRecGauss.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRecGauss** (**HxImageRep** im, double sx, double sy, int dx=0, int dy=0, int recurOrder=3)

5.158.1 Detailed Description

5.159 HxReflect.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxReflect** (**HxImageRep** img, int doX, int doY, int doZ=1)

  Reflection.

5.159.1 Detailed Description

5.159.2 Function Documentation

5.159.2.1 HxImageRep L_HXIMAGEREP HxReflect (HxImageRep *img*, int *doX*, int *doY*, int *doZ* = 1)

Reflection.

```
14 {
15     if (img.dimensionality() == 2) {
16         HxMatrix m =
17             HxMatrix::translate2d(img.dimensionSize(1)/2 - 0.5,
18                                   img.dimensionSize(2)/2 - 0.5) *
19             HxMatrix::reflect2d(doX, doY) *
20             HxMatrix::translate2d(-img.dimensionSize(1)/2 + 0.5,
21                                   -img.dimensionSize(2)/2 + 0.5);
22         return img.geometricOp2d(m, NEAREST, FORWARD, 0, HxValue(0));
23     } else {
24         HxEnvironment::instance()->errorStream()
25             << "3d reflection not implemented yet" << STD_ENDL;
26         HxEnvironment::instance()->flush();
27         return HxImageRep();
28     }
29 }
```

5.160 HxRestrict.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRestrict** (**HxImageRep** img, **HxPoint** begin, **HxPoint** end)

  Restriction of domain.

5.160.1 Detailed Description

5.160.2 Function Documentation

5.160.2.1 HxImageRep L_HXIMAGEREP HxRestrict (HxImageRep *img*, HxPoint *begin*, HxPoint *end*)

Restriction of domain.

Restrict the domain of the image to the region specified by the given points. Points are treated as pixel coordinates (integers).

```
13 {
14     return img.restrict(begin, end);
15 }
```

5.161 HxRGB2Intensity.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRGB2Intensity** (**HxImageRep** im)
  Conversion of RGB to intensity.

5.161.1 Detailed Description

5.161.2 Function Documentation

5.161.2.1 HxImageRep L_HXIMAGEREP HxRGB2Intensity (HxImageRep *im*)

Conversion of RGB to intensity.

The function converts an RGB image to an intensity image

```
77 {
78     return im.unaryPixOp("RGB2Intensity");
79 }
```

5.162 HxRightShift.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRightShift** (**HxImageRep** im1, **HxImageRep** im2)
  Right shift.

5.162.1 Detailed Description

5.162.2 Function Documentation

5.162.2.1 HxImageRep L_HXIMAGEREP HxRightShift (HxImageRep *im1*, HxImageRep *im2*)

Right shift.

The function performs addition (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoRightShift** (p. 198). The image functor instantiator : **HxInstantiatorRightShift** (p. 538).

```
13 {
14     return im1.binaryPixOp(im2, "rightShift");
15 }
```

5.163 HxRightShiftVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRightShiftVal** (**HxImageRep** im, **HxValue** val)

  Right shift.

5.163.1 Detailed Description

5.163.2 Function Documentation

5.163.2.1 HxImageRep L_HXIMAGEREP HxRightShiftVal (HxImageRep *im*, HxValue *val*)

Right shift.

The function performs addition (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoRightShift** (p. 198). The image functor instantiator : **HxInstantiatorRightShiftV** (p. 539).

```
13 {
14     return im.binaryPixOp(val, "rightShift");
15 }
```

5.164 HxRotate.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRotate** (**HxImageRep** img, double alpha, **HxGeoIntType** gi=LINEAR, int adjustSize=1, **HxValue** background=**HxValue**(0))

  Rotation (around Z-axis in middle of image).

5.164.1 Detailed Description

5.164.2 Function Documentation

5.164.2.1 HxImageRep L_HXIMAGEREP HxRotate (HxImageRep *img*, double *alpha*, HxGeoIntType *gi* = LINEAR, int *adjustSize* = 1, HxValue *background* = HxValue(0))

Rotation (around Z-axis in middle of image).

This is the normal rotation in 2D. Alpha in degrees.

```
16 {
17     if (img.dimensionality() == 2) {
18         HxMatrix m =
19             HxMatrix::translate2d(
20                 img.dimensionSize(1)/2, img.dimensionSize(2)/2) *
21             HxMatrix::rotate2dDeg(-alpha) * // Y-axis points down
22             HxMatrix::translate2d(
23                 -img.dimensionSize(1)/2, -img.dimensionSize(2)/2);
24         return img.geometricOp2d(m, gi, FORWARD, adjustSize, background);
25     } else {
26         HxEnvironment::instance()->errorStream()
27             << "3d rotation not implemented yet" << STD_ENDL;
28         HxEnvironment::instance()->flush();
29         return HxImageRep();
30     }
31 }
```

5.165 HxRound.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRound** (**HxImageRep** im)

  Rounding.

5.165.1 Detailed Description

5.165.2 Function Documentation

5.165.2.1 HxImageRep L_HXIMAGEREP HxRound (HxImageRep *im*)

Rounding.

The function computes the rounding (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoRound** (p. 747). The image functor instantiator : **HxInstantiatorRound** (p. 540).

```
13 {
14     return im.unaryPixOp("round");
15 }
```

5.166 HxScale.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxScale** (**HxImageRep** img, double sx, double sy, double sz, **HxGeoIntType** gi, int adjustSize)

  Scaling.

5.166.1 Detailed Description

5.166.2 Function Documentation

5.166.2.1 HxImageRep L_HXIMAGEREP HxScale (HxImageRep *img*, double *sx*, double *sy*, double *sz*, HxGeoIntType *gi*, int *adjustSize*)

Scaling.

```
16 {
17     if (img.dimensionality() == 2) {
18         HxMatrix m = HxMatrix::scale2d(sx, sy);
19         return img.geometricOp2d(m, gi, FORWARD, adjustSize);
20     } else {
21         HxEnvironment::instance()->errorStream()
22             << "3d scaling not implemented yet" << STD_ENDL;
23         HxEnvironment::instance()->flush();
24         return HxImageRep();
25     }
26 }
```

5.167 HxSin.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSin** (**HxImageRep** im)

  Sine.

5.167.1 Detailed Description

5.167.2 Function Documentation

5.167.2.1 HxImageRep L_HXIMAGEREP HxSin (HxImageRep *im*)

Sine.

The function computes the sine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoSin** (p. 748). The image functor instantiator : **HxInstantiatorSin** (p. 540).

```
13 {
14     return im.unaryPixOp("sin");
15 }
```

5.168 HxSinh.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSinh** (**HxImageRep** im)

  Hyperbolic sine.

5.168.1 Detailed Description

5.168.2 Function Documentation

5.168.2.1 HxImageRep L_HXIMAGEREP HxSinh (HxImageRep *im*)

Hyperbolic sine.

The function computes the hyperbolic sine (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoSinh** (p. 750). The image functor instantiator : **HxInstantiatorSinh** (p. 541).

```
13 {
14     return im.unaryPixOp("sinh");
15 }
```

5.169 HxSizes.h File Reference

```
#include "HxVec3Int.h"
#include "HxString.h"
```

Typedefs

- typedef **HxVec3Int HxSizes**

  Definition of sizes.

Functions

- **HxString makeString** (const **HxSizes** &s)
- const char * **ClassName** (const **HxSizes** &)

5.169.1 Detailed Description

5.169.2 Typedef Documentation

5.169.2.1 typedef HxVec3Int HxSizes

Definition of sizes.

HxSizes is used primarily to specify the dimension sizes of an image or a region in an image.

5.170 HxSqrt.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSqrt** (**HxImageRep** im)

  Square root.

5.170.1 Detailed Description

5.170.2 Function Documentation

5.170.2.1 HxImageRep L_HXIMAGEREP HxSqrt (HxImageRep *im*)

Square root.

The function computes the square root (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoSqrt** (p. 751). The image functor instantiator : **HxInstantiatorSqrt** (p. 542).

```
13 {
14     return im.unaryPixOp("sqrt");
15 }
```

5.171 HxSquaredDistance.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSquaredDistance** (**HxImageRep** im1, **HxImageRep** im2)

  Squared distance.

5.171.1 Detailed Description

5.171.2 Function Documentation

5.171.2.1 HxImageRep L_HXIMAGEREP HxSquaredDistance (HxImageRep *im1*, HxImageRep *im2*)

Squared distance.

The function computes the squared distance of all corresponding pixels in the input images via a binary pixel operation.

Implementation specifics : The pixel functor : **HxBpoSqrDst** (p. 199). The image functor instantiator : **HxInstantiatorSqrDst** (p. 541).

```
100 {
101     // call HxImageRep member function to do the image processing
102     return im1.binaryPixOp(im2, "sqrDst");
103 }
```

5.172 HxStringList.h File Reference

```
#include "HxString.h"
#include <list>
#include <vector>
```

Compounds

- class **HxStringList**

  Class definition for list of HxString's.

Typedefs

- typedef HxStringList::iterator **HxStringListIter**

  Iterator for **HxStringList** (p. 715).

- typedef HxStringList::const_iterator **HxStringListConstIter**

  Const iterator for **HxStringList** (p. 715).

- typedef **HxStringList::back_insert_iterator HxStringListBackInserter**

  Back inserter for **HxStringList** (p. 715).

Functions

- **HxString makeString** (**HxStringListConstIter** begin, **HxStringListConstIter** end, **HxString** separator=**HxString**(""))
- int **splitString** (**HxString** src, char separator, **HxStringListBackInserter**)

5.172.1 Detailed Description

5.172.2 Typedef Documentation

5.172.2.1 typedef HxStringList::iterator HxStringListIter

Iterator for **HxStringList** (p. 715).

5.172.2.2 typedef HxStringList::const_iterator HxStringListConstIter

Const iterator for **HxStringList** (p. 715).

5.172.2.3 typedef HxStringList::back_insert_iterator HxStringListBackInserter

Back inserter for **HxStringList** (p. 715).

5.173 HxStringNative.h File Reference

```
#include <string>
```

Typedefs

- typedef std::string **HxString**

  HxString definition.

Functions

- const char * **ClassName** (**HxString**)
- int **atoi** (const **HxString** &s)
- long **atol** (const **HxString** &s)
- double **atof** (const **HxString** &s)
- L_HXBASIS **HxString makeString** (int)
- L_HXBASIS **HxString makeString** (double)
- **HxString makeString** (const **HxString** &s)

5.173.1 Detailed Description

5.173.2 Typedef Documentation

5.173.2.1 typedef std::string HxString

HxString definition.

5.174 HxSub.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSub** (**HxImageRep** im1, **HxImageRep** im2)
  Subtraction.

5.174.1 Detailed Description

5.174.2 Function Documentation

5.174.2.1 HxImageRep L_HXIMAGEREP HxSub (HxImageRep *im1*, HxImageRep *im2*)

Subtraction.

The function performs subtraction (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoSub** (p. 200). The image functor instantiator : **HxInstantiatorSub** (p. 542).

```
13 {
14     return im1.binaryPixOp(im2, "sub");
15 }
```

5.175 HxSubVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSubVal** (**HxImageRep** im, **HxValue** val)
  Subtraction.

5.175.1 Detailed Description

5.175.2 Function Documentation

5.175.2.1 HxImageRep L_HXIMAGEREP HxSubVal (HxImageRep *im*, HxValue *val*)

Subtraction.

The function performs subtraction (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoSub** (p. 200). The image functor instantiator : **HxInstantiatorSubV** (p. 543).

```
13 {
14     return im.binaryPixOp(val, "sub");
15 }
```

5.176 HxSup.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSup** (**HxImageRep** im1, **HxImageRep** im2)
  Supremum.

5.176.1 Detailed Description

5.176.2 Function Documentation

5.176.2.1 HxImageRep L_HXIMAGEREP HxSup (HxImageRep *im1*, HxImageRep *im2*)

Supremum.

The function performs supremum (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoSup** (p. 203). The image functor instantiator : **HxInstantiatorSup** (p. 544).

```
14 {
15     return im1.binaryPixOp(im2, "sup");
16 }
```

5.177 HxSupVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSupVal** (**HxImageRep** im, **HxValue** val)
  Supremum.

5.177.1 Detailed Description

5.177.2 Function Documentation

5.177.2.1 HxImageRep L_HXIMAGEREP HxSupVal (HxImageRep *im*, HxValue *val*)

Supremum.

The function performs supremum (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoSup** (p. 203). The image functor instantiator : **HxInstantiatorSupV** (p. 546).

```
13 {
14     return im.binaryPixOp(val, "sup");
15 }
```

5.178 HxTagList.h File Reference

```
#include "HxStd.h"
#include "HxIoFwd.h"
#include "HxString.h"
#include <list>
#include "HxTagTem.h"
```

Compounds

- class **HxTagList**

  A list of tags.

Functions

- std::ostream & **operator**<< (std::ostream &os, const **HxTagList** &tags)
- template<class ValT> void **HxAddTag** (**HxTagList** &tags, **HxString** name, ValT v)

  Add a tag to the list of tags.

- template<class ValT> ValT **HxGetTag** (const **HxTagList** &tags, **HxString** name)

  Get a tag from the list of tags.

- template<class ValT> ValT **HxGetTag** (const **HxTagList** &tags, **HxString** name, ValT defV)

  Get a tag from the list of tags.

- bool **HxTagIsSet** (const **HxTagList** &tags, **HxString** name)

  Check if tag is set.

- **HxTagList** & **HxMakeTagList** ()

  Function to return a reference to a garbage dump taglist.

5.178.1 Detailed Description

5.178.2 Function Documentation

5.178.2.1 template<class ValT> void HxAddTag (HxTagList & *tags*, HxString *name*, ValT *v*) [inline]

Add a tag to the list of tags.

```
79 {
80     typedef HxTagTem<ValT>  Tag;
81     tags.addTag(new Tag(name, v));
82 }
```

5.178.2.2 template<class ValT> ValT HxGetTag (const HxTagList & *tags*, HxString *name*) [inline]

Get a tag from the list of tags.

```
89 {
90     typedef HxTagTem<ValT>* TagPtr;
91     TagPtr t = TagPtr(tags.getTag(name));
92     ValT v;
93     if (t)
94         v = t->getValue();
95     return v;
96 }
```

5.178.2.3 template<class ValT> ValT HxGetTag (const HxTagList & *tags*, HxString *name*, ValT *defV*) [inline]

Get a tag from the list of tags.

If the tag is not set return the specified default value.

```
104 {
105     typedef HxTagTem<ValT>* TagPtr;
106     TagPtr t = TagPtr(tags.getTag(name));
107     return t ? t->getValue() : defV;
108 }
```

5.178.2.4 bool HxTagIsSet (const HxTagList & *tags*, HxString *name*) [inline]

Check if tag is set.

```
114 {
115     return tags.getTag(name) ? true : false;
116 }
```

5.178.2.5 HxTagList& HxMakeTagList ()

Function to return a reference to a garbage dump taglist.

ADB 14 Feb 2001

```
126 {
127     static HxTagList tl;
128     tl.erase();
129     return tl;
130 }
```

5.179 HxTan.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxTan** (**HxImageRep** im)

  Tangent.

5.179.1 Detailed Description

5.179.2 Function Documentation

5.179.2.1 HxImageRep L_HXIMAGEREP HxTan (HxImageRep *im*)

Tangent.

The function computes the tangent (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoTan** (p. 753). The image functor instantiator : **HxInstantiatorTan** (p. 546).

```
13 {
14     return im.unaryPixOp("tan");
15 }
```

5.180 HxTanh.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxTanh** (**HxImageRep** im)

  Hyperbolic tangent.

5.180.1 Detailed Description

5.180.2 Function Documentation

5.180.2.1 HxImageRep L_HXIMAGEREP HxTanh (HxImageRep *im*)

Hyperbolic tangent.

The function computes the hyperbolic tangent (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoTanh** (p. 754). The image functor instantiator : **HxInstantiatorTanh** (p. 547).

```
13 {
14     return im.unaryPixOp("tanh");
15 }
```

5.181 HxThreshold.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxThreshold** (**HxImageRep** im, **HxValue** level)
  Thresholding.

5.181.1 Detailed Description

5.181.2 Function Documentation

5.181.2.1 HxImageRep L_HXIMAGEREP HxThreshold (HxImageRep *im*, HxValue *level*)

Thresholding.

```
13 {
14     HxTagList tags;
15     HxAddTag(tags, "level", level);
16     return im.unaryPixOp("threshold", tags);
17 }
```

5.182 HxTriStateThreshold.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxTriStateThreshold** (**HxImageRep** im, **HxValue** level, **HxValue** v1, **HxValue** v2, **HxValue** v3)

Tri state threshold.

5.182.1 Detailed Description

5.182.2 Function Documentation

5.182.2.1 HxImageRep L_HXIMAGEREP HxTriStateThreshold (HxImageRep *im*, HxValue *level*, HxValue *v1*, HxValue *v2*, HxValue *v3*)

Tri state threshold.

The function computes the tri state threshold of all pixels in the input image via a unary pixel operation.

Implementation specifics : The pixel functor : **HxUpoTriStateThreshold** (p. 755). The image functor instantiator : **HxInstantiatorTriStateThreshold** (p. 547).

```
128 {
129     // Put all non-image parameters in a TagList
130     HxTagList tags;
131     HxAddTag(tags, "level", level);
132     HxAddTag(tags, "v1", v1);
133     HxAddTag(tags, "v2", v2);
134     HxAddTag(tags, "v3", v3);
135
136     // call HxImageRep member function to do the image processing
137     return im.unaryPixOp("triStateThreshold", tags);
138 }
```

5.183 HxUnaryMax.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryMax** (**HxImageRep** im)

  Unary maximum.

5.183.1 Detailed Description

5.183.2 Function Documentation

5.183.2.1 HxImageRep L_HXIMAGEREP HxUnaryMax (HxImageRep *im*)

Unary maximum.

The function computes the unary maximum (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoMax** (p. 738). The image functor instantiator : **HxInstantiatorUpoMax** (p. 548).

```
13 {
14     return im.unaryPixOp("max");
15 }
```

5.184 HxUnaryMin.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryMin** (**HxImageRep** im)
  Unary minimum.

5.184.1 Detailed Description

5.184.2 Function Documentation

5.184.2.1 HxImageRep L_HXIMAGEREP HxUnaryMin (HxImageRep *im*)

Unary minimum.

The function computes the unary minimum (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoMin** (p. 739). The image functor instantiator : **HxInstantiatorUpoMin** (p. 549).

```
13 {
14     return im.unaryPixOp("min");
15 }
```

5.185 HxUnaryProduct.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryProduct** (**HxImageRep** im)
  Unary product.

5.185.1 Detailed Description

5.185.2 Function Documentation

5.185.2.1 HxImageRep L_HXIMAGEREP HxUnaryProduct (HxImageRep *im*)

Unary product.

The function computes the unary product (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoProduct** (p. 746). The image functor instantiator : **HxInstantiatorProduct** (p. 537).

```
13 {
14     return im.unaryPixOp("product");
15 }
```

5.186 HxUnarySum.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnarySum** (**HxImageRep** im)

  Unary sum.

5.186.1 Detailed Description

5.186.2 Function Documentation

5.186.2.1 HxImageRep L_HXIMAGEREP HxUnarySum (HxImageRep *im*)

Unary sum.

The function computes the unary sum (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxUpoSum** (p. 752). The image functor instantiator : **HxInstantiatorSum** (p. 544).

```
13 {
14     return im.unaryPixOp("sum");
15 }
```

5.187 HxUniform.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUniform** (**HxImageRep** im, **HxSizes** sizes)

  Uniform filter.

5.187.1 Detailed Description

5.187.2 Function Documentation

5.187.2.1 HxImageRep L_HXIMAGEREP HxUniform (HxImageRep *im*, HxSizes *sizes*)

Uniform filter.

```
15 {
16     // An image signature for a 2D image with 64-bit real valued scalar pixels
17     HxImageSignature sig(2, 1, REAL_VALUE, 64);
18 
19     // Construct the separable kernels
20     HxImageRep kx = HxImageFactory::instance().fromValue(sig,
21         HxSizes(sizes.x(),1,1), 1./sizes.x());
22     HxImageRep ky = HxImageFactory::instance().fromValue(sig,
23         HxSizes(sizes.y(),1,1), 1./sizes.y());
24     HxImageRep kz = HxImageFactory::instance().fromValue(sig,
25         HxSizes(sizes.z(),1,1), 1./sizes.z());
26 
27     // and apply the operation
28     if (im.dimensionality() == 3) {
29         HxImageRep res = im.generalizedConvolutionK1d(1, kx,
30             "mul", "addAssign", HxImageRep::ARITH_PREC);
31         res = res.generalizedConvolutionK1d(2, ky,
32             "mul", "addAssign", HxImageRep::ARITH_PREC);
33         return res.generalizedConvolutionK1d(3, kz,
34             "mul", "addAssign", HxImageRep::ARITH_PREC);
35     }
36 
37     return im.genConvSeparated(
38         1, kx, ky, "mul", "addAssign", HxImageRep::ARITH_PREC);
39 }
```

5.188 HxUniformNonSep.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxUniformNonSep** (**HxImageRep** im, **HxSizes** sizes)

  Non separated version of the uniform filter for demo purposes.

5.188.1 Detailed Description

5.188.2 Function Documentation

5.188.2.1 HxImageRep L_HXIMAGEREP HxUniformNonSep (HxImageRep *im*, HxSizes *sizes*)

Non separated version of the uniform filter for demo purposes.

```
14 {
15     // An image signature for a 2D image with 64-bit real valued scalar pixels
```

```
16     HxImageSignature sig(2, 1, REAL_VALUE, 64);
17
18     // The pixel value of the uniform kernel
19     double val = 1.0 / (sizes.x() * sizes.y() * sizes.z());
20
21     // Now construct the kernel image
22     HxImageRep kernel = HxMakeFromValue(sig, sizes, val);
23
24     // and apply the operation
25     return im.generalizedConvolution(kernel, "mul", "addAssign");
26 }
```

5.189 HxValueList.h File Reference

```
#include "HxValue.h"
#include <list>
```

Compounds

- class **HxValueList**

  Class definition for list of **HxValue** (p. 757)*'s.*

Typedefs

- typedef HxValueList::iterator **HxValueListIter**

  Iterator for **HxValueList** (p. 765).

- typedef HxValueList::const_iterator **HxValueListConstIter**

  Const iterator for **HxValueList** (p. 765).

- typedef **HxValueList::back_insert_iterator HxValueListBackInserter**

  Back inserter for **HxValueList** (p. 765).

5.189.1 Detailed Description

5.189.2 Typedef Documentation

5.189.2.1 typedef HxValueList::iterator HxValueListIter

Iterator for **HxValueList** (p. 765).

5.189.2.2 typedef HxValueList::const_iterator HxValueListConstIter

Const iterator for **HxValueList** (p. 765).

5.189.2.3 typedef HxValueList::back_insert_iterator HxValueListBackInserter

Back inserter for **HxValueList** (p. 765).

5.190 HxValueType.h File Reference

```
#include "HxIoFwd.h"
#include "HxString.h"
```

Enumerations

- enum **HxValueType** { **INT_VALUE**, **REAL_VALUE**, **COMPLEX_VALUE** }

  The type of a **HxValue** (p. 757).

Functions

- std::ostream & **HxValueType_put** (std::ostream &os, **HxValueType** val)
- std::ostream & **operator<<** (std::ostream &os, **HxValueType** val)
- **HxString makeString** (**HxValueType** val)

5.190.1 Detailed Description

5.190.2 Enumeration Type Documentation

5.190.2.1 enum HxValueType

The type of a **HxValue** (p. 757).

Enumeration to make a distinction between integer values (INT_VALUE) and real/floating point values (REAL_VALUE), and complex values (COMPLEX_VALUE). operator<< has been overloaded to print HxValueTypes on a stream.

```
26 { INT_VALUE, REAL_VALUE, COMPLEX_VALUE };
```

5.191 HxWriteFile.h File Reference

```
#include "HxImageRep.h"
```

Functions

- bool L_HXIMAGEREP **HxWriteFile** (**HxImageRep** im, **HxString** fileName)

  Write image to file using HxImgFileIo.

5.191.1 Detailed Description

5.191.2 Function Documentation

5.191.2.1 bool L_HXIMAGEREP HxWriteFile (HxImageRep *im*, HxString *fileName*)

Write image to file using HxImgFileIo.

```
14 {
15     HxTagList tags;
16     return HxImageFactory::instance().writeFile(im, fileName, tags);
17 }
```

5.192 HxXor.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxXor** (**HxImageRep** im1, **HxImageRep** im2)
  Exclusive or.

5.192.1 Detailed Description

5.192.2 Function Documentation

5.192.2.1 HxImageRep L_HXIMAGEREP HxXor (HxImageRep *im1*, HxImageRep *im2*)

Exclusive or.

The function performs exclusive or (see **Pixels** (p. 5)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoXor** (p. 205). The image functor instantiator : **HxInstantiatorXor** (p. 549).

```
13 {
14     return im1.binaryPixOp(im2, "xor");
15 }
```

5.193 HxXorVal.h File Reference

```
#include "HxImageRep.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxXorVal** (**HxImageRep** im, **HxValue** val)
  Exclusive or.

5.193.1 Detailed Description

5.193.2 Function Documentation

5.193.2.1 HxImageRep L_HXIMAGEREP HxXorVal (HxImageRep *im*, HxValue *val*)

Exclusive or.

The function performs exclusive or (see **Pixels** (p. 5)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 9)).

Implementation specifics : The pixel functor : **HxBpoXor** (p. 205). The image functor instantiator : **HxInstantiatorXorV** (p. 550).

```
13 {
14     return im.binaryPixOp(val, "xor");
15 }
```

Chapter 6

Class Reference

6.1 HxArrowR2 Class Reference

Class definition for arrows in R2.

`#include <HxArrowR2.h>`

Public Methods

- **HxArrowR2** ()

  Constructor.

- **HxArrowR2** (const **HxVectorR2** &v)

  Construct from vector (origin = (0,0)).

- **HxArrowR2** (const **HxPointR2** &p, const **HxVectorR2** &v)

  Construct from given point and vector.

- **HxArrowR2** (double ox, double oy, double dx, double dy)

  Construct from given point and vector.

- ~**HxArrowR2** ()

  Destructor.

- **HxPointR2 origin** ()

  Get the origin.

- **HxVectorR2 displace** ()

  Get the displacement.

- STD_OSTREAM & **put** (STD_OSTREAM &) const

  Put the arrow on the given stream.

6.1.1 Detailed Description

Class definition for arrows in R2.

An arrow has a position (origin) and a direction (displace). Both have real-value coordinates (R2).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 HxArrowR2::HxArrowR2 () [inline]

Constructor.

```
60 {
61 }
```

6.1.2.2 HxArrowR2::HxArrowR2 (const HxVectorR2 & *v*) [inline]

Construct from vector (origin = (0,0)).

```
64                                          : _origin(0, 0), _displace(v)
65 {
66 }
```

6.1.2.3 HxArrowR2::HxArrowR2 (const HxPointR2 & *p*, const HxVectorR2 & *v*) [inline]

Construct from given point and vector.

```
69                                                                  : _origin(p),
70                                                                    _displace(v)
71 {
72 }
```

6.1.2.4 HxArrowR2::HxArrowR2 (double *ox*, double *oy*, double *dx*, double *dy*) [inline]

Construct from given point and vector.

```
75                                                                  :
76                                              _origin(ox, oy), _displace(dx, dy)
77 {
78 }
```

6.1.2.5 HxArrowR2::~HxArrowR2 () [inline]

Destructor.

```
82 {
83 }
```

6.1.3 Member Function Documentation

6.1.3.1 HxPointR2 HxArrowR2::origin () [inline]

Get the origin.

```
87 {
88     return _origin;
89 }
```

6.1.3.2 HxVectorR2 HxArrowR2::displace () [inline]

Get the displacement.

```
93 {
94     return _displace;
95 }
```

6.1.3.3 STD_OSTREAM & HxArrowR2::put (STD_OSTREAM & *os*) const [inline]

Put the arrow on the given stream.

```
99 {
100     return os << _origin <<  " " << _displace;
101 }
```

The documentation for this class was generated from the following file:

- **HxArrowR2.h**

6.2 HxBpoAdd Class Template Reference

Pixel functor for computation of addition.

`#include <HxBpoAdd.h>`

Public Methods

- **HxBpoAdd** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x + y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : "add".

6.2.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoAdd< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of addition.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoAdd< DstValT, Src1ValT, Src2ValT >::HxBpoAdd (HxTagList &) [inline]

Constructor : empty.

```
26                          {}
```

6.2.3 Member Function Documentation

6.2.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoAdd< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x + y #.

```
30                          { return x + y; }
```

6.2.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoAdd< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”add”.

```
37                          { return HxString("add"); }
```

The documentation for this class was generated from the following file:

- **HxBpoAdd.h**

6.3 HxBpoAddAssign Struct Template Reference

Pixel functor for computation of addition assignment.

`#include <HxBpoAddAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoAddAssign** (**HxTagList** &)
  Constructor : empty.
- void **doIt** (DstValT &x, const SrcValT &y)
  Actual operation : # x += y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : "addAssign".

6.3.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoAddAssign< DstValT, SrcValT >

Pixel functor for computation of addition assignment.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 template<class DstValT, class SrcValT> HxBpoAddAssign< DstValT, SrcValT >::HxBpoAddAssign (HxTagList &) [inline]

Constructor : empty.

```
28                          {}
```

6.3.3 Member Function Documentation

6.3.3.1 template<class DstValT, class SrcValT> void HxBpoAddAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x += y #.

```
32                          { x += y; }
```

6.3.3.2 template<class DstValT, class SrcValT> HxString HxBpoAddAssign< DstValT, SrcValT >::className () [inline, static]

The name : "addAssign".

```
39                          { return HxString("addAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoAddAssign.h**

6.4 HxBpoAnd Class Template Reference

Pixel functor for computation of and.

`#include <HxBpoAnd.h>`

Public Methods

- **HxBpoAnd** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)

  Actual operation : # return x.and(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : "and".

6.4.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoAnd< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of and.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoAnd< DstValT, Src1ValT, Src2ValT >::HxBpoAnd (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.4.3 Member Function Documentation

6.4.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoAnd< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.and(y) #.

```
29                          { return x.and(y); }
```

6.4.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoAnd< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”and”.

```
36                      { return HxString("and"); }
```

The documentation for this class was generated from the following file:

- **HxBpoAnd.h**

6.5 HxBpoCross Class Template Reference

Pixel functor for computation of cross product.

`#include <HxBpoCross.h>`

Public Methods

- **HxBpoCross** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)

  Actual operation : # return x.cross(y) #.

Static Public Methods

- **HxString className** ()

  The name : ”cross”.

6.5.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoCross< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of cross product.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoCross< DstValT, Src1ValT, Src2ValT >::HxBpoCross (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.5.3 Member Function Documentation

6.5.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoCross< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.cross(y) #.

```
29                          { return x.cross(y); }
```

6.5.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoCross< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”cross”.

```
33                          { return HxString("cross"); }
```

The documentation for this class was generated from the following file:

- **HxBpoCross.h**

6.6 HxBpoDiv Class Template Reference

Pixel functor for computation of division.

`#include <HxBpoDiv.h>`

Public Methods

- **HxBpoDiv** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x / y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”div”.

6.6.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoDiv< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of division.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoDiv< DstValT, Src1ValT, Src2ValT >::HxBpoDiv (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.6.3 Member Function Documentation

6.6.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoDiv< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x / y #.

```
29                          { return x / y; }
```

6.6.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoDiv< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”div”.

```
36                          { return HxString("div"); }
```

The documentation for this class was generated from the following file:

- **HxBpoDiv.h**

6.7 HxBpoDot Class Template Reference

Pixel functor for computation of dot product.

`#include <HxBpoDot.h>`

Public Methods

- **HxBpoDot** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.dot(y) #.

Static Public Methods

- **HxString className** ()
  The name : ”dot”.

6.7.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoDot< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of dot product.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoDot< DstValT, Src1ValT, Src2ValT >::HxBpoDot (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.7.3 Member Function Documentation

6.7.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoDot< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.dot(y) #.

```
29                          { return x.dot(y); }
```

6.7.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoDot< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”dot”.

```
33                          { return HxString("dot"); }
```

The documentation for this class was generated from the following file:

- **HxBpoDot.h**

6.8 HxBpoEqual Class Template Reference

Pixel functor for computation of equal.

`#include <HxBpoEqual.h>`

Public Methods

- **HxBpoEqual** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x == y #.

Static Public Methods

- **HxString className** ()

  The name : ”equal”.

6.8.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of equal.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoEqual< DstValT, Src1ValT, Src2ValT >::HxBpoEqual (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.8.3 Member Function Documentation

6.8.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoEqual< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x == y #.

```
29                          { return x == y; }
```

6.8.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”equal”.

```
33                          { return HxString("equal"); }
```

The documentation for this class was generated from the following file:

- **HxBpoEqual.h**

6.9 HxBpoGreaterEqual Class Template Reference

Pixel functor for computation of greater equal.

`#include <HxBpoGreaterEqual.h>`

Public Methods

- **HxBpoGreaterEqual** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x >= y #.

Static Public Methods

- **HxString className** ()
  The name : "greaterEqual".

6.9.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of greater equal.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >::HxBpoGreaterEqual (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.9.3 Member Function Documentation

6.9.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x >= y #.

```
29                          { return x >= y; }
```

6.9.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "greaterEqual".

```
33                          { return HxString("greaterEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoGreaterEqual.h**

6.10 HxBpoGreaterThan Class Template Reference

Pixel functor for computation of greater than.

`#include <HxBpoGreaterThan.h>`

Public Methods

- **HxBpoGreaterThan** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)

  Actual operation : # return x > y #.

Static Public Methods

- **HxString className** ()

  The name : "greaterThan".

6.10.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of greater than.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >::HxBpoGreaterThan (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.10.3 Member Function Documentation

6.10.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x > y #.

```
29                      { return x > y; }
```

6.10.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”greaterThan”.

```
33                          { return HxString("greaterThan"); }
```

The documentation for this class was generated from the following file:

- **HxBpoGreaterThan.h**

6.11 HxBpoInf Class Template Reference

Pixel functor for computation of infimum.

`#include <HxBpoInf.h>`

Public Methods

- **HxBpoInf** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.inf(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”inf”.

6.11.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoInf< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of infimum.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoInf< DstValT, Src1ValT, Src2ValT >::HxBpoInf (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.11.3 Member Function Documentation

6.11.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoInf< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.inf(y) #.

```
29                          { return x.inf(y); }
```

6.11.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoInf< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "inf".

```
36                          { return HxString("inf"); }
```

The documentation for this class was generated from the following file:

- **HxBpoInf.h**

6.12 HxBpoInfAssign Struct Template Reference

Pixel functor for computation of infimum assignment.

`#include <HxBpoInfAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoInfAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x.infAssign(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : "infAssign".

6.12.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoInfAssign< DstValT, SrcValT >

Pixel functor for computation of infimum assignment.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 template<class DstValT, class SrcValT> HxBpoInfAssign< DstValT, SrcValT >::HxBpoInfAssign (HxTagList &) [inline]

Constructor : empty.

```
26                          {}
```

6.12.3 Member Function Documentation

6.12.3.1 template<class DstValT, class SrcValT> void HxBpoInfAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.infAssign(y) #.

```
30                          { x.infAssign(y); }
```

6.12.3.2 template<class DstValT, class SrcValT> HxString HxBpoInfAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”infAssign”.

```
37                          { return HxString("infAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoInfAssign.h**

6.13 HxBpoLeftShift Class Template Reference

Pixel functor for computation of left shift.

`#include <HxBpoLeftShift.h>`

Public Methods

- **HxBpoLeftShift** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.leftShift(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : ”leftShift”.

6.13.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of left shift.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >::HxBpoLeftShift (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.13.3 Member Function Documentation

6.13.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.leftShift(y) #.

```
29                          { return x.leftShift(y); }
```

6.13.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”leftShift”.

```
36                          { return HxString("leftShift"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLeftShift.h**

6.14 HxBpoLessEqual Class Template Reference

Pixel functor for computation of less equal.

`#include <HxBpoLessEqual.h>`

Public Methods

- **HxBpoLessEqual** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)

  Actual operation : # return x <= y #.

Static Public Methods

- **HxString className** ()

  The name : "lessEqual".

6.14.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of less equal.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >::HxBpoLessEqual (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.14.3 Member Function Documentation

6.14.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x <= y #.

```
29                          { return x <= y; }
```

6.14.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "lessEqual".

```
33                          { return HxString("lessEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLessEqual.h**

6.15 HxBpoLessThan Class Template Reference

Pixel functor for computation of less than.

`#include <HxBpoLessThan.h>`

Public Methods

- **HxBpoLessThan** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x < y #.

Static Public Methods

- **HxString className** ()
  The name : "lessThan".

6.15.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLessThan< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of less than.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLessThan< DstValT, Src1ValT, Src2ValT >::HxBpoLessThan (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.15.3 Member Function Documentation

6.15.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoLessThan< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x < y #.

```
29                          { return x < y; }
```

6.15.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLessThan< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”lessThan”.

```
33                          { return HxString("lessThan"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLessThan.h**

6.16 HxBpoMax Class Template Reference

Pixel functor for computation of maximum.

`#include <HxBpoMax.h>`

Public Methods

- **HxBpoMax** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.max(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”max”.

6.16.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMax< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of maximum.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMax< DstValT, Src1ValT, Src2ValT >::HxBpoMax (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.16.3 Member Function Documentation

6.16.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoMax< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.max(y) #.

```
29                          { return x.max(y); }
```

6.16.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMax< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”max”.

```
36                          { return HxString("max"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMax.h**

6.17 HxBpoMaxAssign Struct Template Reference

Pixel functor for computation of maximum assignment.

`#include <HxBpoMaxAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoMaxAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x.maxAssign(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : ”maxAssign”.

6.17.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMaxAssign< DstValT, SrcValT >

Pixel functor for computation of maximum assignment.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 template<class DstValT, class SrcValT> HxBpoMaxAssign< DstValT, SrcValT >::HxBpoMaxAssign (HxTagList &) [inline]

Constructor : empty.

```
27                          {}
```

6.17.3 Member Function Documentation

6.17.3.1 template<class DstValT, class SrcValT> void HxBpoMaxAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.maxAssign(y) #.

```
31                          { x.maxAssign(y); }
```

6.17.3.2 template<class DstValT, class SrcValT> HxString HxBpoMaxAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”maxAssign”.

```
38                          { return HxString("maxAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMaxAssign.h**

6.18 HxBpoMin Class Template Reference

Pixel functor for computation of minimum.

`#include <HxBpoMin.h>`

Public Methods

- **HxBpoMin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.min(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : ”min”.

6.18.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMin< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of minimum.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMin< DstValT, Src1ValT, Src2ValT >::HxBpoMin (HxTagList &) [inline]

Constructor : empty.

```
26                          {}
```

6.18.3 Member Function Documentation

6.18.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoMin< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.min(y) #.

```
30                          { return x.min(y); }
```

6.18.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMin< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”min”.

```
37                          { return HxString("min"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMin.h**

6.19 HxBpoMinAssign Struct Template Reference

Pixel functor for computation of minimum assignment.

`#include <HxBpoMinAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoMinAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x.minAssign(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : "minAssign".

6.19.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMinAssign< DstValT, SrcValT >

Pixel functor for computation of minimum assignment.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 template<class DstValT, class SrcValT> HxBpoMinAssign< DstValT, SrcValT >::HxBpoMinAssign (HxTagList &) [inline]

Constructor : empty.

```
27                      {}
```

6.19.3 Member Function Documentation

6.19.3.1 template<class DstValT, class SrcValT> void HxBpoMinAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.minAssign(y) #.

```
31                      { x.minAssign(y); }
```

6.19.3.2 template<class DstValT, class SrcValT> HxString HxBpoMinAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”minAssign”.

```
38                      { return HxString("minAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMinAssign.h**

6.20 HxBpoMod Class Template Reference

Pixel functor for computation of modulo.

`#include <HxBpoMod.h>`

Public Methods

- **HxBpoMod** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.mod(y) #.

Static Public Methods

- **HxString className** ()
  The name : ”mod”.

6.20.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMod< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of modulo.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMod< DstValT, Src1ValT, Src2ValT >::HxBpoMod (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.20.3 Member Function Documentation

6.20.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoMod< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.mod(y) #.

```
29                          { return x.mod(y); }
```

6.20.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMod< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”mod”.

```
33                          { return HxString("mod"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMod.h**

6.21 HxBpoMul Class Template Reference

Pixel functor for computation of multiplication.

`#include <HxBpoMul.h>`

Public Methods

- **HxBpoMul** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x ∗ y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”mul”.

6.21.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMul< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of multiplication.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMul< DstValT, Src1ValT, Src2ValT >::HxBpoMul (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.21.3 Member Function Documentation

6.21.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoMul< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x ∗ y #.

```
29                          { return x * y; }
```

6.21.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMul< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”mul”.

```
36                          { return HxString("mul"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMul.h**

6.22 HxBpoMulAssign Struct Template Reference

Pixel functor for computation of multiplication assignment.

`#include <HxBpoMulAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoMulAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x ∗= y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : ”mulAssign”.

6.22.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMulAssign< DstValT, SrcValT >

Pixel functor for computation of multiplication assignment.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 template<class DstValT, class SrcValT> HxBpoMulAssign< DstValT, SrcValT >::HxBpoMulAssign (HxTagList &) [inline]

Constructor : empty.

```
28                          {}
```

6.22.3 Member Function Documentation

6.22.3.1 template<class DstValT, class SrcValT> void HxBpoMulAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x *= y #.

```
32                          { x *= y; }
```

6.22.3.2 template<class DstValT, class SrcValT> HxString HxBpoMulAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”mulAssign”.

```
39                          { return HxString("mulAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMulAssign.h**

6.23 HxBpoNotEqual Class Template Reference

Pixel functor for computation of not equal.

`#include <HxBpoNotEqual.h>`

Public Methods

- **HxBpoNotEqual** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)

  Actual operation : # return x != y #.

Static Public Methods

- **HxString className** ()

  The name : "notEqual".

6.23.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of not equal.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >::HxBpoNotEqual (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.23.3 Member Function Documentation

6.23.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x != y #.

```
29                          { return x != y; }
```

6.23.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "notEqual".

```
33                          { return HxString("notEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoNotEqual.h**

6.24 HxBpoOr Class Template Reference

Pixel functor for computation of or.

`#include <HxBpoOr.h>`

Public Methods

- **HxBpoOr** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.or(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : "or".

6.24.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoOr< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of or.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoOr< DstValT, Src1ValT, Src2ValT >::HxBpoOr (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.24.3 Member Function Documentation

6.24.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoOr< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.or(y) #.

```
29                          { return x.or(y); }
```

6.24.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoOr< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”or”.

```
36                          { return HxString("or"); }
```

The documentation for this class was generated from the following file:

- **HxBpoOr.h**

6.25 HxBpoPow Class Template Reference

Pixel functor for computation of power.

`#include <HxBpoPow.h>`

Public Methods

- **HxBpoPow** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.pow(y) #.

Static Public Methods

- **HxString className** ()
  The name : ”pow”.

6.25.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoPow< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of power.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoPow< DstValT, Src1ValT, Src2ValT >::HxBpoPow (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.25.3 Member Function Documentation

6.25.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoPow< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.pow(y) #.

```
29                          { return x.pow(y); }
```

6.25.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoPow< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”pow”.

```
33                          { return HxString("pow"); }
```

The documentation for this class was generated from the following file:

- **HxBpoPow.h**

6.26 HxBpoRightShift Class Template Reference

Pixel functor for computation of right shift.

`#include <HxBpoRightShift.h>`

Public Methods

- **HxBpoRightShift** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.rightShift(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”rightShift”.

6.26.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoRightShift< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of right shift.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoRightShift< DstValT, Src1ValT, Src2ValT >::HxBpoRightShift (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.26.3 Member Function Documentation

6.26.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoRightShift< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.rightShift(y) #.

```
29                          { return x.rightShift(y); }
```

6.26.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoRightShift< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”rightShift”.

```
36                          { return HxString("rightShift"); }
```

The documentation for this class was generated from the following file:

- **HxBpoRightShift.h**

6.27 HxBpoSqrDst Class Template Reference

Pixel functor for computation of squared distance.

Public Methods

- **HxBpoSqrDst** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &arg1, const Src2ValT &arg2)
  Actual operation : # return (x - y)^2 #.

Static Public Methods

- **HxString className** ()
  The name : ”sqrDst”.

6.27.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of squared distance.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >::HxBpoSqrDst (HxTagList &) [inline]

Constructor : empty.

```
26                          { }
```

6.27.3 Member Function Documentation

6.27.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *arg1*, const Src2ValT & *arg2*) [inline]

Actual operation : # return (x - y)^2 #.

```
30                          { return (arg1 - arg2) * (arg1 - arg2); }
```

6.27.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”sqrDst”.

```
34                          { return HxString("sqrDst"); }
```

The documentation for this class was generated from the following file:

- HxSquaredDistance.c

6.28 HxBpoSub Class Template Reference

Pixel functor for computation of subtraction.

`#include <HxBpoSub.h>`

Public Methods

- **HxBpoSub** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x - y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : "sub".

6.28.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSub< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of subtraction.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSub< DstValT, Src1ValT, Src2ValT >::HxBpoSub (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.28.3 Member Function Documentation

6.28.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoSub< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x - y #.

```
29                          { return x - y; }
```

6.28.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSub< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "sub".

```
36                          { return HxString("sub"); }
```

The documentation for this class was generated from the following file:

- **HxBpoSub.h**

6.29 HxBpoSubAssign Struct Template Reference

Pixel functor for computation of subtraction assignment.

`#include <HxBpoSubAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoSubAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x -= y #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : "subAssign".

6.29.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoSubAssign< DstValT, SrcValT >

Pixel functor for computation of subtraction assignment.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 template<class DstValT, class SrcValT> HxBpoSubAssign< DstValT, SrcValT >::HxBpoSubAssign (HxTagList &) [inline]

Constructor : empty.

```
28                      {}
```

6.29.3 Member Function Documentation

6.29.3.1 template<class DstValT, class SrcValT> void HxBpoSubAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x -= y #.

```
32                      { x -= y; }
```

6.29.3.2 template<class DstValT, class SrcValT> HxString HxBpoSubAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”subAssign”.

```
39                          { return HxString("subAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoSubAssign.h**

6.30 HxBpoSup Class Template Reference

Pixel functor for computation of supremum.

`#include <HxBpoSup.h>`

Public Methods

- **HxBpoSup** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.sup(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : ”sup”.

6.30.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSup< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of supremum.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSup< DstValT, Src1ValT, Src2ValT >::HxBpoSup (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.30.3 Member Function Documentation

6.30.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoSup< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.sup(y) #.

```
29                          { return x.sup(y); }
```

6.30.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSup< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”sup”.

```
36                          { return HxString("sup"); }
```

The documentation for this class was generated from the following file:

- **HxBpoSup.h**

6.31 HxBpoSupAssign Struct Template Reference

Pixel functor for computation of supremum assignment.

`#include <HxBpoSupAssign.h>`

Public Types

- typedef DstValT **ArithType**

Public Methods

- **HxBpoSupAssign** (**HxTagList** &)

  Constructor : empty.

- void **doIt** (DstValT &x, const SrcValT &y)

  Actual operation : # x.supAssign(y) #.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()

  The name : ”supAssign”.

6.31.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoSupAssign< DstValT, SrcValT >

Pixel functor for computation of supremum assignment.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 template<class DstValT, class SrcValT> HxBpoSupAssign< DstValT, SrcValT >::HxBpoSupAssign (HxTagList &) [inline]

Constructor : empty.

```
27                          {}
```

6.31.3 Member Function Documentation

6.31.3.1 template<class DstValT, class SrcValT> void HxBpoSupAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.supAssign(y) #.

```
31                          { x.supAssign(y); }
```

6.31.3.2 template<class DstValT, class SrcValT> HxString HxBpoSupAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”supAssign”.

```
38                          { return HxString("supAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoSupAssign.h**

6.32 HxBpoXor Class Template Reference

Pixel functor for computation of exclusive or.

`#include <HxBpoXor.h>`

Public Methods

- **HxBpoXor** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &x, const Src2ValT &y)
  Actual operation : # return x.xor(y) #.

Static Public Methods

- **HxString className** ()

  The name : ”xor”.

6.32.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoXor< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of exclusive or.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoXor< DstValT, Src1ValT, Src2ValT >::HxBpoXor (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.32.3 Member Function Documentation

6.32.3.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoXor< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *x*, const Src2ValT & *y*) [inline]

Actual operation : # return x.xor(y) #.

```
29                          { return x.xor(y); }
```

6.32.3.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoXor< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”xor”.

```
33                          { return HxString("xor"); }
```

The documentation for this class was generated from the following file:

- **HxBpoXor.h**

6.33 HxBSplineBasis Class Reference

Class definition for basis for BSpline curves.

`#include <HxBSplineBasis.h>`

Public Methods

- **HxBSplineBasis** ()

  Construct default basis.

- **HxBSplineBasis** (HxBSplineType typeCurve, int degree, int nIntervals, HxBSplineKnotsAlg typeKnots=uniformKnots, double minT=0.0, double maxT=1.0)

  Construct basis with uniformily distributed knots, given number of intervals, and path interval: minT <= t < maxT.

- **HxBSplineBasis** (HxBSplineType typeCurve, int degree, const std::vector< double > &knots)

  Construct basis with given knots.

- ~**HxBSplineBasis** ()

  Destructor.

- HxBSplineType **curveType** () const

  Get the type of the cuve.

- int **degree** () const

  Get the degree.

- int **nIntervals** () const

  Get the number of intervals.

- HxBSplineKnotsAlg **knotsType** () const

  Get the type of the knots generating algorithm.

- double **minT** () const

  Get the minimum value for t.

- double **maxT** () const

  Get the maximum value for t.

- double **knot** (int j) const

  Get the value of the given knot.

- std::vector< double > **allKnots** () const

  Get values of all knots.

- int **numB** () const

  Get the number of basis functions.

- double **B** (int index, double t) const

  Get value of given basis at path position t.

- double **dB** (int order, int index, double t) const

  Get derivative of given basis at path position t.

- **HxBSplineInterval pathAffectedBy** (int index) const

Get path interval affected by given basis.

- HxBSplineBasis **insertKnot** (double t, int n=1) const

  Insert one knot at given position.

- STD_OSTREAM & **dump** (STD_OSTREAM &) const

  Dump the basis on the given stream.

- double **node** (int i) const
- int **maxBasis** (double t) const
- double **nearestKnot** (double t) const

Friends

- class **HxBSplineCurve**

6.33.1 Detailed Description

Class definition for basis for BSpline curves.

Based on ”Curve and Surface Fitting with Splines”, by ”Dierckx,P.”, ”Oxford”, ”1993”, chapter ”Univariate Splines”, and ”The NURBS book”, ”Piegl, L. and Tiller, W.”, Springer, 1997.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 HxBSplineBasis::HxBSplineBasis ()

Construct default basis.

```
17 {
18     makeDefault();
19 }
```

6.33.2.2 HxBSplineBasis::HxBSplineBasis (HxBSplineType *t*, int *d*, int *n*, HxBSplineKnotsAlg *alg* = uniformKnots, double *min* = 0.0, double *max* = 1.0)

Construct basis with uniformily distributed knots, given number of intervals, and path interval: minT <= t < maxT.

```
32 {
33     _curveType = t;
34     _degree = d;
35     _minT = min;
36     _maxT = max;
37     _nIntervals = n;
38 
39     if ( _degree <= 0 || max <= min || _nIntervals < 1 ||
40          ( t==closed && _nIntervals <= d ) ||
41          alg != uniformKnots ) {
42         message("(constructor) invalid parameters - using default");
43         makeDefault();
44         return;
```

```
45     }
46             // parameters ok
47     _knotsType = uniformKnots;
48     vector<double> knots = makeUniformKnots();
49     completeKnots(knots);
50 }
```

6.33.2.3 HxBSplineBasis::HxBSplineBasis (HxBSplineType *typeCurve*, int *degree*, const std::vector< double > & *knots*)

Construct basis with given knots.

The number of intervals and path are determined from the given knots.

6.33.2.4 HxBSplineBasis::~HxBSplineBasis ()

Destructor.

```
89 {
90 }
```

6.33.3 Member Function Documentation

6.33.3.1 HxBSplineType HxBSplineBasis::curveType () const [inline]

Get the type of the cuve.

```
156 {
157     return _curveType;
158 }
```

6.33.3.2 int HxBSplineBasis::degree () const [inline]

Get the degree.

```
162 {
163     return _degree;
164 }
```

6.33.3.3 int HxBSplineBasis::nIntervals () const [inline]

Get the number of intervals.

```
169 {
170     return _nIntervals;
171 }
```

6.33.3.4 HxBSplineKnotsAlg HxBSplineBasis::knotsType () const [inline]

Get the type of the knots generating algorithm.

```
175 {
176     return _knotsType;
177 }
```

6.33.3.5 double HxBSplineBasis::minT () const [inline]

Get the minimum value for t.

```
181 {
182     return _minT;
183 }
```

6.33.3.6 double HxBSplineBasis::maxT () const [inline]

Get the maximum value for t.

```
187 {
188     return _maxT;
189 }
```

6.33.3.7 double HxBSplineBasis::knot (int *i*) const [inline]

Get the value of the given knot.

```
208 {
209     return _knotsVec[i];
210 }
```

6.33.3.8 vector< double > HxBSplineBasis::allKnots () const [inline]

Get values of all knots.

```
214 {
215     return _knotsVec;
216 }
```

6.33.3.9 int HxBSplineBasis::numB () const [inline]

Get the number of basis functions.

```
193 {
194     switch ( curveType() ) {
195       case closed:
196             return _nIntervals;
```

```
197             break;
198       case openRepeatEndPoints:
199       case open:
200       default:  // this case should never happen!
201             return _nIntervals+degree();
202             break;
203     }
204 }
```

6.33.3.10 double HxBSplineBasis::B (int *i*, double *t*) const

Get value of given basis at path position t.

```
102 {
103     if ( t < minT() ) {
104         message("(B) invalid t - setting to minT()");
105         t = minT();
106     }
107     else    if ( t >= maxT() ) {
108             message("(B) invalid t - setting near to maxT()");
109             t = maxT() - EPS;
110         }
111     if ( ! inRange(i, 0, numB()) ) {
112         message("(B) invalid index - setting to 0");
113         i = 0;
114     }
115
116         // return 0 if i doesn't affect point at t
117     if ( isNullAt(i, t) )
118         return 0.0;
119
120         // Calculate basis
121     return nim(t, internalBasisIndex(i, t), degree()+1);
122 }
```

6.33.3.11 double HxBSplineBasis::dB (int *order*, int *i*, double *t*) const

Get derivative of given basis at path position t.

```
135 {
136     if ( t < minT() ) {
137         message("(dB) invalid t - setting to minT()");
138         t = minT();
139     }
140     else    if ( t >= maxT() ) {
141             message("(dB) invalid t - setting near to maxT()");
142             t = maxT() - EPS;
143         }
144     if ( ! inRange(i, 0, numB()) ) {
145         message("(dB) invalid index - setting to 0");
146         i = 0;
147     }
148     if ( ! inRange(order, 0, degree()-1) ) {
149         message("(dB) derivative not defined - setting to zero");
150         order = 0;
151     }
152
153         // return 0 if i doesn't affect point at t
154     if ( isNullAt(i, t) )
```

```
155         return 0.0;
156
157         // Calculate basis or derivative
158     int index = internalBasisIndex(i,t);
159     if ( order )
160         return dNim(order, t, index, degree()+1);
161     else    return nim(t, index, degree()+1);
162 }
```

6.33.3.12 HxBSplineInterval HxBSplineBasis::pathAffectedBy (int *i*) const

Get path interval affected by given basis.

```
171 {
172     if ( ! inRange(i, 0, numB()) ) {
173         message("(pathAffectedBy) invalid i - fixing to 0");
174         i = 0;
175     }
176
177     double t1 =  _knotsVec[i];
178     if ( t1 < minT() ) {
179         if ( curveType() == closed )
180             t1 = maxT() - absolute(minT() - t1);
181         else    t1 = minT(); //ZZZ
182     }
183     double t2 = _knotsVec[i+degree()+1];
184     if ( t2 >= maxT() ) {
185         if ( curveType() == closed )
186             t2 = minT() + (t2 - maxT());
187         else    t2 = maxT(); //ZZZ
188     }
189
190     HxBSplineInterval tmp( t1,t2, minT(), maxT(), curveType());
191
192     return tmp;
193 }
```

6.33.3.13 HxBSplineBasis HxBSplineBasis::insertKnot (double *t*, int *n* = 1) const

Insert one knot at given position.

```
203 {
204     if ( t < minT() ) {
205         message("(insertKnot) invalid t - setting to minT()");
206         t = minT();
207     }
208     else    if ( t >= maxT() ) {
209             message("(insertKnot) invalid t - setting near to maxT()");
210             t = maxT() - EPS;
211         }
212     HxBSplineBasis tmp(*this);
213     for (int i=degree(); i < tmp._knotsVec.size(); i++ )
214         if ( tmp._knotsVec[i] >= t ) {
215             tmp._nIntervals++;
216             tmp._knotsVec.insert( tmp._knotsVec.begin()+i, n, t);
217             tmp.correctEnds(i);
218             break;
219         }
220     return tmp;
221 }
```

6.33.3.14 STD_OSTREAM & HxBSplineBasis::dump (STD_OSTREAM & *os*) const

Dump the basis on the given stream.

```
229 {
230     os << "Curve Type: " << curveType();
231     os << "  Knots Type: " << knotsType();
232     os << "  Degree: " << degree();
233     os << "  Interval: [" << minT() << "," << maxT() << "]" << STD_ENDL;
234 
235     if ( _knotsVec.empty() ) {
236         os << "No knots";
237     } else {
238         os << "Knots Vector: " << _knotsVec.size() << " knots";
239         os << " (" << _nIntervals << " intervals,";
240         os << " N Basis=" << numB() << ")\n";
241         for ( int i = 0; i < _knotsVec.size(); i++ ) {
242             os << _knotsVec[i] << ", ";
243         }
244     }
245 
246     os << STD_ENDL;
247     return os;
248 }
```

The documentation for this class was generated from the following files:

- **HxBSplineBasis.h**
- HxBSplineBasis.c

6.34 HxBSplineCurve Class Reference

Class definition for BSpline curves.

`#include <HxBSplineCurve.h>`

Public Methods

- **HxBSplineCurve** ()

  Construct default curve.

- **HxBSplineCurve** (const **HxBSplineBasis** &basis, const HxPointSetR2 &cp)

  Construct a curve with given basis and control points.

- ~**HxBSplineCurve** ()

  Destructor.

- int **ident** () const

  Get the identifier.

- **HxBSplineBasis basis** () const

  Get the basis.

- HxBSplineType **curveType** () const

Get the curve type.

- int **degree** () const

  Get the degree of the curve.

- double **minT** () const

  Get the minimum value of T.

- double **maxT** () const

  Get the maximum value of T.

- **HxBSplineInterval getInterval** (double t1, double t2) const

  Get the curve interval determined by t1, t2.

- int **numP** () const

  Get the number of control points.

- **HxPointR2 P** (int i) const

  Get the coordinates of control point i.

- HxPointSetR2 **allP** () const

  Get the coordinates of all control points.

- **HxPolyline2d controlP** () const

  Get the control polygon.

- double **B** (int i, double t) const

  Get the value of basis i at path position t.

- double **dB** (int order, int i, double t) const

  Get derivative of basis i at path position t.

- **HxBSplineInterval pathAffectedBy** (int index) const

  Get the curve interval affected by given control point.

- vector< int > **PThatAffectCAt** (double t) const

  Get the indices of control points that affect the curve at path position t.

- vector< int > **PThatAffectCAt** (const **HxBSplineInterval** &interval) const

  Get the indices of control points that affect the curve inside given interval.

- **HxPointR2 C** (double t) const

  Get the curve point at path position t.

- **HxVectorR2 dC** (int order, double t) const

  Get the curve derivative at path position t.

- double **kAtC** (double t) const

  Get curvature at path position t.

- double **dTurnAngleAtC** (double t) const

  Get derivative of turning angle at path position t.

- double **length** (int n=50) const

  Get the total curve length based on distance between the given number of samples (n).

- double **length** (const **HxBSplineInterval** &interval, int n=50) const

  Get the length of the given curve interval.

- **HxPointR2 center** () const

  Get the center of control polygon.

- **HxPolyline2d sampleC** (int n=50) const

  Sample the curve at n points.

- HxBSplineCurve **changeAllP** (const HxPointSetR2 &p) const

  Replace the coordinates of all control points.

- HxBSplineCurve **translateAllP** (const **HxVectorR2** &v) const

  Translate all control points.

- HxBSplineCurve **scaleAllP** (double s) const

  Scale control polygon using center as origin.

- HxBSplineCurve **translateCurve** (const **HxVectorR2** &v, double t) const

  Translate the given point in the curve in the given direction modifying only one control point (NURBS book p.511-513).

- HxBSplineCurve **translateCurve** (const **HxPointR2** &pDest, double t) const

  Translate the given point in the curve to the given position modifying only one control point (NURBS book p.511-513).

- HxBSplineCurve **translateCurve2** (const **HxVectorR2** &v, double t) const

  Translate the given point in the curve to the given position modifying two control points (NURBS book p.511-513).

- HxBSplineCurve **insertKnot** (double t, int n=1) const

  Insert a given number of knots in the curve with corresponding control point without changing parametrization and geometry.

- STD_OSTREAM & **dump** (STD_OSTREAM &) const

  Dump the curve on the given stream.

Static Public Methods

- HxBSplineCurve **makeUniform** (**HxPolyline2d** cp, int degree)

  Make a curve with uniform knots.

- HxBSplineCurve **makeInterpolating** (**HxPolyline2d** cp)

  Make an interpolating curve (uniform knots).

6.34.1 Detailed Description

Class definition for BSpline curves.

Based on ”Curve and Surface Fitting with Splines”, by ”Dierckx,P.”, ”Oxford”, ”1993”, chapter ”Univariate Splines”, and ”The NURBS book”, ”Piegl, L. and Tiller, W.”, Springer, 1997.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 HxBSplineCurve::HxBSplineCurve ()

Construct default curve.

```
21 {
22     _ident = _nr++;
23     makeDefault();
24 }
```

6.34.2.2 HxBSplineCurve::HxBSplineCurve (const HxBSplineBasis & *basis*, const HxPointSetR2 & *vectorOfCP*)

Construct a curve with given basis and control points.

```
27                                      : _basis(basis)
28 {
29     _ident = _nr++;
30     if ( vectorOfCP.size() != numP() ) {
31         message("(constructor) invalid number of control points - making default");
32         makeDefault();
33     }
34     else
35         _PVec = vectorOfCP;
36 }
```

6.34.2.3 HxBSplineCurve::~HxBSplineCurve ()

Destructor.

```
64 {
65 }
```

6.34.3 Member Function Documentation

6.34.3.1 HxBSplineCurve HxBSplineCurve::makeUniform (HxPolyline2d *cp*, int *degree*) [static]

Make a curve with uniform knots.

```
40 {
41     HxBSplineType type = (cp.getClosed())? closed : openRepeatEndPoints;
42     HxPointSetR2 p = cp.getPoints();
43
```

```
44     int nInt;
45     if (cp.getClosed())
46         nInt = p.size();
47     else
48         nInt = p.size() - degree;
49
50     HxBSplineBasis base(type, degree, nInt, uniformKnots, 0, 1);
51     return HxBSplineCurve(base, p);
52 }
```

6.34.3.2 HxBSplineCurve HxBSplineCurve::makeInterpolating (HxPolyline2d *cp*) [static]

Make an interpolating curve (uniform knots).

```
56 {
57     HxLocalInterpol interpol(3, cp.getPoints(), cp.getClosed());
58     HxBSplineType type = (cp.getClosed())? closed : open;
59     HxBSplineBasis base(type, 3, interpol.allKnots());
60     return HxBSplineCurve(base, interpol.allP());
61 }
```

6.34.3.3 int HxBSplineCurve::ident () const [inline]

Get the identifier.

```
200 {
201     return _ident;
202 }
```

6.34.3.4 HxBSplineBasis HxBSplineCurve::basis () const [inline]

Get the basis.

```
206 {
207     return _basis;
208 }
```

6.34.3.5 HxBSplineType HxBSplineCurve::curveType () const [inline]

Get the curve type.

```
212 {
213     return _basis.curveType();
214 }
```

6.34.3.6 int HxBSplineCurve::degree () const [inline]

Get the degree of the curve.

```
218 {
219     return _basis.degree();
220 }
```

6.34.3.7 double HxBSplineCurve::minT () const [inline]

Get the minimum value of T.

```
224 {
225     return _basis.minT();
226 }
```

6.34.3.8 double HxBSplineCurve::maxT () const [inline]

Get the maximum value of T.

```
230 {
231     return _basis.maxT();
232 }
```

6.34.3.9 HxBSplineInterval HxBSplineCurve::getInterval (double *t1*, double *t2*) const [inline]

Get the curve interval determined by t1, t2.

```
236 {
237     return HxBSplineInterval(t1, t2,
238         _basis.minT(), _basis.maxT(), _basis.curveType());
239 }
```

6.34.3.10 int HxBSplineCurve::numP () const [inline]

Get the number of control points.

```
261 {
262     return _basis.numB();
263 }
```

6.34.3.11 HxPointR2 HxBSplineCurve::P (int *i*) const [inline]

Get the coordinates of control point i.

```
267 {
268     if ( i < 0 || i >= numP() ) {
269         message("(P) invalid index - setting to 0");
270         i = 0;
271     }
272     return _PVec[i];
273 }
```

6.34.3.12 HxPointSetR2 HxBSplineCurve::allP () const [inline]

Get the coordinates of all control points.

```
277 {
278     return _PVec;
279 }
```

6.34.3.13 HxPolyline2d HxBSplineCurve::controlP () const [inline]

Get the control polygon.

```
283 {
284     return HxPolyline2d(_PVec, (curveType() == closed));
285 }
```

6.34.3.14 double HxBSplineCurve::B (int *i*, double *t*) const [inline]

Get the value of basis i at path position t.

```
243 {
244     return _basis.B(i, t);
245 }
```

6.34.3.15 double HxBSplineCurve::dB (int *order*, int *i*, double *t*) const [inline]

Get derivative of basis i at path position t.

```
249 {
250     return _basis.dB(order, i, t);
251 }
```

6.34.3.16 HxBSplineInterval HxBSplineCurve::pathAffectedBy (int *index*) const [inline]

Get the curve interval affected by given control point.

```
255 {
256     return _basis.pathAffectedBy(index);
257 }
```

6.34.3.17 vector< int > HxBSplineCurve::PThatAffectCAt (double *t*) const

Get the indices of control points that affect the curve at path position t.

```
73 {
74     if ( t < minT() ) {
75         message("(PThatAffectCAt) invalid t - setting to minT()");
76         t = minT();
77     }
78     if ( t >= maxT() ) {
79         message("(PThatAffectCAt) invalid t - setting near to maxT()");
80         t = maxT() - EPS;
81     }
82
83     vector<int> b;
84     int index = _basis.whichInterval(t);
85     for ( int i=index; i <= index + degree(); i++)
86         b.push_back(indexPVec(i));
87
88     return b;
89 }
```

6.34.3.18 vector< int > HxBSplineCurve::PThatAffectCAt (const HxBSplineInterval & *interval*) const

Get the indices of control points that affect the curve inside given interval.

```
97 {
98     int i = indexPVec(_basis.whichInterval(interval.begin()));
99     int f = indexPVec(_basis.whichInterval(interval.end()) + degree());
100
101     vector<int> b;
102     while (i != f) {
103         b.push_back(indexPVec(i));
104         if ( ++i >= numP() )
105             i = 0;      // wrap closed curve
106     };
107     b.push_back(indexPVec(f));
108     return b;
109 }
```

6.34.3.19 HxPointR2 HxBSplineCurve::C (double *t*) const [inline]

Get the curve point at path position t.

```
295 {
296     HxVectorR2 tmp = dC( 0, t);
297     return HxPointR2(tmp.x(), tmp.y());
298 }
```

6.34.3.20 HxVectorR2 HxBSplineCurve::dC (int *order*, double *t*) const

Get the curve derivative at path position t.

```
121 {
122     if ( t < minT() ) {
123         message("(dC) invalid t - setting to minT()");
124         t = minT();
125     }
126     if ( t >= maxT() ) {
127         message("(dC) invalid t - setting near to maxT()");
128         t = maxT() - EPS;
129     }
130     if ( ! inRange(order, 0, degree()-1) ) {
131         message("(dC) derivative not defined - fixing to 1st");
132         order = 1;
133     }
134
135     double multterm = 1;
136     int i;
137     for (i = 1 ; i <= order ; i++)
138         multterm *= (degree()+1-i);
139
140     int index = _basis.whichInterval(t);
141     double sumtermx = 0, sumtermy = 0;
142     for (i = index+order ; i <= index + degree(); i++) {
143         double nterm = _basis.nim(t,i,degree()+1-order);
144         HxPointR2 cterm = civ(i,order);
145         sumtermx += cterm.x() * nterm;
```

```
146         sumtermy += cterm.y() * nterm;
147     }
148
149     return HxVectorR2(multterm*sumtermx, multterm*sumtermy);
150 }
```

6.34.3.21 double HxBSplineCurve::kAtC (double *t*) const

Get curvature at path position t.

```
158 {
159     if ( degree() <= 2 ) {
160         message("(kAtC) can't compute curvature - returning 0");
161         return 0;
162     }
163
164     HxVectorR2 d1 = dC(1,t);
165     HxVectorR2 d2 = dC(2,t);
166     double l2 = d1.squaredMagnitude();
167     double l = sqrt(l2);
168     return (d1.x() * d2.y() - d2.x() * d1.y()) / (l * l2);
169 }
```

6.34.3.22 double HxBSplineCurve::dTurnAngleAtC (double *t*) const

Get derivative of turning angle at path position t.

```
177 {
178     if ( degree() <= 2 ) {
179         message("(dTurnAngleAtC) can't compute curvature - returning 0");
180         return 0;
181     }
182
183     HxVectorR2 d1 = dC(1,t);
184     HxVectorR2 d2 = dC(2,t);
185     double l2 = d1.squaredMagnitude();
186     return (d1.x() * d2.y() - d2.x() * d1.y()) / l2;
187 }
```

6.34.3.23 double HxBSplineCurve::length (int *np* = 50) const

Get the total curve length based on distance between the given number of samples (n).

```
231 {
232     double length = 0.0;
233     double d = (maxT()-EPS - minT() ) / np;
234
235     if ( degree() > 1 ) {
236         for ( double t = minT(); t < maxT(); t+= d)
237             length += dC(1,t).magnitude();
238         return length*d;
239     } else  {
240         HxPointR2 p1 = C(minT());
241         for ( double t = minT(); t < maxT(); t+= d) {
242             HxPointR2 p2 = C(t);
```

```
243             HxVectorR2 v(p2,p1);
244             length += v.magnitude();
245             p1 = p2;
246         }
247         if ( curveType() == closed ) {
248             HxVectorR2 v(C(minT()), p1);
249             length += v.magnitude();
250         }
251         return length;
252     }
253 }
```

6.34.3.24 double HxBSplineCurve::length (const HxBSplineInterval & *part*, int *np* = 50) const

Get the length of the given curve interval.

```
197 {
198     double length = 0.0;
199     double d = part.length()/ np;
200 
201     if ( degree() > 1 ) {
202         double t=part.begin();
203         for ( int i=0; i < np; i++) {
204             length += dC(1,t).magnitude();
205             t=part.next(t, d);
206         }
207         return length*d;
208     } else  {
209         HxPointR2 p1 = C(part.begin());
210         HxPointR2 p2;
211         double t= part.next(part.begin(),d);
212         for ( int i=0; i < np; i++) {
213             p2 = C(t);
214             HxVectorR2 v(p2,p1);
215             length += v.magnitude();
216             p1 = p2;
217             t=part.next(t, d);
218         }
219         return length;
220     }
221 }
```

6.34.3.25 HxPointR2 HxBSplineCurve::center () const

Get the center of control polygon.

```
278 {
279     double sumX=0, sumY=0;
280     for (int i=0; i < _PVec.size(); i++) {
281         sumX += _PVec[i].x();
282         sumY += _PVec[i].y();
283     }
284     return HxPointR2( sumX/_PVec.size(), sumY/_PVec.size());
285 }
```

6.34.3.26 HxPolyline2d HxBSplineCurve::sampleC (int *np* = 50) const

Sample the curve at n points.

```
261 {
262     double d = (maxT()-EPS - minT() ) / np;
263     HxPointSetR2 c;
264     HxPointR2 p1 = C(minT());
265     for (double t=minT() ; t<maxT() ; t+=d) {
266         c.push_back(C(t));
267     }
268     return HxPolyline2d(c, (curveType() == closed));
269 }
```

6.34.3.27 HxBSplineCurve HxBSplineCurve::changeAllP (const HxPointSetR2 & *p*) const [inline]

Replace the coordinates of all control points.

```
289 {
290     return HxBSplineCurve(_basis, p);
291 }
```

6.34.3.28 HxBSplineCurve HxBSplineCurve::translateAllP (const HxVectorR2 & *v*) const

Translate all control points.

```
293 {
294     HxPointSetR2 tmp(_PVec.size());
295     for (int i=0; i < tmp.size(); i++) {
296         tmp[i] = _PVec[i].add(v);
297     }
298     return HxBSplineCurve( _basis, tmp);
299 }
```

6.34.3.29 HxBSplineCurve HxBSplineCurve::scaleAllP (double *s*) const

Scale control polygon using center as origin.

```
309 {
310     HxPointSetR2 tmp(_PVec.size());
311     HxPointR2 origin(center());
312     for (int i=0; i < tmp.size(); i++) {
313         HxVectorR2 v(_PVec[i],origin);
314         tmp[i] =  origin.add(v.mul(s));
315     }
316     return HxBSplineCurve( _basis, tmp);
317 }
```

6.34.3.30 HxBSplineCurve HxBSplineCurve::translateCurve (const HxVectorR2 & *vec*, double *t*) const

Translate the given point in the curve in the given direction modifying only one control point (NURBS book p.511-513).

```
329 {
330     HxBSplineCurve tmp(*this);
331     if ( absolute(basis().nearestKnot(t)-t) > EPS )
332         tmp= insertKnot(t);
333 
334         // determine CP to move and vector
335     int i = tmp._basis.maxBasis(t);
336     double d = vec.magnitude();
337     double alpha =  d / tmp._basis.B(i, t);
338     HxVectorR2 V = vec.div(d).mul(alpha);
339 
340         // update control point
341     int index = tmp.indexPVec(i);
342     tmp._PVec[index] = tmp._PVec[index].add(V);
343 
344     return tmp;
345 }
```

6.34.3.31 HxBSplineCurve HxBSplineCurve::translateCurve (const HxPointR2 & *pDest*, double *t*) const [inline]

Translate the given point in the curve to the given position modifying only one control point (NURBS book p.511-513).

```
302 {
303     return translateCurve( HxVectorR2(pDest, C(t)), t);
304 }
```

6.34.3.32 HxBSplineCurve HxBSplineCurve::translateCurve2 (const HxVectorR2 & *vec*, double *t*) const

Translate the given point in the curve to the given position modifying two control points (NURBS book p.511-513).

```
357 {
358     HxBSplineCurve tmp(*this);
359     HxBSplineBasis tmpBasis(tmp._basis);
360 
361         // determine two CP to move
362     int k = tmpBasis.maxBasis(t);
363     double kNode =  tmpBasis.node(k);
364     while ( kNode >= t ) {
365         k--;
366         if ( k < 0 )
367             k = tmpBasis.nIntervals()-1;
368         kNode =  tmpBasis.node(k);
369     }
370 
371     int k1;
372     if ( k >= tmpBasis.nIntervals() )
```

```
373         k1 = 0;
374     else    k1 = k+1;
375     double k1Node =  tmpBasis.node(k1);
376 
377         // determine proportion of each CP
378     HxBSplineInterval intA( kNode, t, minT(), maxT(), curveType());
379     HxBSplineInterval intB( kNode, k1Node, minT(), maxT(), curveType());
380     double l = intA.length() / intB.length();
381 
382         // determine motion vector
383     double d = vec.magnitude();
384     double alpha =  d / ( (1-l)*tmpBasis.B(k, t) + l*tmpBasis.B(k1, t) );
385     HxVectorR2 V = vec.div(d).mul(alpha);
386 
387         // update control points
388     int index = tmp.indexPVec(k);
389     tmp._PVec[index] = tmp._PVec[index].add(V.mul(1-l));
390     index = tmp.indexPVec(k1);
391     tmp._PVec[index] = tmp._PVec[index].add(V.mul(l));
392 
393     return tmp;
394 }
```

6.34.3.33 HxBSplineCurve HxBSplineCurve::insertKnot (double *t*, int *n* = 1) const

Insert a given number of knots in the curve with corresponding control point without changing parametrization and geometry.

```
404 {
405     if ( t < minT() ) {
406         message("(insertKnot) invalid knot - setting to minT()");
407         t = minT();
408     }
409     if ( t >= maxT() ) {
410         message("(insertKnot) invalid knot - setting near to maxT()");
411         t = maxT() - EPS;
412     }
413     if ( n != 1 ) {
414         message("(insertKnot) multiple knot insertion not supported - inserting 1 knot");
415         n=1;
416     }
417 
418     HxBSplineBasis newB = _basis.insertKnot(t,n);
419     HxPointSetR2 newP = _PVec;
420 
421     vector<HxVectorR2> aux(degree()+1);
422     int ii1 = _basis.whichInterval(t);
423     int ii2 = _basis.internalBasisIndex(ii1,t);
424     int k = ii2 + degree();
425     int j =1;
426     int s =0;   // multiplicity
427 
428     int i;
429     for (i=0; i <= degree()-s; i++) {
430         HxPointR2 p = newP[indexPVec(k-degree()+i)];
431         aux[i] = HxVectorR2(p.x(), p.y());
432     }
433 
434     int L = k - degree() + j;
435 //  newP.insert(newP.begin()+indexPVec(L), n, HxPointR2(0,0));
436     newP.insert(newP.begin()+L, n, HxPointR2(0,0));
437     for ( i=0; i <= degree() -j-s; i++) {
```

```
438         double a = (t - _basis._knotsVec[L+i]) /
439             (_basis._knotsVec[i+k+1] - _basis._knotsVec[L+i]);
440         aux[i] = aux[i+1].mul(a).add(aux[i].mul(1.0-a));
441     }
442
443     HxBSplineCurve tmp(newB, newP);
444     for ( i=L; i <= k-s; i++ ) {
445         HxVectorR2 v(aux[i-L]);
446         tmp._PVec[tmp.indexPVec(i)] = HxPointR2(v.x(), v.y());
447     }
448
449     return tmp;
450 }
```

6.34.3.34 STD_OSTREAM & HxBSplineCurve::dump (STD_OSTREAM & *os*) const

Dump the curve on the given stream.

```
458 {
459     _basis.dump(os);
460
461     if ( _PVec.empty() ) {
462         os << "\nNo Control Points";
463     } else {
464         os << "P Vector: " << _PVec.size() << " points\n";
465         for ( int i = 0; i < _PVec.size(); i++ ) {
466             os <<"("<< _PVec[i].x() << ","<<_PVec[i].y()<<") ";
467         }
468     }
469
470     os << STD_ENDL;
471     return os;
472 }
```

The documentation for this class was generated from the following files:

- **HxBSplineCurve.h**
- HxBSplineCurve.c

6.35 HxBSplineInterval Class Reference

Class definition for HxBSplineInterval to facilitate manipulation of path intervals in open and closed curves.

`#include <HxBSplineInterval.h>`

Public Methods

- **HxBSplineInterval** ()

  Default constructor.

- **HxBSplineInterval** (double b, double e, double min, double max, HxBSplineType type)

  Construct interval [b,e) in a curve with total path from [min,max) and given type.

- **HxBSplineInterval** (double b, double e, double min, double max, int closed)

Construct interval [b,e) in a curve with total path from [min,max) and given type.

- ~**HxBSplineInterval** ()

  Destructor.

- double **begin** () const

  first t in interval.

- double **end** () const

  last t in interval.

- int **contains** (double t) const

  Determine if the path parameter t is inside this interval.

- double **next** (double t, double delta) const

  returns t+delta, with wrapping for closed intervals.

- double **prev** (double t, double delta) const

  returns t-delta, with wrapping for closed intervals.

- double **length** () const

  path lenght of interval.

- double **middle** () const

  middle t of interval.

- double **ratio** (double r) const

  t corresponding to given proportion of interval.

- int **isClosed** () const

  1 if interval is wrapped.

- HxBSplineInterval **cropBegin** (double t) const

  cut begin of interval: [t,e).

- HxBSplineInterval **cropEnd** (double t) const

  cut end of interval: [b,t).

- HxBSplineInterval **part** (double t1, double t2) const

  get subinterval [t1,t2).

6.35.1 Detailed Description

Class definition for HxBSplineInterval to facilitate manipulation of path intervals in open and closed curves.

path interval (oriented). interval = [first, second). For open paths, first $<$ second (always). For closed paths, first $>$ second indicates wrap around t=min, max.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 HxBSplineInterval::HxBSplineInterval () [inline]

Default constructor.

```
99                                      : pair<double,double>()
100 {
101     _min = 0;
102     _max = 0;
103     _wrap = 0;
104 }
```

6.35.2.2 HxBSplineInterval::HxBSplineInterval (double *b*, double *e*, double *min*, double *max*, HxBSplineType *type*) [inline]

Construct interval [b,e) in a curve with total path from [min,max) and given type.

```
109     : pair<double,double>(b,e)
110 {
111     _min = min;
112     _max = max;
113     _wrap = (type == closed);
114 }
```

6.35.2.3 HxBSplineInterval::HxBSplineInterval (double *b*, double *e*, double *min*, double *max*, int *wrap*) [inline]

Construct interval [b,e) in a curve with total path from [min,max) and given type.

```
119     : pair<double,double>(b,e)
120 {
121     _min = min;
122     _max = max;
123     _wrap = wrap;
124 }
```

6.35.2.4 HxBSplineInterval::~HxBSplineInterval () [inline]

Destructor.

```
128 {
129 }
```

6.35.3 Member Function Documentation

6.35.3.1 double HxBSplineInterval::begin () const [inline]

first t in interval.

```
133 {
134     return first;
135 }
```

6.35.3.2 double HxBSplineInterval::end () const [inline]

last t in interval.

```
139 {
140     return second;
141 }
```

6.35.3.3 int HxBSplineInterval::contains (double *t*) const [inline]

Determine if the path parameter t is inside this interval.

Consider wrapping for closed curves. interval = [first, second)

```
145 {
146     if ( first == second )      // special case?
147         return (t >= _min && t < _max);
148     if ( first > second )       // is wrapped?
149         return (t >= first && t < _max) || (t >= _min && t < second);
150     else    return (t >= first && t < second);
151 }
```

6.35.3.4 double HxBSplineInterval::next (double *t*, double *delta*) const [inline]

returns t+delta, with wrapping for closed intervals.

```
155 {
156     t += delta;
157     if ( t >= _max && _wrap )
158         t = _min + (t - _max);
159     return t;
160 }
```

6.35.3.5 double HxBSplineInterval::prev (double *t*, double *delta*) const [inline]

returns t-delta, with wrapping for closed intervals.

```
164 {
165     t -= delta;
166     if ( t <  _min && _wrap )
167         t = _max - (_min - t);
168     return t;
169 }
```

6.35.3.6 double HxBSplineInterval::length () const [inline]

path lenght of interval.

```
173 {
174     double tmp = second - first;
175     if ( tmp < 0 && _wrap )
```

```
176         return (_max - EPS - first) + (second - _min);
177     else    if ( tmp < EPS )
178             return EPS;
179         else    return tmp - EPS;
180 }
```

6.35.3.7 double HxBSplineInterval::middle () const [inline]

middle t of interval.

```
184 {
185     return ratio(0.5);
186 }
```

6.35.3.8 double HxBSplineInterval::ratio (double *r*) const [inline]

t corresponding to given proportion of interval.

```
190 {
191     double t = length() * r + first;
192     if ( t >= _max )
193         t = _min + (t - _max);
194     return t;
195 }
```

6.35.3.9 int HxBSplineInterval::isClosed () const [inline]

1 if interval is wrapped.

```
199 {
200     return _wrap;
201 }
```

6.35.3.10 HxBSplineInterval HxBSplineInterval::cropBegin (double *t*) const [inline]

cut begin of interval: [t,e).

```
205 {
206     HxBSplineInterval tmp(t, second, _min, _max, _wrap);
207     return tmp;
208 }
```

6.35.3.11 HxBSplineInterval HxBSplineInterval::cropEnd (double *t*) const [inline]

cut end of interval: [b,t).

```
212 {
213     HxBSplineInterval tmp(first, t, _min, _max, _wrap);
214     return tmp;
215 }
```

6.35.3.12 HxBSplineInterval HxBSplineInterval::part (double *t1*, double *t2*) const [inline]

get subinterval [t1,t2).

```
219 {
220     HxBSplineInterval tmp(t1, t2, _min, _max, closed);
221     tmp._wrap = _wrap;
222     return tmp;
223 }
```

The documentation for this class was generated from the following file:

- **HxBSplineInterval.h**

6.36 HxColor Class Reference

Class definition color semantics.

`#include <HxColor.h>`

Public Methods

- **HxColor** ()
  Default constructor.
- **HxColor** (**HxVec3Double** color, **HxColorModel** space=RGB)
  Construction from given color in color space.
- **HxColor** (const HxColor &rhs)
  Copy constructor.
- HxColor & **operator=** (const HxColor &rhs)
  Assignment operator.
- const **HxVec3Double** & **value** () const
  Return the value.
- HxColor **convert** (const **HxColorModel** space) const
  General color space convertor.
- HxColor **toRGB** () const
  to RGB.
- HxColor **toCMY** () const
  to CMY.
- HxColor **toXYZ** () const
  to XYZ.
- HxColor **toLab** () const

to Lab.

- HxColor **toLuv** () const

  to Luv.

- HxColor **toOOO** () const

  to OOO.

- HxColor **toHSI** () const

  to HSI.

- int **operator==** (const HxColor &v) const

  Equal.

- int **operator!=** (const HxColor &v) const

  Not equal.

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print color on stream.

- **HxString toString** () const

  Color as a string.

6.36.1 Detailed Description

Class definition color semantics.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 HxColor::HxColor () [inline]

Default constructor.

```
102     : _value(0,0,0), _space(RGB)
103 {
104 }
```

6.36.2.2 HxColor::HxColor (HxVec3Double *color*, HxColorModel *space* = RGB) [inline]

Construction from given color in color space.

```
108     : _value(color), _space(space)
109 {
110 }
```

6.36.2.3 HxColor::HxColor (const HxColor & *rhs*) [inline]

Copy constructor.

```
114     : _value(rhs._value), _space(rhs._space)
115 {
116 }
```

6.36.3 Member Function Documentation

6.36.3.1 HxColor & HxColor::operator= (const HxColor & *rhs*) [inline]

Assignment operator.

```
120 {
121     _value = rhs._value;
122     _space = rhs._space;
123     return *this;
124 }
```

6.36.3.2 const HxVec3Double & HxColor::value () const [inline]

Return the value.

```
128 {
129     return _value;
130 }
```

6.36.3.3 HxColor HxColor::convert (const HxColorModel *space*) const [inline]

General color space convertor.

See also : **Color conversion functions** (p. 7).

```
134 {
135     switch (space) {
136     case RGB: return toRGB();
137     case CMY: return toCMY();
138     case XYZ: return toXYZ();
139     case Lab: return toLab();
140     case Luv: return toLuv();
141     case OOO: return toOOO();
142     case HSI: return toHSI();
143     }
144     return HxColor();
145 }
```

6.36.3.4 HxColor HxColor::toRGB () const

to RGB.

```
50 {
51     switch (_space) {
52     case RGB:
53         return *this;
54     case CMY:
55         return HxColor(HxColCMY2RGB(_value), RGB);
56     case XYZ:
57         return HxColor(HxColXYZ2RGB(_value), RGB);
58     case Lab: {
59         HxVec3Double xyz = HxColLab2XYZ(_value);
60         return HxColor(HxColXYZ2RGB(xyz), RGB);
61       }
62     case Luv: {
63         HxVec3Double xyz = HxColLuv2XYZ(_value);
64         return HxColor(HxColXYZ2RGB(xyz), RGB);
65       }
66     case OOO:
67         return HxColor(HxColOOO2RGB(_value), RGB);
68     case HSI:
69         return HxColor(HxColHSI2RGB(_value), RGB);
70     }
71
72     return HxColor();
73 }
```

6.36.3.5 HxColor HxColor::toCMY () const

to CMY.

```
78 {
79     switch (_space) {
80     case RGB:
81         return HxColor(HxColRGB2CMY(_value), CMY);
82     case CMY:
83         return *this;
84     case XYZ:
85         return HxColor(HxColXYZ2CMY(_value), CMY);
86     case Lab: {
87         HxVec3Double xyz = HxColLab2XYZ(_value);
88         return HxColor(HxColXYZ2CMY(xyz), CMY);
89       }
90     case Luv: {
91         HxVec3Double xyz = HxColLuv2XYZ(_value);
92         return HxColor(HxColXYZ2CMY(xyz), CMY);
93       }
94     case OOO: {
95         HxVec3Double rgb = HxColOOO2RGB(_value);
96         return HxColor(HxColRGB2CMY(rgb), CMY);
97       }
98     case HSI: {
99         HxVec3Double rgb = HxColHSI2RGB(_value);
100         return HxColor(HxColRGB2CMY(rgb), CMY);
101       }
102     }
103
104     return HxColor();
105 }
```

6.36.3.6 HxColor HxColor::toXYZ () const

to XYZ.

```
109 {
110     switch (_space) {
111     case RGB:
112         return HxColor(HxColRGB2XYZ(_value), XYZ);
113     case CMY:
114         return HxColor(HxColCMY2XYZ(_value), XYZ);
115     case XYZ:
116         return *this;
117     case Lab:
118         return HxColor(HxColLab2XYZ(_value), XYZ);
119     case Luv:
120         return HxColor(HxColLuv2XYZ(_value), XYZ);
121     case OOO:
122         return HxColor(HxColOOO2XYZ(_value), XYZ);
123     case HSI: {
124         HxVec3Double rgb = HxColHSI2RGB(_value);
125         return HxColor(HxColRGB2XYZ(rgb), XYZ);
126       }
127     }
128 
129     return HxColor();
130 }
```

6.36.3.7 HxColor HxColor::toLab () const

to Lab.

```
134 {
135     switch (_space) {
136     case RGB: {
137         HxVec3Double xyz = HxColRGB2XYZ(_value);
138         return HxColor(HxColXYZ2Lab(xyz), Lab);
139     }
140     case CMY: {
141         HxVec3Double xyz = HxColCMY2XYZ(_value);
142         return HxColor(HxColXYZ2Lab(xyz), Lab);
143     }
144     case XYZ:
145         return HxColor(HxColXYZ2Lab(_value), Lab);
146     case Lab:
147         return *this;
148     case Luv: {
149         HxVec3Double xyz = HxColLuv2XYZ(_value);
150         return HxColor(HxColXYZ2Lab(xyz), Lab);
151     }
152     case OOO: {
153         HxVec3Double xyz = HxColOOO2XYZ(_value);
154         return HxColor(HxColXYZ2Lab(xyz), Lab);
155     }
156     case HSI: {
157         HxVec3Double rgb = HxColHSI2RGB(_value);
158         HxVec3Double xyz = HxColRGB2XYZ(rgb);
159         return HxColor(HxColXYZ2Lab(xyz), Lab);
160       }
161     }
162 
163     return HxColor();
164 }
```

6.36.3.8 HxColor HxColor::toLuv () const

to Luv.

```
169 {
170     switch (_space) {
171     case RGB: {
172         HxVec3Double xyz = HxColRGB2XYZ(_value);
173         return HxColor(HxColXYZ2Luv(xyz), Luv);
174     }
175     case CMY: {
176         HxVec3Double xyz = HxColCMY2XYZ(_value);
177         return HxColor(HxColXYZ2Luv(xyz), Luv);
178     }
179     case XYZ:
180         return HxColor(HxColXYZ2Luv(_value), Luv);
181     case Lab: {
182         HxVec3Double xyz = HxColLab2XYZ(_value);
183         return HxColor(HxColXYZ2Luv(xyz), Luv);
184     }
185     case Luv:
186         return *this;
187     case OOO: {
188         HxVec3Double xyz = HxColOOO2XYZ(_value);
189         return HxColor(HxColXYZ2Luv(xyz), Luv);
190     }
191     case HSI: {
192         HxVec3Double rgb = HxColHSI2RGB(_value);
193         HxVec3Double xyz = HxColRGB2XYZ(rgb);
194         return HxColor(HxColXYZ2Luv(xyz), Luv);
195       }
196     }
197
198     return HxColor();
199 }
```

6.36.3.9 HxColor HxColor::toOOO () const

to OOO.

```
203 {
204     switch (_space) {
205     case RGB:
206         return HxColor(HxColRGB2OOO(_value), OOO);
207     case CMY: {
208         HxVec3Double rgb = HxColCMY2RGB(_value);
209         return HxColor(HxColRGB2OOO(rgb), OOO);
210       }
211     case XYZ:
212         return HxColor(HxColXYZ2OOO(_value), OOO);
213     case Lab: {
214         HxVec3Double xyz = HxColLab2XYZ(_value);
215         return HxColor(HxColXYZ2OOO(_value), OOO);
216     }
217     case Luv: {
218         HxVec3Double xyz = HxColLuv2XYZ(_value);
219         return HxColor(HxColXYZ2OOO(_value), OOO);
220     }
221     case OOO:
222         return *this;
223     case HSI: {
```

```
224         HxVec3Double rgb = HxColHSI2RGB(_value);
225         return HxColor(HxColRGB2OOO(rgb), OOO);
226       }
227     }
228
229     return HxColor();
230 }
```

6.36.3.10 HxColor HxColor::toHSI () const

to HSI.

```
234 {
235     switch (_space) {
236     case RGB:
237         return HxColor(HxColRGB2HSI(_value), HSI);
238     case CMY: {
239         HxVec3Double rgb = HxColCMY2RGB(_value);
240         return HxColor(HxColRGB2HSI(rgb), HSI);
241       }
242     case XYZ: {
243         HxVec3Double rgb = HxColXYZ2RGB(_value);
244         return HxColor(HxColRGB2HSI(rgb), HSI);
245       }
246     case Lab: {
247         HxVec3Double xyz = HxColLab2XYZ(_value);
248         HxVec3Double rgb = HxColXYZ2RGB(xyz);
249         return HxColor(HxColRGB2HSI(rgb), HSI);
250       }
251     case Luv: {
252         HxVec3Double xyz = HxColLuv2XYZ(_value);
253         HxVec3Double rgb = HxColXYZ2RGB(xyz);
254         return HxColor(HxColRGB2HSI(rgb), HSI);
255       }
256     case OOO: {
257         HxVec3Double rgb = HxColOOO2RGB(_value);
258         return HxColor(HxColRGB2HSI(rgb), HSI);
259       }
260     case HSI:
261         return *this;
262     }
263
264     return HxColor();
265 }
```

6.36.3.11 int HxColor::operator== (const HxColor & *c*) const [inline]

Equal.

```
149 {
150     return (_space == c._space) && (_value == c._value);
151 }
```

6.36.3.12 int HxColor::operator!= (const HxColor & *c*) const [inline]

Not equal.

```
155 {
156     return !(*this == c);
157 }
```

6.36.3.13 STD_OSTREAM & HxColor::put (STD_OSTREAM & *os*) const [inline]

Print color on stream.

For global operator<<

```
89 {
90     return os << toString();
91 }
```

6.36.3.14 HxString HxColor::toString () const

Color as a string.

```
17 {
18     HxString colspace;
19
20     switch (_space) {
21     case RGB:
22         colspace = "RGB";
23         break;
24     case CMY:
25         colspace = "CMY";
26         break;
27     case XYZ:
28         colspace = "XYZ";
29         break;
30     case Lab:
31         colspace = "Lab";
32         break;
33     case Luv:
34         colspace = "Luv";
35         break;
36     case OOO:
37         colspace = "OOO";
38         break;
39     case HSI:
40         colspace = "HSI";
41         break;
42     default:
43         colspace = "(Unknown)";
44     }
45     return colspace + _value.toString();
46 }
```

The documentation for this class was generated from the following files:

- **HxColor.h**
- HxColor.c

6.37 HxComplex Class Reference

Class definition complex.

`#include <HxComplex.h>`

Constructors

- **HxComplex** ()
  Default constructor.
- **HxComplex** (double re, double im)
  Conversion from native type.
- **HxComplex** (const HxComplex &rhs)
  Copy constructor.

Inquiry

- int **dim** () const
  Dimensionality.
- double **x** () const
  Real Value.
- double **y** () const
  Imaginary Value.
- double **getValue** (int dimension) const
  Element in given dimension.
- void **setValue** (int dimension, double value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 677).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 785).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 766).
- **operator HxVec3Int** () const

Cast to **HxVec3Int** (p. 824).

- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 804).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxComplex &v) const
  Equal.
- int **operator!=** (const HxComplex &v) const
  Not equal.
- int **operator<** (const HxComplex &v) const
  Less than.
- int **operator<=** (const HxComplex &v) const
  Less equal.
- int **operator>** (const HxComplex &v) const
  Greater than.
- int **operator>=** (const HxComplex &v) const
  Greater equal.
- const HxComplex **SMALL_VAL** = HxComplex(0, 0)
  A small value w.r.t to the comparison operators "<" and ">".
- const HxComplex **LARGE_VAL** = HxComplex(1e300, 1e300)
  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxComplex **operator-** () const
  Negation.
- HxComplex **complement** () const
  Complement.
- HxComplex **conjugate** () const
  Conjugate.
- **HxScalarDouble abs** () const

Absolute value.

- **HxScalarDouble arg** () const
  Argument.
- HxComplex **ceil** () const
  Ceiling.
- HxComplex **floor** () const
  Floor.
- HxComplex **round** () const
  Round.
- HxComplex **sum** () const
  Sum.
- HxComplex **product** () const
  Product.
- **HxScalarDouble min** () const
  Minimum.
- **HxScalarDouble max** () const
  Maximum.
- **HxScalarDouble norm1** () const
  L1 norm.
- **HxScalarDouble norm2** () const
  L2 norm.
- **HxScalarDouble normInf** () const
  L infinity norm.
- HxComplex **sqrt** () const
  Square root.
- HxComplex **sin** () const
  Sine.
- HxComplex **cos** () const
  Cosine.
- HxComplex **tan** () const
  Tangent.
- HxComplex **asin** () const
  Arc sine.

- HxComplex **acos** () const
  Arc cosine.
- HxComplex **atan** () const
  Arc tangent.
- HxComplex **atan2** () const
  Arc tangent.
- HxComplex **sinh** () const
  Hyperbolic sine.
- HxComplex **cosh** () const
  Hyperbolic cosine.
- HxComplex **tanh** () const
  Hyperbolic tangent.
- HxComplex **exp** () const
  Exponent.
- HxComplex **log** () const
  Natural logarithm.
- HxComplex **log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxComplex & **operator+=** (const HxComplex &v)
  Addition and assignment.
- HxComplex & **operator-=** (const HxComplex &v)
  Subtraction and assignment.
- HxComplex & **operator** *= (const HxComplex &v)
  Multiplication and assignment.
- HxComplex & **operator/=** (const HxComplex &v)
  Division and assignment.
- HxComplex **min** (const HxComplex &v) const
  Minimum.
- HxComplex & **minAssign** (const HxComplex &v)
  Minimum and assignment.

- HxComplex **max** (const HxComplex &v) const
  Maximum.
- HxComplex & **maxAssign** (const HxComplex &v)
  Maximum and assignment.
- HxComplex **inf** (const HxComplex &v) const
  Infimum.
- HxComplex & **infAssign** (const HxComplex &v)
  Infimum and assignment.
- HxComplex **sup** (const HxComplex &v) const
  Supremum.
- HxComplex & **supAssign** (const HxComplex &v)
  Supremum and assignment.
- HxComplex **pow** (const HxComplex &v) const
  Power.
- HxComplex **mod** (const HxComplex &v) const
  Modulo.
- HxComplex **and** (const HxComplex &v) const
  And.
- HxComplex **or** (const HxComplex &v) const
  Or.
- HxComplex **xor** (const HxComplex &v) const
  Xor.
- HxComplex **leftShift** (const HxComplex &v) const
  Left shift.
- HxComplex **rightShift** (const HxComplex &v) const
  Right shift.
- **HxScalarDouble dot** (const HxComplex &v) const
  Dot product.
- HxComplex **cross** (const HxComplex &v) const
  Cross product.
- HxComplex **operator+** (const HxComplex &v1, const HxComplex &v2)
  Addition.
- HxComplex **operator-** (const HxComplex &v1, const HxComplex &v2)

Subtraction.

- HxComplex **operator** ∗ (const HxComplex &v1, const HxComplex &v2)

  Multiplication.

- HxComplex **operator/** (const HxComplex &v1, const HxComplex &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void ∗ **operator new** (size_t, void ∗=0)
- HxComplex & **operator=** (const HxComplex &rhs)

6.37.1 Detailed Description

Class definition complex.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 HxComplex::HxComplex () [inline]

Default constructor.

```
330 {
331 }
```

6.37.2.2 HxComplex::HxComplex (double *re*, double *im*) [inline]

Conversion from native type.

```
335 {
336     _values[0] = re;
337     _values[1] = im;
338 }
```

6.37.2.3 HxComplex::HxComplex (const HxComplex & *v*) [inline]

Copy constructor.

```
342 {
343     _values[0] = v._values[0];
344     _values[1] = v._values[1];
345 }
```

6.37.3 Member Function Documentation

6.37.3.1 int HxComplex::dim () const [inline]

Dimensionality.

```
363 {
364     return 2;
365 }
```

6.37.3.2 double HxComplex::x () const [inline]

Real Value.

```
369 {
370     return _values[0];
371 }
```

6.37.3.3 double HxComplex::y () const [inline]

Imaginary Value.

```
375 {
376     return _values[1];
377 }
```

6.37.3.4 double HxComplex::getValue (int *dim*) const [inline]

Element in given dimension.

```
381 {
382     return _values[dim - 1];
383 }
```

6.37.3.5 HxComplex::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
27 {
28     return (int) _values[0];
29 }
```

6.37.3.6 HxComplex::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
32 {
33     return _values[0];
34 }
```

6.37.3.7 HxComplex::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
37 {
38     return HxVec2Int(int(_values[0]), int(_values[1]));
39 }
```

6.37.3.8 HxComplex::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
42 {
43     return HxVec2Int(_values[0], _values[1]);
44 }
```

6.37.3.9 HxComplex::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
47 {
48     return HxVec3Int(int(_values[0]), int(_values[1]), 0);
49 }
```

6.37.3.10 HxComplex::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
52 {
53     return HxVec3Double(_values[0], _values[1], 0);
54 }
```

6.37.3.11 int HxComplex::operator== (const HxComplex & *v*) const [inline]

Equal.

```
393 {
394     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
395 }
```

6.37.3.12 int HxComplex::operator!= (const HxComplex & *v*) const [inline]

Not equal.

```
399 {
400     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
401 }
```

6.37.3.13 int HxComplex::operator< (const HxComplex & *v*) const [inline]

Less than.

```
405 {
406     return (fabs(_values[0]) + fabs(_values[1])) <
407            (fabs(v._values[0]) + fabs(v._values[1]));
408 }
```

6.37.3.14 int HxComplex::operator<= (const HxComplex & *v*) const [inline]

Less equal.

```
412 {
413     return (fabs(_values[0]) + fabs(_values[1])) <=
414            (fabs(v._values[0]) + fabs(v._values[1]));
415 }
```

6.37.3.15 int HxComplex::operator> (const HxComplex & *v*) const [inline]

Greater than.

```
419 {
420     return (fabs(_values[0]) + fabs(_values[1])) >
421            (fabs(v._values[0]) + fabs(v._values[1]));
422 }
```

6.37.3.16 int HxComplex::operator>= (const HxComplex & *v*) const [inline]

Greater equal.

```
426 {
427     return (fabs(_values[0]) + fabs(_values[1])) >=
428            (fabs(v._values[0]) + fabs(v._values[1]));
429 }
```

6.37.3.17 HxComplex HxComplex::operator- () const [inline]

Negation.

```
433 {
434     return HxComplex(-_values[0], -_values[1]);
435 }
```

6.37.3.18 HxComplex HxComplex::complement () const [inline]

Complement.

```
439 {
440     return HxComplex(-_values[0], -_values[1]);
441 }
```

6.37.3.19 HxComplex HxComplex::conjugate () const [inline]

Conjugate.

```
445 {
446     return HxComplex(_values[0], -_values[1]);
447 }
```

6.37.3.20 HxScalarDouble HxComplex::abs () const [inline]

Absolute value.

```
451 {
452     return HxScalarDouble(::sqrt(_values[0]*_values[0]+_values[1]*_values[1]));
453 }
```

6.37.3.21 HxScalarDouble HxComplex::arg () const [inline]

Argument.

```
457 {
458     return HxScalarDouble(::atan2(_values[1], _values[0]));
459 }
```

6.37.3.22 HxComplex HxComplex::ceil () const [inline]

Ceiling.

```
463 {
464     return HxComplex(::ceil(_values[0]), ::ceil(_values[1]));
465 }
```

6.37.3.23 HxComplex HxComplex::floor () const [inline]

Floor.

```
469 {
470     return HxComplex(::floor(_values[0]), ::floor(_values[1]));
471 }
```

6.37.3.24 HxComplex HxComplex::round () const [inline]

Round.

```
475 {
476     return HxComplex((int) (_values[0] + ((_values[0] >= 0) ? 0.5 : -0.5)),
477                      (int) (_values[1] + ((_values[1] >= 0) ? 0.5 : -0.5)));
478 }
```

6.37.3.25 HxComplex HxComplex::sum () const [inline]

Sum.

```
798 {
799     return *this;
800 }
```

6.37.3.26 HxComplex HxComplex::product () const [inline]

Product.

```
804 {
805     return *this;
806 }
```

6.37.3.27 HxScalarDouble HxComplex::min () const [inline]

Minimum.

```
810 {
811     return (_values[0] < _values[1]) ? _values[0] : _values[1];
812 }
```

6.37.3.28 HxScalarDouble HxComplex::max () const [inline]

Maximum.

```
816 {
817     return (_values[0] > _values[1]) ? _values[0] : _values[1];
818 }
```

6.37.3.29 HxScalarDouble HxComplex::norm1 () const

L1 norm.

```
58 {
59     return fabs(_values[0]) + fabs(_values[1]);
60 }
```

6.37.3.30 HxScalarDouble HxComplex::norm2 () const

L2 norm.

```
64 {
65     return ::sqrt(_values[0]*_values[0] + _values[1]*_values[1]);
66 }
```

6.37.3.31 HxScalarDouble HxComplex::normInf () const

L infinity norm.

```
70 {
71     return (fabs(_values[0]) > fabs(_values[1])) ? fabs(_values[0]) :
72                                                    fabs(_values[1]);
73 }
```

6.37.3.32 HxComplex HxComplex::sqrt () const [inline]

Square root.

```
482 {
483     double a = _values[0];
484     double b = _values[1];
485     double sq = a*a+b*b;
486     double arg = ::atan(b/a)*0.5;
487     double mul = ::pow(sq, 0.25)*::exp(a*0.5);
488 
489     return HxComplex(mul*::cos(arg), mul*::sin(arg));
490 }
```

6.37.3.33 HxComplex HxComplex::sin () const [inline]

Sine.

```
494 {
495     return HxComplex(::sin(_values[0])*::cosh(_values[1]),
496         ::cos(_values[0])*::sinh(_values[1]));
497 }
```

6.37.3.34 HxComplex HxComplex::cos () const [inline]

Cosine.

```
501 {
502     return HxComplex(::cos(_values[0])*::cosh(_values[1]),
503         -::sin(_values[0])*::sinh(_values[1]));
504 }
```

6.37.3.35 HxComplex HxComplex::tan () const [inline]

Tangent.

```
508 {
509     double den = ::cos(2*_values[0])+::cosh(2*_values[1]);
510
511     return HxComplex(::sin(2*_values[0])/den, ::sinh(2*_values[1])/den);
512 }
```

6.37.3.36 HxComplex HxComplex::asin () const [inline]

Arc sine.

```
516 {
517     double a = _values[0];
518     double b = _values[1];
519
520     HxComplex c = HxComplex(1-a*a+b*b, 2*a*b).sqrt() + HxComplex(-b,a);
521
522     return HxComplex(c.arg().x(), -::log(c.abs().x()));
523 }
```

6.37.3.37 HxComplex HxComplex::acos () const [inline]

Arc cosine.

```
527 {
528     double a = _values[0];
529     double b = _values[1];
530
531     HxComplex c = HxComplex(1-a*a+b*b, 2*a*b).sqrt() + HxComplex(-b,a);
532
533     return HxComplex(M_PI/2.0 - c.arg().x(), ::log(c.abs().x()));
534 }
```

6.37.3.38 HxComplex HxComplex::atan () const [inline]

Arc tangent.

```
538 {
539     double a = _values[0];
540     double b = _values[1];
541
542     HxComplex cp = HxComplex(1-b,a);
543     HxComplex cm = HxComplex(1+b,-a);
544
545     return HxComplex(0.5*(cp.arg().x()-cm.arg().x()),
546         0.5*(::log(cm.abs().x())-::log(cp.abs().x())));
547 }
```

6.37.3.39 HxComplex HxComplex::atan2 () const

Arc tangent.

```
77 {
78     double a = _values[0];
79     double b = _values[1];
80 
81     HxComplex cp = HxComplex(1-b,a);
82     HxComplex cm = HxComplex(1+b,-a);
83 
84     return HxComplex(0.5*(cp.arg().x()-cm.arg().x()),
85         0.5*(::log(cm.abs().x())-::log(cp.abs().x())));
86 }
```

6.37.3.40 HxComplex HxComplex::sinh () const [inline]

Hyperbolic sine.

```
551 {
552     return HxComplex(::sinh(_values[0])*::cos(_values[1]),
553         ::cosh(_values[0])*::sin(_values[1]));
554 }
```

6.37.3.41 HxComplex HxComplex::cosh () const [inline]

Hyperbolic cosine.

```
558 {
559     return HxComplex(::cosh(_values[0])*::cos(_values[1]),
560         ::sinh(_values[0])*::sin(_values[1]));
561 }
```

6.37.3.42 HxComplex HxComplex::tanh () const [inline]

Hyperbolic tangent.

```
565 {
566     double den = ::cosh(2*_values[0])+::cos(2*_values[1]);
567 
568     return HxComplex(::sinh(2*_values[0])/den, ::sin(2*_values[1])/den);
569 }
```

6.37.3.43 HxComplex HxComplex::exp () const [inline]

Exponent.

```
573 {
574     double ea = ::exp(_values[0]);
575     return HxComplex(::cos(_values[1])*ea, ::sin(_values[1])*ea);
576 }
```

6.37.3.44 HxComplex HxComplex::log () const [inline]

Natural logarithm.

```
580 {
581     double re = _values[0];
582     double im = _values[1];
583     double mag = ::sqrt(re*re+im*im);
584 
585     return HxComplex(::log(mag), ::atan(re/im));
586 }
```

6.37.3.45 HxComplex HxComplex::log10 () const [inline]

Base 10 logarithm.

```
592 {
593     double re = _values[0];
594     double im = _values[1];
595     double mag = ::sqrt(re*re+im*im);
596 
597     return HxComplex(::log(mag)*theLog10, ::atan(re/im)*theLog10);
598 }
```

6.37.3.46 HxComplex & HxComplex::operator+= (const HxComplex & *v*) [inline]

Addition and assignment.

```
602 {
603     _values[0] += v._values[0];
604     _values[1] += v._values[1];
605     return *this;
606 }
```

6.37.3.47 HxComplex & HxComplex::operator-= (const HxComplex & *v*) [inline]

Subtraction and assignment.

```
610 {
611     _values[0] -= v._values[0];
612     _values[1] -= v._values[1];
613     return *this;
614 }
```

6.37.3.48 HxComplex & HxComplex::operator *= (const HxComplex & *v*) [inline]

Multiplication and assignment.

```
618 {
619     double re = _values[0]*v._values[0] - _values[1]*v._values[1];
620     double im = _values[0]*v._values[1] + _values[1]*v._values[0];
621     _values[0] = re;
622     _values[1] = im;
623     return *this;
624 }
```

6.37.3.49 HxComplex & HxComplex::operator/= (const HxComplex & *v*) [inline]

Division and assignment.

```
628 {
629     double re = v._values[0];
630     double im = v._values[1];
631     double sq = re*re+im*im;
632 
633     double mulre = _values[0]*re + _values[1]*im;
634     double mulim = _values[1]*re - _values[0]*im;
635     _values[0] = mulre / sq;
636     _values[1] = mulim / sq;
637 
638     return *this;
639 }
```

6.37.3.50 HxComplex HxComplex::min (const HxComplex & *v*) const [inline]

Minimum.

```
679 {
680     return (operator<(v)) ? (*this) : v;
681 }
```

6.37.3.51 HxComplex & HxComplex::minAssign (const HxComplex & *v*) [inline]

Minimum and assignment.

```
685 {
686     if (operator<(v))
687         return *this;
688     operator=(v);
689     return *this;
690 }
```

6.37.3.52 HxComplex HxComplex::max (const HxComplex & *v*) const [inline]

Maximum.

```
694 {
695     return (operator>(v)) ? (*this) : v;
696 }
```

6.37.3.53 HxComplex & HxComplex::maxAssign (const HxComplex & *v*) [inline]

Maximum and assignment.

```
700 {
701     if (operator>(v))
702         return *this;
703     operator=(v);
704     return *this;
705 }
```

6.37.3.54 HxComplex HxComplex::inf (const HxComplex & *v*) const [inline]

Infimum.

```
709 {
710     return HxComplex((_values[0] < v._values[0]) ? _values[0] : v._values[0],
711                      (_values[1] < v._values[1]) ? _values[1] : v._values[1]);
712 }
```

6.37.3.55 HxComplex & HxComplex::infAssign (const HxComplex & *v*) [inline]

Infimum and assignment.

```
716 {
717     _values[0] = (_values[0] < v._values[0]) ? _values[0] : v._values[0];
718     _values[1] = (_values[1] < v._values[1]) ? _values[1] : v._values[1];
719     return *this;
720 }
```

6.37.3.56 HxComplex HxComplex::sup (const HxComplex & *v*) const [inline]

Supremum.

```
724 {
725     return HxComplex((_values[0] > v._values[0]) ? _values[0] : v._values[0],
726                      (_values[1] > v._values[1]) ? _values[1] : v._values[1]);
727 }
```

6.37.3.57 HxComplex & HxComplex::supAssign (const HxComplex & *v*) [inline]

Supremum and assignment.

```
731 {
732     _values[0] = (_values[0] > v._values[0]) ? _values[0] : v._values[0];
733     _values[1] = (_values[1] > v._values[1]) ? _values[1] : v._values[1];
734     return *this;
735 }
```

6.37.3.58 HxComplex HxComplex::pow (const HxComplex & *v*) const [inline]

Power.

```
739 {
740     if (v._values[1] == 0) {
741         double a = _values[0];
742         double b = _values[1];
743         double c = v._values[0];
744         double sq = a*a+b*b;
745         double arg = ::atan(b/a)*c;
746         double mul = ::pow(sq, c/2)*::exp(a*c);
747
```

```
748         return HxComplex(mul*::cos(arg), mul*::sin(arg));
749     }
750
751     return (v * (*this).log()).exp();
752 }
```

6.37.3.59 HxComplex HxComplex::mod (const HxComplex & *v*) const [inline]

Modulo.

```
756 {
757     return (*this);
758 }
```

6.37.3.60 HxComplex HxComplex::and (const HxComplex & *v*) const [inline]

And.

```
762 {
763     return (*this);
764 }
```

6.37.3.61 HxComplex HxComplex::or (const HxComplex & *v*) const [inline]

Or.

```
768 {
769     return (*this);
770 }
```

6.37.3.62 HxComplex HxComplex::xor (const HxComplex & *v*) const [inline]

Xor.

```
774 {
775     return (*this);
776 }
```

6.37.3.63 HxComplex HxComplex::leftShift (const HxComplex & *v*) const [inline]

Left shift.

```
780 {
781     return (*this);
782 }
```

6.37.3.64 HxComplex HxComplex::rightShift (const HxComplex & *v*) const [inline]

Right shift.

```
786 {
787     return (*this);
788 }
```

6.37.3.65 HxScalarDouble HxComplex::dot (const HxComplex & *v*) const

Dot product.

```
90 {
91     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
92 }
```

6.37.3.66 HxComplex HxComplex::cross (const HxComplex & *v*) const [inline]

Cross product.

```
792 {
793     return HxComplex(0, 0);
794 }
```

6.37.3.67 STD_OSTREAM & HxComplex::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
96 {
97     return os << _values[0] << (_values[1]>=0 ? "+":"") << _values[1] << "i";
98 }
```

6.37.3.68 HxString HxComplex::toString () const

Value as a string.

```
101                           {
102     return makeString(_values[0]) + (_values[1]>=0 ? "+" : "")
103             + makeString(_values[1]) + "i";
104 }
```

6.37.4 Friends And Related Function Documentation

6.37.4.1 HxComplex operator+ (const HxComplex & *v1*, const HxComplex & *v2*) [friend]

Addition.

```
643 {
644     return HxComplex(v1._values[0] + v2._values[0],
645                      v1._values[1] + v2._values[1]);
646 }
```

6.37.4.2 HxComplex operator- (const HxComplex & *v1*, const HxComplex & *v2*) [friend]

Subtraction.

```
650 {
651     return HxComplex(v1._values[0] - v2._values[0],
652                      v1._values[1] - v2._values[1]);
653 }
```

6.37.4.3 HxComplex operator ∗ (const HxComplex & *v1*, const HxComplex & *v2*) [friend]

Multiplication.

```
657 {
658     double re = v1._values[0]*v2._values[0] - v1._values[1]*v2._values[1];
659     double im = v1._values[0]*v2._values[1] + v1._values[1]*v2._values[0];
660
661     return HxComplex(re, im);
662 }
```

6.37.4.4 HxComplex operator/ (const HxComplex & *v1*, const HxComplex & *v2*) [friend]

Division.

```
666 {
667     double re = v2._values[0];
668     double im = v2._values[1];
669     double sq = re*re+im*im;
670
671     double mulre = v1._values[0]*re + v1._values[1]*im;
672     double mulim = v1._values[1]*re - v1._values[0]*im;
673
674     return HxComplex(mulre / sq, mulim / sq);
675 }
```

6.37.5 Member Data Documentation

6.37.5.1 const HxComplex HxComplex::SMALL_VAL = HxComplex(0, 0) [static]

A small value w.r.t to the comparison operators ”<” and ”>”.

Not actually the minimum to avoid overflow.

6.37.5.2 const HxComplex HxComplex::LARGE_VAL = HxComplex(1e300, 1e300) [static]

A large value w.r.t to the comparison operators ”<” and ”>”.

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxComplex.h**
- HxComplex.c

6.38 HxHistogram Class Reference

Class definition of histogram.

`#include <HxHistogram.h>`

Public Methods

Constructors

- **HxHistogram** ()
  Construct an empty histogram.
- **HxHistogram** (const HxHistogram &)
  Copy constructor.
- **HxHistogram** (int dimSize)
  Construct a 1D histogram with a range from 0 to dimSize - 1 and a binWidth of 1.
- **HxHistogram** (int dimSize1, int dimSize2)
  Construct a 2D histogram with a range from 0 to dimSize - 1 and a binWidth of 1 in each dimension.
- **HxHistogram** (int dimSize1, int dimSize2, int dimSize3)
  Construct a 3D histogram with a range from 0 to dimSize - 1 and a binWidth of 1 in each dimension.
- **HxHistogram** (**HxValueType** dataType, int dimensions, int dimSize1, int dimSize2=0, int dimSize3=0)
  Construct Histogram with given data type and number of dimensions.
- **HxHistogram** (**HxValueType** dataType, int dimensions, double lowBin1, double highBin1, int nBins1, double lowBin2, double highBin2, int nBins2, double lowBin3, double highBin3, int nBins3)
  Construct a Histogram with given parameters.
- **HxHistogram** (**HxString** filename)
  Read a Histogram from disk.

Destructor

- ~**HxHistogram** ()
  Destructor.

Operators

- HxHistogram & **operator=** (const HxHistogram &)
  Assignment operator.
- int **isNull** () const
  Indicates wether this is a valid histogram.
- **operator int** () const
  Indicates wether this is a valid histogram.

Inquiry

- int **ident** () const
  The unique identifier of the histogram.
- **HxValueType dataType** () const
  The data value type.
- int **dimensionality** () const
  The number of dimensions of the histogram.
- int **dimensionSize** (int dim) const
  The size of the histogram in the i-th dimension.
- int **nrOfBins** () const
  The total size of the histogram.
- double **lowBin** (int dim) const
  The lowest bin in the i-th dimension.
- double **highBin** (int dim) const
  The highest bin in the i-th dimension.
- double **binWidth** (int dim) const
  The bin width in the i-th dimension.
- double **binToValue** (int bin, int dimension) const
  Translate bin to value.
- int **valueToBin** (double value, int dimension) const
  Translate value to bin.
- double **get** (int bin1) const
  Get the number of elements in the given bin.
- double **get** (int bin1, int bin2) const
  Get the number of elements in the given bin.
- double **get** (int bin1, int bin2, int bin3) const
  Get the number of elements in the given bin.

Checked modification

- void **insertValChecked** (int val)
  Insert value in 1D histogram.
- void **insertValChecked** (double val)
  Insert value in 1D histogram.
- void **insertValChecked** (**HxScalarInt** val)
  Insert value in 1D histogram.

- void **insertValChecked** (**HxScalarDouble** val)
  Insert value in 1D histogram.
- void **insertValChecked** (**HxVec2Int** val)
  Insert value in 2D histogram.
- void **insertValChecked** (**HxVec2Double** val)
  Insert value in 2D histogram.
- void **insertValChecked** (**HxVec3Int** val)
  Insert value in 3D histogram.
- void **insertValChecked** (**HxVec3Double** val)
  Insert value in 3D histogram.
- void **incBinChecked** (int bin)
  Increment the given bin.
- void **incBinChecked** (int bin1, int bin2)
  Increment the given bin.
- void **incBinChecked** (int bin1, int bin2, int bin3)
  Increment the given bin.

Unchecked modification

- void **insertVal** (int val)
  Insert value in 1D histogram.
- void **insertVal** (double val)
  Insert value in 1D histogram.
- void **insertVal** (double val, double sigma)
  Kernel Density Estimator.
- void **insertVal** (**HxScalarInt** val)
  Insert value in 1D histogram.
- void **insertVal** (**HxScalarDouble** val)
  Insert value in 1D histogram.
- void **insertVal** (**HxVec2Int** val)
  Insert value in 2D histogram.
- void **insertVal** (**HxVec2Double** val)
  Insert value in 2D histogram.
- void **insertVal** (**HxVec3Int** val)
  Insert value in 3D histogram.
- void **insertVal** (**HxVec3Double** val)
  Insert value in 3D histogram.
- void **incBin** (int bin)

Increment the given bin (assumes 1D histogram).

- void **incBin** (int bin1, int bin2)
  Increment the given bin (assumes 2D histogram).
- void **incBin** (int bin1, int bin2, int bin3)
  Increment the given bin (assumes 3D histogram).
- void **setBin** (int bin1, long val)
  Reset the value of the given bin to val (assumes 1D histogram).
- void **setBin** (int bin1, int bin2, long val)
  Reset the value of the given bin to val (assumes 2D histogram).
- void **setBin** (int bin1, int bin2, int bin3, long val)
  Reset the value of the given bin to val (assumes 3D histogram).
- void **setBin** (int bin1, double val)
  Reset the value of the given bin to val (assumes 1D histogram).
- void **setBin** (int bin1, int bin2, double val)
  Reset the value of the given bin to val (assumes 2D histogram).
- void **setBin** (int bin1, int bin2, int bin3, double val)
  Reset the value of the given bin to val (assumes 3D histogram).

Statistics

- HxHistogram **smooth** (double sigma=3.0)
  Smooth histogram data.
- std::list< **HxVec2Double** > **modes** ()
  Get histogram modes.
- HxHistogram **normalize** (double weight=1.0)
  Normalize histogram data.
- double **sum** () const
  The sum of the histogram.
- double **minVal** () const
  The minimum number of elements in the histogram.
- double **minVal** (int *index) const
  The minimum number of elements in the histogram.
- double **maxVal** () const
  The maximum number of elements in the histogram.
- double **maxVal** (int *index) const
  The maximum number of elements in the histogram.
- double **intersection** (const HxHistogram &) const
  Intersection of histograms.

- double **chiSquare** (const HxHistogram &) const

  Chi square intersection of histograms.

- double **chiSquareNorm** (const HxHistogram &) const

  Normalized chi square intersection of histograms.

Output/display

- HxHistogram **convert** (**HxValueType** dataType)

  convert dataType.

- void **getDataDouble** (double *data)

  Fill the externally allocated buffer with data from this histogram.

- void **getDataInt** (int *data)

  Fill the externally allocated buffer with data from this histogram.

- void **render3d** (int *data, int dataWidth, int dataHeight, double elevation, double alpha, double threshold)

  Assume the histogram is a 3d rgb cube and render it for display in Java in the externally allocated data buffer.

- STD_OSTREAM & **put** (STD_OSTREAM &, **HxString** delimit="") const

  Print histogram data in the given stream.

- int **write** (**HxString** filename)

  Write histogram to disk.

Reduce operations

- HxHistogram **threshold** (double valThreshold)

  Returns new histogram in which bins with count<=valThreshold are set to 0.0, bins with count>valThreshold keep original value.

- int **countBins** (double valThreshold=0.0)

  Returns number of bins with count>valThreshold.

- HxHistogram **reduceRange** (int binMin1, int binMax1=-1, int binMin2=0, int binMax2=-1, int binMin3=0, int binMax3=-1)

  Returns new histogram containing given range of bins only.

- HxHistogram **reduceRangeVal** (double binValMin1, double binValMax1, double binValMin2=0, double binValMax2=0, double binValMin3=0, double binValMax3=0)

  Returns new histogram containing given range of values (mapped to bin numbers) only.

- HxHistogram **to1D** (int dim=1)

  Projects n-dimensional (1<n<=3) histogram on a 1-D histogram for given dimension.

6.38.1 Detailed Description

Class definition of histogram.

The current implementation supports 1, 2, and 3 dimensional histograms with real-valued data (no integer data yet). For each dimension a range (lowB - highB) and a number of bins has to be specified (typically highB - lowB + 1 to get a binWidth of 1). The number of bins in a given dimension is known as the dimensionSize. The width of a bin is defined as (highB - lowB) / (nBins - 1). The first bin goes from lowB to lowB + binWidth, the last bin from highB to highB + binWidth.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 HxHistogram::HxHistogram ()

Construct an empty histogram.

```
67                          : _data(0)
68 {
69 }
```

6.38.2.2 HxHistogram::HxHistogram (const HxHistogram & *rhs*)

Copy constructor.

```
71                                                  : _data(rhs._data)
72 {
73 }
```

6.38.2.3 HxHistogram::HxHistogram (int *dimSize*)

Construct a 1D histogram with a range from 0 to dimSize - 1 and a binWidth of 1.

```
76 {
77     _data = new HxHistogramData(REAL_VALUE, 1,
78                                 0, dimSize - 1, dimSize,
79                                 0, 0, 0,
80                                 0, 0, 0);
81 }
```

6.38.2.4 HxHistogram::HxHistogram (int *dimSize1*, int *dimSize2*)

Construct a 2D histogram with a range from 0 to dimSize - 1 and a binWidth of 1 in each dimension.

```
84 {
85     _data = new HxHistogramData(REAL_VALUE, 2,
86                                 0, dimSize1 - 1, dimSize1,
87                                 0, dimSize2 - 1, dimSize2,
88                                 0, 0, 0);
89 }
```

6.38.2.5 HxHistogram::HxHistogram (int *dimSize1*, int *dimSize2*, int *dimSize3*)

Construct a 3D histogram with a range from 0 to dimSize - 1 and a binWidth of 1 in each dimension.

```
92 {
93     _data = new HxHistogramData(REAL_VALUE, 3,
94                                 0, dimSize1 - 1, dimSize1,
95                                 0, dimSize2 - 1, dimSize2,
96                                 0, dimSize3 - 1, dimSize3);
97 }
```

6.38.2.6 HxHistogram::HxHistogram (HxValueType *dataType*, int *dimensions*, int *dimSize1*, int *dimSize2* = 0, int *dimSize3* = 0)

Construct Histogram with given data type and number of dimensions.

The range in a dimension is 0 to dimSize - 1, the binWidth is 1.

```
101 {
102     _data = new HxHistogramData(dataType, dimensions,
103                                 0, dimSize1 - 1, dimSize1,
104                                 0, dimSize2 - 1, dimSize2,
105                                 0, dimSize3 - 1, dimSize3);
106 }
```

6.38.2.7 HxHistogram::HxHistogram (HxValueType *dataType*, int *dimensions*, double *lowBin1*, double *highBin1*, int *nBins1*, double *lowBin2*, double *highBin2*, int *nBins2*, double *lowBin3*, double *highBin3*, int *nBins3*)

Construct a Histogram with given parameters.

If dataType == INT_VALUE parameters are casted.

```
112 {
113     _data = new HxHistogramData(dataType, dimensions,
114                                 lowBin1, highBin1, nBins1,
115                                 lowBin2, highBin2, nBins2,
116                                 lowBin3, highBin3, nBins3);
117 }
```

6.38.2.8 HxHistogram::HxHistogram (HxString *filename*)

Read a Histogram from disk.

```
124 {
125     HxString name;
126     FILE *fp;
127     IOHEADER header;
128 
129     name = filename + ".hst";
130 
131     fp = fopen(name.c_str(), "rb");
132     if (!fp)
133         _data = 0;
```

```
134     else {
135         fread(&header, sizeof(header), 1, fp);
136 
137         _data = new HxHistogramData(header.dataType, header.dimensions,
138                                     header.lowBin1, header.highBin1, header.nBins1,
139                                     header.lowBin2, header.highBin2, header.nBins2,
140                                     header.lowBin3, header.highBin3, header.nBins3);
141 
142         if (_data->_dataType == REAL_VALUE)
143             fread(_data->_bins.realValue, _data->_totSize,
144                 sizeof(*(_data->_bins.realValue)), fp);
145         else
146             fread(_data->_bins.intValue, _data->_totSize,
147                 sizeof(*(_data->_bins.intValue)), fp);
148         fclose(fp);
149     }
150 }
```

6.38.2.9 HxHistogram::~HxHistogram ()

Destructor.

```
153 {
154 }
```

6.38.3 Member Function Documentation

6.38.3.1 HxHistogram & HxHistogram::operator= (const HxHistogram & *rhs*)

Assignment operator.

```
165 {
166     _data = rhs._data;
167     return *this;
168 }
```

6.38.3.2 int HxHistogram::isNull () const

Indicates wether this is a valid histogram.

```
172 {
173     return _data ? 0 : 1;
174 }
```

6.38.3.3 HxHistogram::operator int () const

Indicates wether this is a valid histogram.

```
178 {
179     return _data ? 1 : 0;
180 }
```

6.38.3.4 int HxHistogram::ident () const

The unique identifier of the histogram.

```
184 {
185     return _data ? _data->_id : 0 ;
186 }
```

6.38.3.5 HxValueType HxHistogram::dataType () const

The data value type.

Either INT_VALUE or REAL_VALUE.

```
190 {
191     return _data ? _data->_dataType : INT_VALUE;
192 }
```

6.38.3.6 int HxHistogram::dimensionality () const

The number of dimensions of the histogram.

```
196 {
197     return _data ? _data->_nDims : 0;
198 }
```

6.38.3.7 int HxHistogram::dimensionSize (int *dim*) const

The size of the histogram in the i-th dimension.

The first dimension has i = 1.

```
202 {
203     if (!_data)
204         return 0;
205     switch (dim) {
206     case 1: return _data->_dimSize1;
207     case 2: return _data->_dimSize2;
208     case 3: return _data->_dimSize3;
209     }
210     return 0;
211 }
```

6.38.3.8 int HxHistogram::nrOfBins () const

The total size of the histogram.

```
215 {
216     if (!_data)
217         return 0;
218     return _data->_totSize;
219 }
```

6.38.3.9 double HxHistogram::lowBin (int *dim*) const

The lowest bin in the i-th dimension.

```
223 {
224     if (!_data)
225         return 0;
226     switch (dim) {
227     case 1: return _data->_lowBin1;
228     case 2: return _data->_lowBin2;
229     case 3: return _data->_lowBin3;
230     }
231     return 0;
232 }
```

6.38.3.10 double HxHistogram::highBin (int *dim*) const

The highest bin in the i-th dimension.

```
236 {
237     if (!_data)
238         return 0;
239     switch (dim) {
240     case 1: return _data->_highBin1;
241     case 2: return _data->_highBin2;
242     case 3: return _data->_highBin3;
243     }
244     return 0;
245 }
```

6.38.3.11 double HxHistogram::binWidth (int *dim*) const

The bin width in the i-th dimension.

```
249 {
250     if (!_data)
251         return 0;
252     switch (dim) {
253     case 1: return _data->_binWidth1;
254     case 2: return _data->_binWidth2;
255     case 3: return _data->_binWidth3;
256     }
257     return 0;
258 }
```

6.38.3.12 double HxHistogram::binToValue (int *bin*, int *dimension*) const

Translate bin to value.

```
262 {
263     if (!_data)
264         return 0;
265     switch (dimension) {
266     case 1: return bin * _data->_binWidth1 + _data->_lowBin1;
```

```
267     case 2: return bin * _data->_binWidth2 + _data->_lowBin2;
268     case 3: return bin * _data->_binWidth3 + _data->_lowBin3;
269     }
270     return 0;
271 }
```

6.38.3.13 int HxHistogram::valueToBin (double *val*, int *dimension*) const

Translate value to bin.

```
275 {
276     if (!_data)
277         return -1;
278     switch (dimension) {
279     case 1: return (int) (((val - _data->_lowBin1) / _data->_binWidth1) + 0.5);
280     case 2: return (int) (((val - _data->_lowBin2) / _data->_binWidth2) + 0.5);
281     case 3: return (int) (((val - _data->_lowBin3) / _data->_binWidth3) + 0.5);
282     }
283     return -1;
284 }
```

6.38.3.14 double HxHistogram::get (int *bin1*) const [inline]

Get the number of elements in the given bin.

```
428 {
429     return _data ? _data->getBin(bin1) : 0;
430 }
```

6.38.3.15 double HxHistogram::get (int *bin1*, int *bin2*) const [inline]

Get the number of elements in the given bin.

```
434 {
435     return _data ? _data->getBin(bin2 * _data->_dimSize1 + bin1) : 0;
436 }
```

6.38.3.16 double HxHistogram::get (int *bin1*, int *bin2*, int *bin3*) const [inline]

Get the number of elements in the given bin.

```
440 {
441     return _data ? _data->getBin((bin3 * _data->_dimSize2 + bin2) *
442                                  _data->_dimSize1 + bin1) : 0;
443 }
```

6.38.3.17 void HxHistogram::insertValChecked (int *val*)

Insert value in 1D histogram.

```
288 {
289     if (_data && (_data->_nDims == 1))
290         incBinChecked(
291             (int) (((val - _data->_lowBin1) / _data->_binWidth1) + 0.5));
292 }
```

6.38.3.18 void HxHistogram::insertValChecked (double *val*)

Insert value in 1D histogram.

```
296 {
297     if (_data && (_data->_nDims == 1))
298         incBinChecked(
299             (int) (((val - _data->_lowBin1) / _data->_binWidth1) + 0.5));
300 }
```

6.38.3.19 void HxHistogram::insertValChecked (HxScalarInt *val*)

Insert value in 1D histogram.

```
304 {
305     if (_data && (_data->_nDims == 1))
306         incBinChecked(
307             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5));
308 }
```

6.38.3.20 void HxHistogram::insertValChecked (HxScalarDouble *val*)

Insert value in 1D histogram.

```
312 {
313     if (_data && (_data->_nDims == 1))
314         incBinChecked(
315             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5));
316 }
```

6.38.3.21 void HxHistogram::insertValChecked (HxVec2Int *val*)

Insert value in 2D histogram.

```
320 {
321     if (_data && (_data->_nDims == 2))
322         incBinChecked(
323             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
324             (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5));
325 }
```

6.38.3.22 void HxHistogram::insertValChecked (HxVec2Double *val*)

Insert value in 2D histogram.

```
329 {
330     if (_data && (_data->_nDims == 2))
331         incBinChecked(
332             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
333             (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5));
334 }
```

6.38.3.23 void HxHistogram::insertValChecked (HxVec3Int *val*)

Insert value in 3D histogram.

```
338 {
339     if (_data && (_data->_nDims == 3))
340         incBinChecked(
341             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
342             (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5),
343             (int) (((val.z() - _data->_lowBin3) / _data->_binWidth3) + 0.5));
344 }
```

6.38.3.24 void HxHistogram::insertValChecked (HxVec3Double *val*)

Insert value in 3D histogram.

```
348 {
349     if (_data && (_data->_nDims == 3))
350         incBinChecked(
351             (int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
352             (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5),
353             (int) (((val.z() - _data->_lowBin3) / _data->_binWidth3) + 0.5));
354 }
```

6.38.3.25 void HxHistogram::incBinChecked (int *bin*)

Increment the given bin.

Checks whether this is a 1D histogram and preserves reference count. Values outside the histogram range are ignored.

```
358 {
359     if (_data) {
360         if ((_data->_nDims == 1) &&
361                 (bin >= 0) && (bin < _data->_dimSize1)) {
362             _data->getUnshared();
363             _data->incBin(bin);
364         }
365     }
366 }
```

6.38.3.26 void HxHistogram::incBinChecked (int *bin1*, int *bin2*)

Increment the given bin.

Checks whether this is a 2D histogram and preserves reference count. Values outside the histogram range are ignored.

```
370 {
371     if (_data) {
372         if ((_data->_nDims == 2) &&
373                 (bin1 >= 0) && (bin1 < _data->_dimSize1) &&
374                 (bin2 >= 0) && (bin2 < _data->_dimSize2)) {
375             _data->getUnshared();
376             _data->incBin(bin2 * _data->_dimSize1 + bin1);
377         }
378     }
379 }
```

6.38.3.27 void HxHistogram::incBinChecked (int *bin1*, int *bin2*, int *bin3*)

Increment the given bin.

Checks whether this is a 3D histogram and preserves reference count. Values outside the histogram range are ignored.

```
383 {
384     if (_data) {
385         if ((_data->_nDims == 3) &&
386                 (bin1 >= 0) && (bin1 < _data->_dimSize1) &&
387                 (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
388                 (bin3 >= 0) && (bin3 < _data->_dimSize3)) {
389             _data->getUnshared();
390             _data->incBin((bin3 * _data->_dimSize2 + bin2) *
391                           _data->_dimSize1 + bin1);
392         }
393     }
394 }
```

6.38.3.28 void HxHistogram::insertVal (int *val*) [inline]

Insert value in 1D histogram.

```
522 {
523     incBin((int) (((val - _data->_lowBin1) / _data->_binWidth1) + 0.5));
524 }
```

6.38.3.29 void HxHistogram::insertVal (double *val*) [inline]

Insert value in 1D histogram.

```
528 {
529     incBin((int) (((val - _data->_lowBin1) / _data->_binWidth1) + 0.5));
530 }
```

6.38.3.30 void HxHistogram::insertVal (double *val*, double *sigma*)

Kernel Density Estimator.

```
399 {
400     if (!_data || (_data->_dataType != REAL_VALUE))
401         return;
402 
403     double ori = (val - _data->_lowBin1) / _data->_binWidth1;
404     double s = sigma/_data->_binWidth1;
405     double t, sat;
406 
407     int binmin = (int) (ori-3*s+0.5);
408     int binmax = (int) (ori+3*s+0.5);
409     int bin;
410 
411     if (binmin < 0)
412         binmin = 0;
413 
414     /* gaussian mass left outside kernel */
415     t = (ori-binmin+0.5)/(sqrt(2.0)*s);
416 
417     sat = erfc(t)*0.5;
418 
419     _data->_bins.realValue[binmin ? binmin-1 : 0] += sat;
420 
421     if (binmax >= _data->_dimSize1)
422         binmax = _data->_dimSize1-1;
423 
424     /* gaussian mass right outside kernel */
425     t = (binmax-ori+0.5)/(sqrt(2.0)*s);
426 
427     sat = erfc(t)*0.5;
428 
429     _data->_bins.realValue[binmax < _data->_dimSize1-1 ? binmax+1 : binmax]
430         += sat;
431 
432     double a = 1./(sqrt(2*M_PI)*s);
433     double b = -0.5/(s*s);
434 
435     for (bin=binmin; bin<=binmax; bin++) {
436         double v = bin-ori;
437         _data->_bins.realValue[bin] += a*exp(b*v*v);
438     }
439 }
```

6.38.3.31 void HxHistogram::insertVal (HxScalarInt *val*) [inline]

Insert value in 1D histogram.

```
534 {
535     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5));
536 }
```

6.38.3.32 void HxHistogram::insertVal (HxScalarDouble *val*) [inline]

Insert value in 1D histogram.

```
540 {
541     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5));
542 }
```

6.38.3.33 void HxHistogram::insertVal (HxVec2Int *val*) [inline]

Insert value in 2D histogram.

```
546 {
547     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
548            (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5));
549 }
```

6.38.3.34 void HxHistogram::insertVal (HxVec2Double *val*) [inline]

Insert value in 2D histogram.

```
553 {
554     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
555            (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5));
556 }
```

6.38.3.35 void HxHistogram::insertVal (HxVec3Int *val*) [inline]

Insert value in 3D histogram.

```
560 {
561     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
562            (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5),
563            (int) (((val.z() - _data->_lowBin3) / _data->_binWidth3) + 0.5));
564 }
```

6.38.3.36 void HxHistogram::insertVal (HxVec3Double *val*) [inline]

Insert value in 3D histogram.

```
568 {
569     incBin((int) (((val.x() - _data->_lowBin1) / _data->_binWidth1) + 0.5),
570            (int) (((val.y() - _data->_lowBin2) / _data->_binWidth2) + 0.5),
571            (int) (((val.z() - _data->_lowBin3) / _data->_binWidth3) + 0.5));
572 }
```

6.38.3.37 void HxHistogram::incBin (int *bin*) [inline]

Increment the given bin (assumes 1D histogram).

Values outside the histogram range are ignored.

```
447 {
448     if ((bin >= 0) && (bin < _data->_dimSize1))
449         _data->incBin(bin);
450 }
```

6.38.3.38 void HxHistogram::incBin (int *bin1*, int *bin2*) [inline]

Increment the given bin (assumes 2D histogram).

Values outside the histogram range are ignored.

```
454 {
455     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
456             (bin2 >= 0) && (bin2 < _data->_dimSize2))
457         _data->incBin(bin2 * _data->_dimSize1 + bin1);
458 }
```

6.38.3.39 void HxHistogram::incBin (int *bin1*, int *bin2*, int *bin3*) [inline]

Increment the given bin (assumes 3D histogram).

Values outside the histogram range are ignored.

```
462 {
463     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
464             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
465             (bin3 >= 0) && (bin3 < _data->_dimSize3))
466         _data->incBin((bin3 * _data->_dimSize2 + bin2) *
467                       _data->_dimSize1 + bin1);
468 }
```

6.38.3.40 void HxHistogram::setBin (int *bin1*, long *val*) [inline]

Reset the value of the given bin to val (assumes 1D histogram).

Values outside the histogram range are ignored.

```
472 {
473     if ((bin1 >= 0) && (bin1 < _data->_dimSize1))
474         _data->setBin(bin1, val);
475 }
```

6.38.3.41 void HxHistogram::setBin (int *bin1*, int *bin2*, long *val*) [inline]

Reset the value of the given bin to val (assumes 2D histogram).

Values outside the histogram range are ignored.

```
479 {
480     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
481             (bin2 >= 0) && (bin2 < _data->_dimSize2))
482         _data->setBin(bin2 * _data->_dimSize1 + bin1, val);
483 }
```

6.38.3.42 void HxHistogram::setBin (int *bin1*, int *bin2*, int *bin3*, long *val*) [inline]

Reset the value of the given bin to val (assumes 3D histogram).

Values outside the histogram range are ignored.

```
487 {
488     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
489             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
490             (bin3 >= 0) && (bin3 < _data->_dimSize3))
491         _data->setBin((bin3 * _data->_dimSize2 + bin2) *
492                       _data->_dimSize1 + bin1, val);
493 }
```

6.38.3.43 void HxHistogram::setBin (int *bin1*, double *val*) [inline]

Reset the value of the given bin to val (assumes 1D histogram).

Values outside the histogram range are ignored.

```
497 {
498     if ((bin1 >= 0) && (bin1 < _data->_dimSize1))
499         _data->setBin(bin1, val);
500 }
```

6.38.3.44 void HxHistogram::setBin (int *bin1*, int *bin2*, double *val*) [inline]

Reset the value of the given bin to val (assumes 2D histogram).

Values outside the histogram range are ignored.

```
504 {
505     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
506             (bin2 >= 0) && (bin2 < _data->_dimSize2))
507         _data->setBin(bin2 * _data->_dimSize1 + bin1, val);
508 }
```

6.38.3.45 void HxHistogram::setBin (int *bin1*, int *bin2*, int *bin3*, double *val*) [inline]

Reset the value of the given bin to val (assumes 3D histogram).

Values outside the histogram range are ignored.

```
512 {
513     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
514             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
515             (bin3 >= 0) && (bin3 < _data->_dimSize3))
516         _data->setBin((bin3 * _data->_dimSize2 + bin2) *
517                       _data->_dimSize1 + bin1, val);
518 }
```

6.38.3.46 HxHistogram HxHistogram::smooth (double *sigma* = 3.0)

Smooth histogram data.

```
443 {
444     if (!_data)
445         return HxHistogram();
446
```

```
447     GaussIIR g(sigma);
448     HxHistogramData *data = new HxHistogramData(REAL_VALUE, *_data);
449 
450     /* for now only works for 1D histogram */
451     /* will be fixed when image processing works on sampled density fields */
452     /* instead of HxImageRep only... */
453 
454 #ifndef _DEBUG
455     g.lineFilter(data->_bins.realValue, data->_totSize);
456 #endif
457 
458     return HxHistogram(data);
459 }
```

6.38.3.47 std::list< HxVec2Double > HxHistogram::modes ()

Get histogram modes.

```
501 {
502     HxHistogramData *data = (_data->_dataType != REAL_VALUE) ?
503         new HxHistogramData(REAL_VALUE, *_data) : _data.pointee();
504 
505 
506 
507     /* for now only works for 1D histogram */
508     /* will be fixed when image processing works on sampled density fields */
509     /* instead of HxImageRep only... */
510     /* then, see PAMI 22(11), pp. 1318--1323, 2000 */
511 
512     data->getUnshared();
513     std::list<MicrosoftListSortDoesntWork> l;
514 
515     /* maxima detection */
516     double *ptr = data->_bins.realValue;
517     double max = *ptr++;
518     int dir = 1;
519     for (int i=1; i < data->_totSize; i++) {
520         double val = *ptr++;
521         double grad = dir ? val-max : max-val;
522         if (grad <= 0) {
523             if (dir == 1) // maximum
524                 l.push_back(MicrosoftListSortDoesntWork(binToValue(i-1,1), max));
525             // else minimum
526             dir = dir ? 0 : 1;
527         }
528         max = val;
529     }
530 
531     if (data != _data.pointee())
532         delete data;
533 
534     l.sort();
535     std::list<HxVec2Double> res;
536     for (std::list<MicrosoftListSortDoesntWork>::iterator it = l.begin();
537             it != l.end(); it++)
538         res.push_back(HxVec2Double(*it));
539 
540     return res;
541 }
```

6.38.3.48 HxHistogram HxHistogram::normalize (double *weight* = 1.0)

Normalize histogram data.

```
545 {
546     if (!_data)
547         return HxHistogram();
548     HxHistogramData *data = new HxHistogramData(REAL_VALUE, *_data);
549 
550     int n = data->_totSize;
551     double norm = sum() / weight;
552     if (fabs(norm) > 0.0)
553     {
554         while (--n >= 0)
555             data->_bins.realValue[n] /= norm;
556         data->_sum = weight;
557     }
558 
559     return HxHistogram(data);
560 }
```

6.38.3.49 double HxHistogram::sum () const

The sum of the histogram.

```
564 {
565     if (!_data)
566         return 0;
567     int n = _data->_totSize;
568     double s = 0;
569     if (_data->_dataType == REAL_VALUE)
570         while (--n >= 0)
571             s += _data->_bins.realValue[n];
572     else
573         while (--n >= 0)
574             s += _data->_bins.intValue[n];
575     _data->_sum = s;
576     return s;
577 }
```

6.38.3.50 double HxHistogram::minVal () const

The minimum number of elements in the histogram.

```
581 {
582     if (!_data)
583         return 0;
584     int n = _data->_totSize;
585     double m;
586     if (_data->_dataType == REAL_VALUE) {
587         m = _data->_bins.realValue[0];
588         while (--n >= 0)
589             if (_data->_bins.realValue[n] < m)
590                 m = _data->_bins.realValue[n];
591     }
592     else {
593         m = _data->_bins.intValue[0];
594         while (--n >= 0)
```

```
595             if (_data->_bins.intValue[n] < m)
596                 m = _data->_bins.intValue[n];
597     }
598     return m;
599 }
```

6.38.3.51 double HxHistogram::minVal (int * *index*) const

The minimum number of elements in the histogram.

Index is the number of the bin at which the minimum number is first found.

```
625 {
626     if (!_data)
627         return 0;
628     int n = _data->_totSize;
629     double m;
630     *index = 0;
631     if (_data->_dataType == REAL_VALUE) {
632         m = _data->_bins.realValue[0];
633         while (--n >= 0)
634             if (_data->_bins.realValue[n] < m) {
635                 m = _data->_bins.realValue[n];
636                 *index = n;
637             }
638     }
639     else {
640         m = _data->_bins.intValue[0];
641         while (--n >= 0)
642             if (_data->_bins.intValue[n] < m) {
643                 m = _data->_bins.intValue[n];
644                 *index = n;
645             }
646     }
647     return m;
648 }
```

6.38.3.52 double HxHistogram::maxVal () const

The maximum number of elements in the histogram.

```
603 {
604     if (!_data)
605         return 0;
606     int n = _data->_totSize;
607     double m;
608     if (_data->_dataType == REAL_VALUE) {
609         m = _data->_bins.realValue[0];
610         while (--n >= 0)
611             if (_data->_bins.realValue[n] > m)
612                 m = _data->_bins.realValue[n];
613     }
614     else {
615         m = _data->_bins.intValue[0];
616         while (--n >= 0)
617             if (_data->_bins.intValue[n] > m)
618                 m = _data->_bins.intValue[n];
619     }
620     return m;
621 }
```

6.38.3.53 double HxHistogram::maxVal (int * *index*) const

The maximum number of elements in the histogram.

Index is the number of the bin at which the maximum number is first found.

```
652 {
653     if (!_data)
654         return 0;
655     int n = _data->_totSize;
656     double m;
657     *index = 0;
658     if (_data->_dataType == REAL_VALUE) {
659         m = _data->_bins.realValue[0];
660         while (--n >= 0)
661             if (_data->_bins.realValue[n] > m) {
662                 m = _data->_bins.realValue[n];
663                 *index = n;
664             }
665     }
666     else {
667         m = _data->_bins.intValue[0];
668         while (--n >= 0)
669             if (_data->_bins.intValue[n] > m) {
670                 m = _data->_bins.intValue[n];
671                 *index = n;
672             }
673     }
674     return m;
675 }
```

6.38.3.54 double HxHistogram::intersection (const HxHistogram & *rhs*) const

Intersection of histograms.

```
699 {
700     if (!_data)
701         return 0;
702     if (!isEqualSize(rhs))
703         return -1;
704     int n = _data->_totSize;
705     double s = 0;
706     if (_data->_dataType == REAL_VALUE)
707         if (rhs._data->_dataType == REAL_VALUE)
708             s = ::intersect(_data->_bins.realValue, rhs._data->_bins.realValue, n);
709         else
710             s = ::intersect(_data->_bins.realValue, rhs._data->_bins.intValue, n);
711     else
712         if (rhs._data->_dataType == REAL_VALUE)
713             s = ::intersect(_data->_bins.intValue, rhs._data->_bins.realValue, n);
714         else
715             s = ::intersect(_data->_bins.intValue, rhs._data->_bins.intValue, n);
716
717     return s;
718 }
```

6.38.3.55 double HxHistogram::chiSquare (const HxHistogram & *rhs*) const

Chi square intersection of histograms.

```
736 {
737     if (!_data)
738         return 0;
739     if (!isEqualSize(rhs))
740         return -1;
741     int n = _data->_totSize;
742     double s = 0;
743     if (_data->_dataType == REAL_VALUE)
744         if (rhs._data->_dataType == REAL_VALUE)
745             s = ::chiSq(_data->_bins.realValue, rhs._data->_bins.realValue, n);
746         else
747             s = ::chiSq(_data->_bins.realValue, rhs._data->_bins.intValue, n);
748     else
749         if (rhs._data->_dataType == REAL_VALUE)
750             s = ::chiSq(_data->_bins.intValue, rhs._data->_bins.realValue, n);
751         else
752             s = ::chiSq(_data->_bins.intValue, rhs._data->_bins.intValue, n);
753     return s;
754 }
```

6.38.3.56 double HxHistogram::chiSquareNorm (const HxHistogram & *rhs*) const

Normalized chi square intersection of histograms.

```
773 {
774     if (!_data)
775         return 0;
776     if (!isEqualSize(rhs))
777         return -1;
778     int n = _data->_totSize;
779     double s = 0;
780     if (_data->_dataType == REAL_VALUE)
781         if (rhs._data->_dataType == REAL_VALUE)
782             s = ::chiSqNorm(_data->_bins.realValue, rhs._data->_bins.realValue, n);
783         else
784             s = ::chiSqNorm(_data->_bins.realValue, rhs._data->_bins.intValue, n);
785     else
786         if (rhs._data->_dataType == REAL_VALUE)
787             s = ::chiSqNorm(_data->_bins.intValue, rhs._data->_bins.realValue, n);
788         else
789             s = ::chiSqNorm(_data->_bins.intValue, rhs._data->_bins.intValue, n);
790     return s;
791 }
```

6.38.3.57 HxHistogram HxHistogram::convert (HxValueType *dataType*)

convert dataType.

```
158 {
159     HxHistogramData *data = new HxHistogramData(*_data);
160     return HxHistogram(data);
161 }
```

6.38.3.58 void HxHistogram::getDataDouble (double * *data*)

Fill the externally allocated buffer with data from this histogram.

```
795 {
796     if (!_data)
797         return;
798     int n = _data->_totSize;
799     if (_data->_dataType == REAL_VALUE)
800         while (--n >= 0)
801             data[n] = _data->_bins.realValue[n];
802     else
803         while (--n >= 0)
804             data[n] = _data->_bins.intValue[n];
805 }
```

6.38.3.59 void HxHistogram::getDataInt (int * *data*)

Fill the externally allocated buffer with data from this histogram.

```
809 {
810     if (!_data)
811         return;
812     int n = _data->_totSize;
813     if (_data->_dataType == REAL_VALUE)
814         while (--n >= 0)
815             data[n] = (int)(_data->_bins.realValue[n] + 0.5);
816     else
817         while (--n >= 0)
818             data[n] = _data->_bins.intValue[n];
819 }
```

6.38.3.60 void HxHistogram::render3d (int * *data*, int *dataWidth*, int *dataHeight*, double *elevation*, double *alpha*, double *threshold*)

Assume the histogram is a 3d rgb cube and render it for display in Java in the externally allocated data buffer.

```
896 {
897     if (!_data)
898         return;
899     /*
900     el = (elevation * M_PI) / 180.;
901     al = (alpha * M_PI) / 180.;
902     cosa = cos(al);
903     sina = sin(al);
904     cose = cos(el);
905     sine = sin(el);
906     dataW = dataWidth;
907     xMin = 190; //for 3D histo
908     yMax = 190; //255;
909     */
910     double el = (elevation * M_PI) / 180.;
911     double al = (alpha * M_PI) / 180.;
912     double cosa = cos(al);
913     double sina = sin(al);
914     double cose = cos(el);
915     double sine = sin(el);
916     int dataW = dataWidth;
917     int xMin = 190; //for 3D histo
918     int yMax = 190; //255;
919     drawAxis(data,cosa,sina,cose,sine,dataW);
920
```

```
921     int n = 0;
922
923     if (_data->_dataType == REAL_VALUE) {
924         for (int b=0 ; b < _data->_dimSize3 ; b++) {
925             for (int g=0 ; g < _data->_dimSize2 ; g++) {
926                 for (int r=0 ; r < _data->_dimSize1 ; r++) {
927                     if (_data->_bins.realValue[n] > threshold) {
928                         /*
929                           Doen't work if hist range unequals 0..255
930                         double rV = r * _data->_binWidth1 + _data->_lowBin1;
931                         double gV = g * _data->_binWidth2 + _data->_lowBin2;
932                         double bV = b * _data->_binWidth3 + _data->_lowBin3;
933                         */
934                         double rV = r * 255./(_data->_dimSize1-1);
935                         double gV = g * 255./(_data->_dimSize2-1);
936                         double bV = b * 255./(_data->_dimSize3-1);
937                         setPixelV(data, transf3Dto2D(rV,gV,bV,cosa,sina,cose,sine), HxVec3Int(rV,gV,bV), c
938                     }
939                     n++;
940                 }
941             }
942         }
943     }
944     else {
945         for (int b=0 ; b < _data->_dimSize3 ; b++) {
946             for (int g=0 ; g < _data->_dimSize2 ; g++) {
947                 for (int r=0 ; r < _data->_dimSize1 ; r++) {
948                     if (_data->_bins.intValue[n] > threshold) {
949                         /*
950                           Doen't work if hist range unequals 0..255
951                         double rV = r * _data->_binWidth1 + _data->_lowBin1;
952                         double gV = g * _data->_binWidth2 + _data->_lowBin2;
953                         double bV = b * _data->_binWidth3 + _data->_lowBin3;
954                         */
955                         double rV = r * 255./(_data->_dimSize1-1);
956                         double gV = g * 255./(_data->_dimSize2-1);
957                         double bV = b * 255./(_data->_dimSize3-1);
958                         setPixelV(data, transf3Dto2D(rV,gV,bV,cosa,sina,cose,sine), HxVec3Int(rV,gV,bV), c
959                     }
960                     n++;
961                 }
962             }
963         }
964     }
965 }
```

6.38.3.61 STD_OSTREAM & HxHistogram::put (STD_OSTREAM & *os*, HxString *delimit* = " ") const

Print histogram data in the given stream.

```
969 {
970     if (!_data)
971         return os << "HxHistogram(null)" << STD_ENDL;
972     if (_data->_dataType == REAL_VALUE)
973         for (int dim3=0; dim3<dimensionSize(3) ; dim3++)
974             for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++)
975                 for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++)
976                     os << _data->_bins.realValue[(dim3 * _data->_dimSize2
977                         + dim2) * _data->_dimSize1 + dim1] << delimit;
978     else
979         for (int dim3=0; dim3<dimensionSize(3) ; dim3++)
```

```
980             for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++)
981                 for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++)
982                     os << _data->_bins.intValue[(dim3 * _data->_dimSize2
983                         + dim2) * _data->_dimSize1 + dim1] << delimit;
984     return os;
985 }
```

6.38.3.62 int HxHistogram::write (HxString *filename*)

Write histogram to disk.

```
1004 {
1005     HxString name;
1006     FILE *fp;
1007     IOHEADER header;
1008
1009     if (!_data)
1010         return 0;
1011
1012     name = filename + ".hst";
1013
1014     fp = fopen(name.c_str(), "wb");
1015     if (!fp)
1016         return 0;
1017
1018     header.dataType = _data->_dataType;
1019     header.dimensions = _data->_nDims;
1020     header.lowBin1 = _data->_lowBin1;
1021     header.highBin1 = _data->_highBin1;
1022     header.nBins1 = _data->_dimSize1;
1023     header.lowBin2 = _data->_lowBin2;
1024     header.highBin2 = _data->_highBin2;
1025     header.nBins2 = _data->_dimSize2;
1026     header.lowBin3 = _data->_lowBin3;
1027     header.highBin3 = _data->_highBin3;
1028     header.nBins3 = _data->_dimSize3;
1029     fwrite(&header, sizeof(header), 1, fp);
1030
1031     if (_data->_dataType == REAL_VALUE)
1032         fwrite(_data->_bins.realValue, _data->_totSize,
1033             sizeof(*(_data->_bins.realValue)), fp);
1034     else
1035         fwrite(_data->_bins.intValue, _data->_totSize,
1036             sizeof(*(_data->_bins.intValue)), fp);
1037
1038     fclose(fp);
1039
1040     return 1;
1041 }
```

6.38.3.63 HxHistogram HxHistogram::threshold (double *valThreshold*)

Returns new histogram in which bins with count<=valThreshold are set to 0.0, bins with count>valThreshold keep original value.

```
1045 {
1046     if (!_data)
1047         return HxHistogram();
```

```
1048     HxHistogramData *data = new HxHistogramData(*_data);
1049
1050     int n = data->_totSize;
1051
1052     if (data->_dataType == REAL_VALUE)
1053         while (--n >= 0) {
1054             if (data->_bins.realValue[n]<=valThreshold)
1055                 data->_bins.realValue[n]=0.0;
1056         }
1057     else
1058         while (--n >= 0) {
1059             if (data->_bins.intValue[n]<=valThreshold)
1060                 data->_bins.intValue[n]=0.0;
1061         }
1062
1063     return HxHistogram(data);
1064 }
```

6.38.3.64 int HxHistogram::countBins (double *valThreshold* = 0.0)

Returns number of bins with count>valThreshold.

Counts number of non-empty bins by default.

```
1068 {
1069     if (!_data)
1070         return 0;
1071
1072     int n = _data->_totSize;
1073
1074     int counter=0;
1075
1076     if (_data->_dataType == REAL_VALUE)
1077         while (--n >= 0) {
1078             if (_data->_bins.realValue[n]>valThreshold)
1079                counter++;
1080         }
1081     else
1082         while (--n >= 0) {
1083             if (_data->_bins.intValue[n]>valThreshold)
1084                counter++;
1085         }
1086
1087     return counter;
1088 }
```

6.38.3.65 HxHistogram HxHistogram::reduceRange (int *binMin1*, int *binMax1* = -1, int *binMin2* = 0, int *binMax2* = -1, int *binMin3* = 0, int *binMax3* = -1)

Returns new histogram containing given range of bins only.

(Including given min and max.) Default value -1 maps to **dimensionSize**() (p. 267)-1.

```
1094 {
1095     if (!_data)
1096         return HxHistogram();
1097
1098     // Default value binMax=-1 maps to dimensionSize()-1.
1099     if (binMax1<0)
```

```
1100         binMax1=dimensionSize(1)-1;
1101     if (binMax2<0)
1102         binMax2=dimensionSize(2)-1;
1103     if (binMax3<0)
1104         binMax3=dimensionSize(3)-1;
1105 
1106     if (binMin1<0 || binMin2<0 || binMin3<0)
1107     {
1108         STD_CERR << "Error in HxHistogram::reduceRange, minimum smaller than 0." << STD_ENDL;
1109         return HxHistogram();
1110     }
1111 
1112     if (binMax1<binMin1 || binMax2<binMin2 || binMax3<binMin3)
1113     {
1114         STD_CERR << "Error in HxHistogram::reduceRange, minimum higher than maximum." << STD_ENDL;
1115         return HxHistogram();
1116     }
1117 
1118     // Make new histogram for given range
1119     HxHistogram h=HxHistogram(dataType(),dimensionality(),
1120         binToValue(binMin1,1),binToValue(binMax1,1),binMax1-binMin1+1,
1121         binToValue(binMin2,2),binToValue(binMax2,2),binMax2-binMin2+1,
1122         binToValue(binMin3,3),binToValue(binMax3,3),binMax3-binMin3+1);
1123 
1124     if (_data->_dataType == REAL_VALUE) {
1125         for (int z=binMin3; z<=binMax3; z++)
1126             for (int y=binMin2; y<=binMax2; y++)
1127                 for (int x=binMin1; x<=binMax1; x++) {
1128                     int n = ((z-binMin3) * _data->_dimSize2 + (y-binMin2)) *
1129                          _data->_dimSize1 + x-binMin1;
1130                     int m = (z * _data->_dimSize2 + y) *
1131                          _data->_dimSize1 + x;
1132                     h._data->_bins.realValue[n] = _data->_bins.realValue[m];
1133                 }
1134     }
1135     else {
1136         for (int z=binMin3; z<=binMax3; z++)
1137             for (int y=binMin2; y<=binMax2; y++)
1138                 for (int x=binMin1; x<=binMax1; x++) {
1139                     int n = ((z-binMin3) * _data->_dimSize2 + (y-binMin2)) *
1140                          _data->_dimSize1 + x-binMin1;
1141                     int m = (z * _data->_dimSize2 + y) *
1142                          _data->_dimSize1 + x;
1143                     h._data->_bins.intValue[n] = _data->_bins.intValue[m];
1144                 }
1145     }
1146 
1147     return h;
1148 }
```

6.38.3.66 HxHistogram HxHistogram::reduceRangeVal (double *binValMin1*, double *binValMax1*, double *binValMin2* = 0, double *binValMax2* = 0, double *binValMin3* = 0, double *binValMax3* = 0)

Returns new histogram containing given range of values (mapped to bin numbers) only.

(Including bins for given min and max.) If min==max, the whole range for that dimension is returned.

```
1153 {
1154     if (binValMin1==binValMax1) {
1155         binValMin1=_data->_lowBin1;
1156         binValMax1=_data->_highBin1;
```

```
1157     }
1158     if (binValMin1<_data->_lowBin1)
1159         binValMin1=_data->_lowBin1;
1160     if (binValMax1>_data->_highBin1)
1161         binValMax1=_data->_highBin1;
1162     if (binValMin2==binValMax2) {
1163         binValMin2=_data->_lowBin2;
1164         binValMax2=_data->_highBin2;
1165     }
1166     if (binValMin2<_data->_lowBin2)
1167         binValMin2=_data->_lowBin2;
1168     if (binValMax2>_data->_highBin2)
1169         binValMax2=_data->_highBin2;
1170     if (binValMin3==binValMax3) {
1171         binValMin3=_data->_lowBin3;
1172         binValMax3=_data->_highBin3;
1173     }
1174     if (binValMin3<_data->_lowBin3)
1175         binValMin3=_data->_lowBin3;
1176     if (binValMax3>_data->_highBin3)
1177         binValMax3=_data->_highBin3;
1178     return reduceRange(valueToBin(binValMin1,1),valueToBin(binValMax1,1),
1179         valueToBin(binValMin2,2),valueToBin(binValMax2,2),
1180         valueToBin(binValMin3,3),valueToBin(binValMax3,3));
1181 }
```

6.38.3.67 HxHistogram HxHistogram::to1D (int *dim* = 1)

Projects n-dimensional (1<n<=3) histogram on a 1-D histogram for given dimension.

```
1186 {
1187     if (dimensionality()<2)
1188     {
1189 //      STD_CERR << "Warning. HxHistogram::to1D, dimensionality is " << dimensionality() << STD_ENDL;
1190         return *this;
1191     }
1192
1193     if (dim<1 || dim>3)
1194     {
1195         STD_CERR << "Error. Dimension out of bounds: " << dim << STD_ENDL;
1196         return HxHistogram();
1197     }
1198
1199
1200     HxHistogram result;
1201     result=HxHistogram(dataType(), 1,
1202                        lowBin(dim),highBin(dim),dimensionSize(dim),
1203                        0,0,0,
1204                        0,0,0);
1205
1206     int otherDim1=-1;
1207     int otherDim2=-1;
1208
1209     if (dim==1)
1210     {
1211         otherDim1=3;
1212         otherDim2=2;
1213     }
1214     if (dim==2)
1215     {
1216         otherDim1=3;
1217         otherDim2=1;
```

```
1218     }
1219     if (dim==3)
1220     {
1221         otherDim1=2;
1222         otherDim2=1;
1223     }
1224
1225     for (int b=0; b<dimensionSize(dim); b++)
1226     {
1227         double total=0;
1228         for (int sb=0; sb<dimensionSize(otherDim1); sb++)
1229             for (int tb=0; tb<dimensionSize(otherDim2); tb++)
1230             {
1231                 if (dim==1)
1232                     total+=get(b,tb,sb);
1233                 if (dim==2)
1234                     total+=get(tb,b,sb);
1235                 if (dim==3)
1236                     total+=get(tb,sb,b);
1237             }
1238         result.setBin(b, total);
1239     }
1240
1241     return result;
1242 }
```

The documentation for this class was generated from the following files:

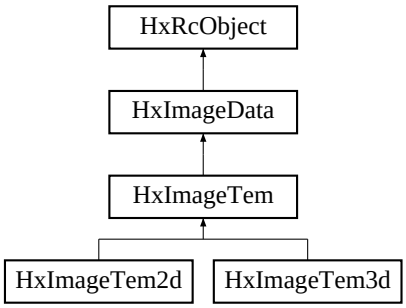
- **HxHistogram.h**
- HxHistogram.c

6.39 HxImageData Class Reference

The HxImageData class is the root in the cpp image hierarchy.

`#include <HxImageData.h>`

Inheritance diagram for HxImageData::



Constructors and destructor

- **HxImageData** ()

  Constructor.

- **HxImageData** (const HxImageData &)

  Copy constructor.

- virtual ~**HxImageData** ()

  Destructor.

Inquiry

- int **ident** () const
- **HxString name** () const
- void **name** (**HxString** s)
- virtual int **dimensionality** () const=0
- virtual int **dimensionSize** (int i) const=0
- virtual **HxSizes sizes** () const=0
- virtual int **numberOfPixels** () const=0
- virtual int **pixelDimensionality** () const=0
- virtual **HxValueType pixelType** () const=0
- virtual int **pixelPrecision** () const=0
- virtual **HxImageSignature signature** () const=0

Import/export operations

- virtual void **import** (HxByte *data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp="importPix") const
- virtual void **import** (short *data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp="importPix") const
- virtual void **import** (int *data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp="importPix") const
- virtual void **import** (float *data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp="importPix") const
- virtual void **import** (double *data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp="importPix") const
- virtual void **exportOp** (HxByte *data, **HxTagList** &tags, **HxString** exportOp="exportPix") const
- virtual void **exportOp** (short *data, **HxTagList** &tags, **HxString** exportOp="exportPix") const
- virtual void **exportOp** (int *data, **HxTagList** &tags, **HxString** exportOp="exportPix") const
- virtual void **exportOp** (float *data, **HxTagList** &tags, **HxString** exportOp="exportPix") const
- virtual void **exportOp** (double *data, **HxTagList** &tags, **HxString** exportOp="exportPix") const
- virtual void **inout** (**HxString** inOutOp, **HxTagList** &tags) const
- virtual void **inout** (void *data, **HxString** dataType, **HxString** inOutOp, **HxTagList** &tags) const

Set and border operations

- virtual void **set** (const **HxValue** val)
- virtual void **set** (const HxImageData *arg)
- virtual void **set** (int *pixels)
- virtual void **set** (HxByte *pixels)
- virtual void **set** (double *pixels)=0
- void **setPartImage** (const HxImageData *src)

- void **setPartImage** (const HxImageData ∗src, **HxTagList** &tags)
- void **setPartImage** (const HxImageData ∗src, **HxPointInt** srcBegin, **HxPointInt** srcEnd, **HxPointInt** dstBegin)
- void **setBorder** (**HxTagList** &tags)
- void **setBorder** (**HxSizes** borderSize, **HxTagList** &tags, HxBorderType borderType=HXBORDERMIRROR)
- void **setBorder** (HxBorderType borderType, **HxSizes** borderSize, **HxTagList** &tags)
- void **setBorder** (**HxSizes** borderSize, **HxValue** val)
- void **mirrorBorder** (**HxSizes** borderSize)
- void **propagateBorder** (**HxSizes** borderSize)

Generic operations

- virtual void **unaryPixOp** (HxImageData ∗src, **HxString** upoName, **HxTagList** &tags)

  Unary pixel operation.

- virtual void **binaryPixOp** (const HxImageData ∗arg1, const HxImageData ∗arg2, **HxString** bpoName, **HxTagList** &tags)

  Binary pixel operation.

- virtual void **multiPixOp** (HxImageData ∗∗args, const int nArgs, **HxString** mpoName, **HxTagList** &tags)

  Multi pixel operation.

- virtual void **generalizedConvolution** (const HxImageData ∗srcImg, const HxImageData ∗kerImg, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Generalized convolution operation.

- virtual void **generalizedConvolutionK1d** (const HxImageData ∗srcImg, int dimension, const HxImageData ∗kerImg, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Generalized convolution operation in one dimension.

- virtual void **neighbourhoodOp** (const HxImageData ∗src, **HxString** ngbName, **HxTagList** &tags)=0

  Neighbourhood operation.

- virtual void **neighbourhoodOp** (const HxImageData ∗src, const HxImageData ∗kernel, **HxString** ngbName, **HxTagList** &tags)=0

  Neighbourhood operation with kernel.

- virtual void **recursiveNeighOp** (const HxImageData ∗srcImg, const HxImageData ∗kerData, **HxString** genMul, **HxString** genAdd, **HxTagList** &tags)

  Recursive neighbourhood operation.

- virtual void **rgbOp** (**HxString** rgbName, **HxTagList** &tags)

  A scratch image with the same signature as the source image and the proper border size will be allocated.

- void **MNPixOp** (HxImageData ∗∗results, const int resultCnt, HxImageData ∗∗args, const int argCnt, **HxString** mpoName, **HxTagList** &tags)

  M output N input pixel operation.

Geometric operations

- virtual void **geometricOp2d** (const HxImageData *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)=0
- virtual void **restrict** (const HxImageData *arg, **HxPoint** begin)
- virtual void **extend** (const HxImageData *arg, **HxPoint** begin)
- virtual HxImageData * **projectDomain** (int dimension, int coordinate)=0
- virtual void **inverseProjectDomain** (int dimension, int coordinate, const HxImageData *arg)=0

Sample operations

- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)=0
- virtual **HxValue sampleIdentMask** (const HxImageData *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc)=0
- virtual void **sampleIdentMask** (const HxImageData *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc, **HxValueListBackInserter** res)=0
- virtual **HxValue sampleWeightMask** (const HxImageData *mask, **HxPoint** p, **HxString** sFunc)=0

Misc operations

- virtual void **setAt** (int x, int y, int z, const **HxValue** v)=0
- virtual **HxValue getAt** (int x, int y, int z) const=0
- virtual **HxRcObject** * **clone** () const
- virtual void **setPpmPixels** (const HxByte *pixels)
- virtual void **getPpmPixels** (HxByte *pixels)
- virtual void **getRgbPixels2d** (int *pixels, **HxString** dispF, int bufWidth, int bufHeight, int VX, int VY, int VW, int VH, double SX, double SY, double scaleX, double scaleY, **HxGeoIntType** gi) const
- virtual void **getDoublePixels** (double *pixels)=0
- virtual STD_OSTREAM & **printInfo** (STD_OSTREAM &os, int doData=0) const=0
- void **setObjectObserver** (const HxObjectObserver &)
- bool **probeMNpo** (const **HxImageSignature** resultsSig, const **HxImageSignature** argsSig, **HxString** mpoName, **HxTagList** &tags)

To be rewritten

- virtual void **convertColor** (**HxVec3Double** scale1, **HxVec3Double** gamma1, HxUpoVec3Double step1, HxUpoVec3Double step2, HxUpoVec3Double step3, **HxVec3Double** gamma2, **HxVec3Double** scale2)=0
- virtual HxImageData * **projectRange** (int dimension)
- virtual void **inverseProjectRange** (int dimension, const HxImageData *arg)
- virtual void **transpose** (const HxImageData *src)=0

Public Methods

- void **weight** (double w)
- **HxScalarDouble weight** () const

Protected Methods

- int **checkEqualImageSig** (**HxString** func, const HxImageData *arg)
- int **checkEqualImageSizes** (**HxString** func, const HxImageData *arg)
- int **checkEqualImageSigAndSizes** (**HxString** func, const HxImageData *arg)
- int **checkLargerImageSigAndSizes** (**HxString** func, const HxImageData *arg)
- int **checkProperKernelSigAndSizes** (**HxString** func, const HxImageData *kernel, int reqDim, int reqScalar, int appliedDim=0)
- int **checkEqualImageSizesDim** (**HxString** func, const HxImageData *arg, int dimension)
- int **checkImageDimension** (**HxString** func, int dim, int coord)
- int **checkPixelDimension** (**HxString** func, int dim)
- bool **checkBorderSize** (**HxString** funcName, **HxSizes** borderSize) const
- **HxSizes getProjectDomainSizes** (int dimension) const

6.39.1 Detailed Description

The HxImageData class is the root in the cpp image hierarchy.

All operations on images can be accessed by this interface, yet little of them are implemented by this class. This class serves as a handle of all types of images that are derived from this base class. The HxImageData class provides methods to make inquiries about the type of the image.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 HxImageData::HxImageData ()

Constructor.

```
42 {
43     _ident = _nr++;
44     _name = HxString("image") + makeString(_ident);
45     _weight = 1;
46     HxImageDataRepository::instance()->insertImage(this);
47     if (_objectObserver)
48         _objectObserver->constructed(_name);
49 }
```

6.39.2.2 HxImageData::HxImageData (const HxImageData &)

Copy constructor.

```
51                                          : HxRcObject()
52 {
53     HxEnvironment::instance()->warningStream() <<
54         "HxImageData copy constructor called" << STD_ENDL;
55     HxEnvironment::instance()->flush();
56 }
```

6.39.2.3 HxImageData::~HxImageData () [virtual]

Destructor.

```
59 {
60     HxImageDataRepository::instance()->removeImage(this);
61     if (_objectObserver)
62         _objectObserver->destructed(_name);
63 }
```

6.39.3 Member Function Documentation

6.39.3.1 void HxImageData::unaryPixOp (HxImageData * *srcImg*, HxString *upoName*, HxTagList & *tags*) [virtual]

Unary pixel operation.

```
344 {
345     HxImgFtorUpoKey funcKey(signature().toString(),
346                             srcImg->signature().toString(),
347                             upoName);
348
349     typedef HxImgFtorI2 FunctorType;
350
351     static HxImgFtorTableTem<FunctorType> funcTable;
352     FunctorType* func = funcTable.find(funcKey);
353
354     if (func) {
355         func->callIt(this, srcImg, tags);
356     } else {
357         HxEnvironment::instance()->errorStream()
358             << "Can't find " << funcKey << STD_ENDL;
359         HxEnvironment::instance()->flush();
360     }
361 }
```

6.39.3.2 void HxImageData::binaryPixOp (const HxImageData * *arg1*, const HxImageData * *arg2*, HxString *bpoName*, HxTagList & *tags*) [virtual]

Binary pixel operation.

```
366 {
367     if (!checkEqualImageSizes("binaryPixOp", arg1) ||
368             !checkEqualImageSizes("binaryPixOp", arg2))
369         return;
370
371     HxImgFtorBpoKey funcKey(signature().toString(),
372                             arg1->signature().toString(),
373                             arg2->signature().toString(),
374                             bpoName);
375
376     typedef HxImgFtorI3 FunctorType;
377
378     static HxImgFtorTableTem<FunctorType> funcTable;
379     FunctorType* func = funcTable.find(funcKey);
380
381     if (func) {
```

```
382         func->callIt(this, arg1, arg2, tags);
383     } else {
384         HxEnvironment::instance()->errorStream()
385             << "Can't find " << funcKey << STD_ENDL;
386         HxEnvironment::instance()->flush();
387     }
388 }
```

6.39.3.3 void HxImageData::multiPixOp (HxImageData ** *srcImgs*, const int *nImgs*, HxString *mpoName*, HxTagList & *tags*) [virtual]

Multi pixel operation.

```
394 {
395     HxImgFtorMpoKey funcKey(signature().toString(),
396                             srcImgs[0]->signature().toString(),
397                             mpoName);
398 
399     typedef HxImgFtorIM FunctorType;
400 
401     static HxImgFtorTableTem<FunctorType> funcTable;
402     FunctorType* func = funcTable.find(funcKey);
403 
404     if (func) {
405         func->callIt(this, srcImgs, nImgs, tags);
406     } else {
407         HxEnvironment::instance()->errorStream()
408             << "Can't find " << funcKey << STD_ENDL;
409         HxEnvironment::instance()->flush();
410     }
411 }
```

6.39.3.4 void HxImageData::MNPixOp (HxImageData ** *resImgs*, const int *resCnt*, HxImageData ** *srcImgs*, const int *srcCnt*, HxString *mpoName*, HxTagList & *tags*) [static]

M output N input pixel operation.

```
441 {
442     HxImgFtorMNpoKey funcKey(resImgs[0]->signature().toString(),
443                              srcImgs[0]->signature().toString(),
444                              mpoName);
445 
446     typedef HxImgFtorIMN FunctorType;
447 
448     static HxImgFtorTableTem<FunctorType> funcTable;
449     FunctorType* func = funcTable.find(funcKey);
450 
451     if (func) {
452         func->callIt(resImgs, resCnt, srcImgs, srcCnt, tags);
453     } else {
454         HxEnvironment::instance()->errorStream()
455             << "Can't find " << funcKey << STD_ENDL;
456         HxEnvironment::instance()->flush();
457     }
458 }
```

6.39.3.5 void HxImageData::generalizedConvolution (const HxImageData * *srcImg*, const HxImageData * *kerImg*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Generalized convolution operation.

```
466 {
467     HxSizes borderSize = kerImg->sizes() / HxSizes(2, 2, 2);
468     if (!checkBorderSize("generalizedConvolution", borderSize))
469         return;
470     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
471     HxImageData* scratchImg
472         = HxImgDataFactory::instance().makeImage(
473                                       kerImg->signature(), scratchSize);
474     scratchImg->setPartImage(
475                     srcImg, HxPointInt(0, 0, 0),
476                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
477     scratchImg->setBorder(borderSize, tags);
478
479     HxImgFtorGenConvKey funcKey(
480                             signature().toString(),
481                             scratchImg->signature().toString(),
482                             kerImg->signature().toString(),
483                             genMul, genAdd, kerName);
484
485     typedef HxImgFtorI3 FunctorType;
486
487     static HxImgFtorTableTem<FunctorType> funcTable;
488     FunctorType* func = funcTable.find(funcKey);
489
490     if (func) {
491         func->callIt(this, scratchImg, kerImg, tags);
492     } else {
493         HxEnvironment::instance()->errorStream()
494             << "Can't find " << funcKey << STD_ENDL;
495         HxEnvironment::instance()->flush();
496     }
497
498     delete scratchImg;
499 }
```

6.39.3.6 void HxImageData::generalizedConvolutionK1d (const HxImageData * *srcImg*, int *dimension*, const HxImageData * *kerImg*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Generalized convolution operation in one dimension.

```
507 {
508     HxSizes borderSize;
509
510     switch (dimension) {
511     case 1  :
512         borderSize = HxSizes(kerImg->sizes().x() / 2, 0, 0);
513         break;
514     case 2  :
515         borderSize = HxSizes(0, kerImg->sizes().x() / 2, 0);
516         break;
517     case 3  :
518         borderSize = HxSizes(0, 0, kerImg->sizes().x() / 2);
519         break;
```

```
520     }
521
522     if (!checkBorderSize("generalizedConvolutionK1d", borderSize))
523         return;
524
525     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
526     HxAddTag<int>(tags, "dimension", dimension);
527
528     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
529     HxImageSignature scratchSig(kerImg->signature());
530     scratchSig.setImageDimensionality(signature().imageDimensionality());
531     HxImageData* scratchImg
532         = HxImgDataFactory::instance().makeImage(scratchSig, scratchSize);
533
534     scratchImg->setPartImage(
535                     srcImg, HxPointInt(0, 0, 0),
536                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
537     scratchImg->setBorder(borderSize, tags);
538
539     HxImgFtorGenConvK1dKey funcKey(
540                                 signature().toString(),
541                                 scratchImg->signature().toString(),
542                                 kerImg->signature().toString(),
543                                 genMul, genAdd, kerName);
544
545     typedef HxImgFtorI3 FunctorType;
546
547     static HxImgFtorTableTem<FunctorType> funcTable;
548     FunctorType* func = funcTable.find(funcKey);
549
550     if (func) {
551         func->callIt(this, scratchImg, kerImg, tags);
552     } else {
553         HxEnvironment::instance()->errorStream()
554             << "Can't find " << funcKey << STD_ENDL;
555         HxEnvironment::instance()->flush();
556     }
557
558     delete scratchImg;
559 }
```

6.39.3.7 virtual void HxImageData::neighbourhoodOp (const HxImageData * *src*, HxString *ngbName*, HxTagList & *tags*) [pure virtual]

Neighbourhood operation.

Reimplemented in **HxImageTem** (p. 369), and **HxImageTem2d** (p. 371).

6.39.3.8 virtual void HxImageData::neighbourhoodOp (const HxImageData * *src*, const HxImageData * *kernel*, HxString *ngbName*, HxTagList & *tags*) [pure virtual]

Neighbourhood operation with kernel.

Reimplemented in **HxImageTem** (p. 369), and **HxImageTem2d** (p. 371).

6.39.3.9 void HxImageData::recursiveNeighOp (const HxImageData * *srcImg*, const HxImageData * *kerImg*, HxString *genMul*, HxString *genAdd*, HxTagList & *tags*) [virtual]

Recursive neighbourhood operation.

```
566 {
567     int     dimension = HxGetTag(tags, "dimension", 0);
568     HxSizes borderSize = kerImg->sizes() / HxSizes(2, 2, 2);
569 
570     switch(dimension)
571     {
572     case 1  :
573         borderSize = HxSizes(borderSize.x(), 0, 0);
574         break;
575     case 2  :
576         borderSize = HxSizes(0, borderSize.x(), 0);
577         break;
578     case 3  :
579         borderSize = HxSizes(0, 0, borderSize.x());
580         break;
581     }
582 
583     if (!checkBorderSize("recursiveNeighOp", borderSize))
584         return;
585 
586     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
587     HxImageData* scratchImg
588         = HxImgDataFactory::instance().makeImage(
589                                         kerImg->signature(), scratchSize);
590 
591     scratchImg->setPartImage(
592                     srcImg, HxPointInt(0, 0, 0),
593                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
594     scratchImg->setBorder(borderSize, tags, HXBORDERPROPAGATE);
595 
596     HxImgFtorRecNgbKey funcKey(
597                             scratchImg->signature().toString(),
598                             kerImg->signature().toString(), genMul, genAdd);
599 
600     typedef HxImgFtorI2 FunctorType;
601 
602     static HxImgFtorTableTem<FunctorType> funcTable;
603     FunctorType* func = funcTable.find(funcKey);
604 
605     if (func) {
606         HxAddTag(tags, "borderSize", borderSize);
607         func->callIt(scratchImg, kerImg, tags);
608     } else {
609         HxEnvironment::instance()->errorStream()
610             << "Can't find " << funcKey << STD_ENDL;
611         HxEnvironment::instance()->flush();
612     }
613 
614     setPartImage(
615         scratchImg, borderSize, borderSize + sizes() - HxPointInt(1, 1, 1),
616         HxPointInt(0, 0, 0));
617 
618     delete scratchImg;
619 }
```

6.39.3.10 void HxImageData::rgbOp (HxString *rgbName*, HxTagList & *tags*) [virtual]

A scratch image with the same signature as the source image and the proper border size will be allocated.

The border size will be stored in the tag list. The scratch image will be set to the source image with a mirrored border unless specified in the tag list otherwise

```
623 {
```

```
624     HxImgFtorRgbKey funcKey(signature().toString(), rgbName);
625
626     typedef HxImgFtorI1 FunctorType;
627
628     static HxImgFtorTableTem<FunctorType> funcTable;
629     FunctorType* func = funcTable.find(funcKey);
630
631     if (func) {
632         func->callIt(this, tags);
633     } else {
634         HxEnvironment::instance()->errorStream()
635             << "Can't find " << funcKey << STD_ENDL;
636         HxEnvironment::instance()->flush();
637     }
638 }
```

The documentation for this class was generated from the following files:

- **HxImageData.h**
- HxImageData.c

6.40 HxImageFactory Class Reference

Factory for **HxImageRep** (p. 313) objects.

`#include <HxImageFactory.h>`

Public Methods

- **HxImageRep fromSignature** (const **HxImageSignature** &signature, **HxSizes** sizes)

  Uninitialized image with given signature and sizes.

- **HxImageRep fromImage** (const **HxImageSignature** &signature, **HxImageRep** src)

  New image with given signature.

- **HxImageRep fromValue** (const **HxImageSignature** &signature, **HxSizes** sizes, **HxValue** val)

  New image with given signature and sizes.

- **HxImageRep fromByteData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, HxByte *data)

  New image with given signature and sizes.

- **HxImageRep fromShortData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, short *data)

  New image with given signature and sizes.

- **HxImageRep fromIntData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, int *data)

  New image with given signature and sizes.

- **HxImageRep fromFloatData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, float *data)

  New image with given signature and sizes.

- **HxImageRep fromDoubleData** (int pixelDimensionality, int dimensions, **HxSizes** sizes, double *data)

New image with given signature and sizes.

- **HxImageRep fromGenerator** (const **HxImageSignature** &signature, const HxImageGenerator ∗imgGenerator) const

  New image with given signature.

- **HxImageRep fromNamedGenerator** (const **HxImageSignature** &signature, **HxString** generatorName, **HxTagList** &tags) const

  New image with given signature.

- **HxImageRep fromJavaRgb** (const **HxImageSignature** &signature, **HxSizes** sizes, int ∗pixels)

  New image with given signature and sizes.

- **HxImageRep fromGrayValue** (const **HxImageSignature** &signature, **HxSizes** sizes, HxByte ∗pixels)

  New image with given signature and sizes.

- **HxImageRep fromMatlab** (const **HxImageSignature** &signature, **HxSizes** sizes, double ∗pixels)

  New image with given signature and sizes.

- **HxImageRep fromImport** (const **HxImageSignature** &signature, **HxSizes** sizes, **HxString** importOp, **HxTagList** &tags)

  New image with given signature and sizes.

- **HxImageRep from2Images** (**HxImageRep** i1, **HxImageRep** i2)

  New image with pixel values "stacked" from given arguments.

- **HxImageRep from3Images** (**HxImageRep** i1, **HxImageRep** i2, **HxImageRep** i3)

  New image with pixel values "stacked" from given arguments.

- **HxImageRep fromFile** (**HxString** fileName)

  New image read from file using scil-image.

- **HxImageRep fromFile** (**HxString** fileName, **HxTagList** &tags)

  New image read from file using HxImgFileIo lib.

- bool **writeFile** (**HxImageRep** img, **HxString** fileName, **HxTagList** &tags) const

  Write image to file using HxImgFileIo lib.

- bool **imagesToFile** (**HxImageList** imgs, **HxString** fileName, **HxTagList** &tags) const

  Write set of image to file using HxImgFileIo lib.

- **HxImageList imagesFromFile** (**HxString** fileName, **HxTagList** &tags)

  Read set of image from file using HxImgFileIo lib.

- void **subscribeGenerator** (**HxString** name, HxImageGenerator ∗)
- void **unSubscribeGenerator** (**HxString** name, HxImageGenerator ∗)
- HxImageGenerator ∗ **getGenerator** (**HxString** name) const

Static Public Methods

- HxImageFactory & **instance** ()

  The one and only instance of this class.

6.40.1 Detailed Description

Factory for **HxImageRep** (p. 313) objects.

6.40.2 Member Function Documentation

6.40.2.1 HxImageFactory & HxImageFactory::instance () [static]

The one and only instance of this class.

```
25 {
26     static HxImageFactory theFactory;
27     return theFactory;
28 }
```

6.40.2.2 HxImageRep HxImageFactory::fromSignature (const HxImageSignature & *signature*, HxSizes *sizes*)

Uninitialized image with given signature and sizes.

```
33 {
34     HxImageData* imgData = construct(signature, sizes);
35     return HxImageRep(imgData);
36 }
```

6.40.2.3 HxImageRep HxImageFactory::fromImage (const HxImageSignature & *signature*, HxImageRep *src*)

New image with given signature.

Sizes and data are taken from given image.

```
40 {
41     HxImageData* imgData = construct(signature, src.sizes());
42     if (imgData && src.pointee())
43         imgData->set(src.pointee());
44     return HxImageRep(imgData);
45 }
```

6.40.2.4 HxImageRep HxImageFactory::fromValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxValue *val*)

New image with given signature and sizes.

Pixel data is initialized with given value.

```
50 {
51     HxImageData* imgData = construct(signature, sizes);
52     if (imgData)
53         imgData->set(val);
54     return HxImageRep(imgData);
55 }
```

6.40.2.5 HxImageRep HxImageFactory::fromByteData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, HxByte * *data*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
60 {
61     HxImageSignature signature(
62         dimensions, pixelDimensionality, INT_VALUE, sizeof(HxByte)<<3);
63     HxImageData* imgData = construct(signature, sizes);
64     if (imgData)
65         imgData->import(data);
66     return HxImageRep(imgData);
67 }
```

6.40.2.6 HxImageRep HxImageFactory::fromShortData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, short * *data*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
72 {
73     HxImageSignature signature(
74         dimensions, pixelDimensionality, INT_VALUE, sizeof(short)<<3);
75     HxImageData* imgData = construct(signature, sizes);
76     if (imgData)
77         imgData->import(data);
78     return HxImageRep(imgData);
79 }
```

6.40.2.7 HxImageRep HxImageFactory::fromIntData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, int * *data*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
84 {
85     HxImageSignature signature(
86         dimensions, pixelDimensionality, INT_VALUE, sizeof(int)<<3);
87     HxImageData* imgData = construct(signature, sizes);
88     if (imgData)
89         imgData->import(data);
90     return HxImageRep(imgData);
91 }
```

6.40.2.8 HxImageRep HxImageFactory::fromFloatData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, float * *data*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
96 {
97     HxImageSignature signature(
98         dimensions, pixelDimensionality, REAL_VALUE, sizeof(float)<<3);
99     HxImageData* imgData = construct(signature, sizes);
100     if (imgData)
101         imgData->import(data);
102     return HxImageRep(imgData);
103 }
```

6.40.2.9 HxImageRep HxImageFactory::fromDoubleData (int *pixelDimensionality*, int *dimensions*, HxSizes *sizes*, double * *data*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
108 {
109     HxImageSignature signature(
110         dimensions, pixelDimensionality, REAL_VALUE, sizeof(double)<<3);
111     HxImageData* imgData = construct(signature, sizes);
112     if (imgData)
113         imgData->import(data);
114     return HxImageRep(imgData);
115 }
```

6.40.2.10 HxImageRep HxImageFactory::fromGenerator (const HxImageSignature & *signature*, const HxImageGenerator * *imgGenerator*) const

New image with given signature.

Pixel data and size are determined by image generator.

```
121 {
122     HxImageData* imgData = construct(signature, imgGenerator->domainSize());
123     HxTagList tags;
124     HxAddTag(tags, "imageGenerator", imgGenerator);
125     if (imgData)
126     {
127         imgData->inout("generate", tags);
128     }
129     return HxImageRep(imgData);
130 }
```

6.40.2.11 HxImageRep HxImageFactory::fromNamedGenerator (const HxImageSignature & *signature*, HxString *generatorName*, HxTagList & *tags*) const

New image with given signature.

Pixel data and size are determined by image generator.

```
136 {
137     HxImageGenerator* generator = getGenerator(generatorName);
138 
139     if (!generator)
140         return HxImageRep();
141 
142     generator->init(tags);
143     HxImageData* imgData = construct(signature, generator->domainSize());
144     HxTagList ttags;
145     HxAddTag(ttags, "imageGenerator", generator);
146     if (imgData)
147     {
148         imgData->inout("generate", ttags);
149     }
150     return HxImageRep(imgData);
151 }
```

6.40.2.12 HxImageRep HxImageFactory::fromJavaRgb (const HxImageSignature & *signature*, HxSizes *sizes*, int * *pixels*)

New image with given signature and sizes.

Pixel data is initialized from given values. The given values are stored in Java RGB format.

```
156 {
157     HxImageData* imgData = construct(signature, sizes);
158     if (imgData)
159         imgData->set(pixels);
160     return HxImageRep(imgData);
161 }
```

6.40.2.13 HxImageRep HxImageFactory::fromGrayValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxByte * *pixels*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
166 {
167     HxImageData* imgData = construct(signature, sizes);
168     if (imgData)
169         imgData->set(pixels);
170     return HxImageRep(imgData);
171 }
```

6.40.2.14 HxImageRep HxImageFactory::fromMatlab (const HxImageSignature & *signature*, HxSizes *sizes*, double * *pixels*)

New image with given signature and sizes.

Pixel values are initialized from given values. The given values are stored in Matlab format.

```
176 {
177     HxImageData* imgData = construct(signature, sizes);
178     if (imgData)
```

```
179         imgData->set(pixels);
180     return HxImageRep(imgData);
181 }
```

6.40.2.15 HxImageRep HxImageFactory::fromImport (const HxImageSignature & *signature*, HxSizes *sizes*, HxString *importOp*, HxTagList & *tags*)

New image with given signature and sizes.

Pixel values are initialized by specified import operator.

```
187 {
188     HxImageData* imgData = construct(signature, sizes);
189     if (imgData)
190         imgData->inout(importOp, tags);
191     return HxImageRep(imgData);
192 }
```

6.40.2.16 HxImageRep HxImageFactory::from2Images (HxImageRep *i1*, HxImageRep *i2*)

New image with pixel values ”stacked” from given arguments.

For example, if i1 and i2 are scalar images the pixel values in the new image are 2-vectors. Result may need exceed highest pixel dimensionality.

```
196 {
197     STD_OSTREAM& errorStream = HxEnvironment::instance()->errorStream();
198 
199     if (!(i1 && i2)) {
200         errorStream << "Source images may not be null" << STD_ENDL;
201         HxEnvironment::instance()->flush();
202         return HxImageRep();
203     }
204     if (i1.signature() != i2.signature()) {
205         errorStream << "Source images differ in type" << STD_ENDL;
206         HxEnvironment::instance()->flush();
207         return HxImageRep();
208     }
209     if (i1.sizes() != i2.sizes()) {
210         errorStream << "Source images differ in size" << STD_ENDL;
211         HxEnvironment::instance()->flush();
212         return HxImageRep();
213     }
214     HxImageSignature signature(i1.signature());
215     HxSizes sizes(i1.sizes());
216 
217     if (signature.pixelDimensionality() != 1) {
218         errorStream << "HxImageRep: images must have pixelDimensionality of 1"
219                     << STD_ENDL;
220         HxEnvironment::instance()->flush();
221         return HxImageRep();
222     }
223     signature.setPixelDimensionality(2);
224     HxImageData* imgData = construct(signature, sizes);
225     if (imgData)
226     {
227         HxTagList tags;
228         imgData->binaryPixOp(
229             i1.pointee(), i2.pointee(), "vector2d", tags);
```

```
230     }
231     return HxImageRep(imgData);
232 }
```

6.40.2.17 HxImageRep HxImageFactory::from3Images (HxImageRep *i1*, HxImageRep *i2*, HxImageRep *i3*)

New image with pixel values ”stacked” from given arguments.

For example, if i1, i2, and i3 are scalar images the pixel values in the new image are 3-vectors. Result may need exceed highest pixel dimensionality.

```
236 {
237     STD_OSTREAM& errorStream = HxEnvironment::instance()->errorStream();
238     if (!(i1 && i2 && i3)) {
239         errorStream << "Source images may not be null" << STD_ENDL;
240         HxEnvironment::instance()->flush();
241         return HxImageRep();
242     }
243     if ((i1.signature() != i2.signature())
244             || (i2.signature() != i3.signature())) {
245         errorStream << "Source images differ in type" << STD_ENDL;
246         HxEnvironment::instance()->flush();
247         return HxImageRep();
248     }
249     if ((i1.sizes() != i2.sizes()) || (i2.sizes() != i3.sizes())) {
250         errorStream << "Source images differ in size" << STD_ENDL;
251         HxEnvironment::instance()->flush();
252         return HxImageRep();
253     }
254     HxImageSignature signature(i1.signature());
255     HxSizes sizes(i1.sizes());
256     if (signature.pixelDimensionality() != 1) {
257         errorStream << "HxImageRep: images must have pixelDimensionality of 1"
258                     << STD_ENDL;
259         HxEnvironment::instance()->flush();
260         return HxImageRep();
261     }
262     signature.setPixelDimensionality(3);
263     HxImageData* imgData = construct(signature, sizes);
264     if (imgData)
265     {
266         HxTagList tags;
267         HxImageData *pointees[3] = {i1.pointee(), i2.pointee(), i3.pointee()};
268         imgData->multiPixOp(pointees, 3, "vector3d", tags);
269     }
270     return HxImageRep(imgData);
271 }
```

6.40.2.18 HxImageRep HxImageFactory::fromFile (HxString *fileName*)

New image read from file using scil-image.

```
275 {
276     HxImageData* imgData
277         = HxImageSiLib::getHxImageSiLib()->readFile(fileName.data());
278     return HxImageRep(imgData);
279 }
```

6.40.2.19 HxImageRep HxImageFactory::fromFile (HxString *fileName*, HxTagList & *tags*)

New image read from file using HxImgFileIo lib.

```
285 {
286     HxString extName = HxSystem::instance().extName(fileName);
287 
288     HxImgFileReader* reader = HxImgFileIoTable::instance().getReader(extName);
289 
290     int imgNum = HxGetTag(tags, "setImage", 0);
291 
292     if (!reader)
293     {
294         HxEnvironment::instance()->errorStream()
295             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
296         HxEnvironment::instance()->flush();
297         return HxImageRep();
298     }
299 
300     if (reader->openFile(fileName))
301     {
302         HxImageData* imgData = 0;
303 
304         bool imgIsSet = reader->setImage(imgNum);
305         reader->getInfo(tags);
306         if (imgIsSet)
307         {
308             HxPointZ begin(0, 0, 0);
309             HxTagList tags;
310             imgData = construct(
311                         reader->getSignature(), reader->getImageSize());
312             HxString inOutOp =
313                 HxString("importPix<") + reader->getDataTypeString() + ">";
314             while (reader->moreData())
315             {
316                 HxBoundingBox boundingBox(reader->getDataSize());
317                 boundingBox = boundingBox.translate(reader->getDataBegin());
318                 HxAddTag(tags, "boundingBox", boundingBox);
319                 void* data = reader->getData();
320                 if (data)
321                     imgData->inout(
322                         data, reader->getDataTypeString(), "importPix", tags);
323             }
324         }
325         else
326         {
327             HxEnvironment::instance()->errorStream()
328                 << "" << reader->errorDescription() << STD_ENDL;
329             HxEnvironment::instance()->flush();
330         }
331         reader->closeFile();
332         return imgData ? HxImageRep(imgData) : HxImageRep();
333     }
334     else
335     {
336         HxEnvironment::instance()->errorStream()
337             << "" << reader->errorDescription() << STD_ENDL;
338         HxEnvironment::instance()->flush();
339     }
340 
341     return HxImageRep();
342 }
```

6.40.2.20 bool HxImageFactory::writeFile (HxImageRep *img*, HxString *fileName*, HxTagList & *tags*) const

Write image to file using HxImgFileIo lib.

```
347 {
348     if (!img) return false;
349 
350     HxString extName = HxSystem::instance().extName(fileName);
351 
352     HxImgFileWriter* writer = HxImgFileIoTable::instance().getWriter(extName);
353 
354     if (!writer)
355     {
356         HxEnvironment::instance()->errorStream()
357             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
358         HxEnvironment::instance()->flush();
359         return false;
360     }
361 
362     bool asRgb = HxGetTag(tags, "asRgb", true);
363     HxAddTag(tags, "asRgb", asRgb);
364 
365     if (!(writer->openFile(fileName)            &&
366         writer->setSignature(img.signature())   &&
367         writer->setImageSize(img.sizes())       &&
368         writer->setInfo(tags)))
369     {
370         HxEnvironment::instance()->errorStream()
371             << "" << writer->errorDescription() << STD_ENDL;
372         HxEnvironment::instance()->flush();
373         return false;
374     }
375 
376     HxPointZ begin(0, 0, 0);
377     HxTagList opTags;
378 
379     while (writer->moreData())
380     {
381         HxString exportName = "exportPix<" + writer->getDataTypeString() + ">";
382         HxBoundingBox boundingBox(writer->getDataSize());
383         boundingBox = boundingBox.translate(writer->getDataBegin());
384         HxAddTag(opTags, "boundingBox", boundingBox);
385         void* data = writer->getData();
386         if (data)
387         {
388             HxAddTag(opTags, "dataPtr", data);
389             img.exportOp(exportName, opTags);
390             writer->putData(data);
391         }
392     }
393 
394     writer->closeFile();
395 
396     return true;
397 }
```

6.40.2.21 bool HxImageFactory::imagesToFile (HxImageList *imgs*, HxString *fileName*, HxTagList & *tags*) const

Write set of image to file using HxImgFileIo lib.

```
466 {
467     if (!imgs.size()) return false;
468 
469     HxString extName = HxSystem::instance().extName(fileName);
470 
471     HxImgFileWriter* writer = HxImgFileIoTable::instance().getWriter(extName);
472 
473     if (!writer)
474     {
475         HxEnvironment::instance()->errorStream()
476             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
477         HxEnvironment::instance()->flush();
478         return false;
479     }
480 
481     bool asRgb = HxGetTag(tags, "asRgb", true);
482     HxAddTag(tags, "asRgb", asRgb);
483 
484     HxImageList::const_iterator img = imgs.begin();
485 
486     if (!(writer->firstImage()                  &&
487         writer->openFile(fileName)             &&
488         writer->setSignature((*img).signature())   &&
489         writer->setImageSize((*img).sizes())       &&
490         writer->setInfo(tags)))
491     {
492         HxEnvironment::instance()->errorStream()
493             << "" << writer->errorDescription() << STD_ENDL;
494         HxEnvironment::instance()->flush();
495         return false;
496     }
497 
498     HxTagList opTags;
499 
500     while (img != imgs.end()) {
501         while (writer->moreData())
502         {
503             HxString exportName = "exportPix<" + writer->getDataTypeString() + ">";
504             HxBoundingBox boundingBox(writer->getDataSize());
505             boundingBox = boundingBox.translate(writer->getDataBegin());
506             HxAddTag(opTags, "boundingBox", boundingBox);
507             void* data = writer->getData();
508             if (data)
509             {
510                 HxAddTag(opTags, "dataPtr", data);
511                 (*img).exportOp(exportName, opTags);
512                 writer->putData(data);
513             }
514         }
515         img++;
516         if (img != imgs.end()) {
517             writer->nextImage();
518             if (!( writer->setSignature((*img).signature())   &&
519                 writer->setImageSize((*img).sizes())       &&
520                 writer->setInfo(tags)))
521             {
522                 HxEnvironment::instance()->errorStream()
523                     << "" << writer->errorDescription() << STD_ENDL;
524                 HxEnvironment::instance()->flush();
525                 return false;
526             }
527         }
528     }
529 
530     writer->closeFile();
```

```
531
532     return true;
533 }
```

6.40.2.22 HxImageList HxImageFactory::imagesFromFile (HxString *fileName*, HxTagList & *tags*)

Read set of image from file using HxImgFileIo lib.

```
401 {
402     HxString extName = HxSystem::instance().extName(fileName);
403
404     HxImgFileReader* reader = HxImgFileIoTable::instance().getReader(extName);
405
406     HxImageList imgs;
407
408     if (!reader)
409     {
410         HxEnvironment::instance()->errorStream()
411             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
412         HxEnvironment::instance()->flush();
413         return imgs;
414     }
415
416     if (reader->openFile(fileName))
417     {
418         int imgNum;
419         for (imgNum = 0; imgNum < reader->imageCount(); imgNum++) {
420             bool imgIsSet = reader->setImage(imgNum);
421             reader->getInfo(tags);
422
423             if (imgIsSet)
424             {
425                 HxImageData* imgData = 0;
426                 HxPointZ begin(0, 0, 0);
427                 HxTagList tags;
428                 imgData = construct(
429                             reader->getSignature(), reader->getImageSize());
430                 HxString inOutOp =
431                     HxString("importPix<") + reader->getDataTypeString() + ">";
432                 while (reader->moreData())
433                 {
434                     HxBoundingBox boundingBox(reader->getDataSize());
435                     boundingBox = boundingBox.translate(reader->getDataBegin());
436                     HxAddTag(tags, "boundingBox", boundingBox);
437                     void* data = reader->getData();
438                     if (data && imgData)
439                         imgData->inout(
440                             data, reader->getDataTypeString(), "importPix", tags);
441                 }
442                 imgs += imgData ? HxImageRep(imgData) : HxImageRep();
443             }
444             else
445             {
446                 HxEnvironment::instance()->errorStream()
447                     << "" << reader->errorDescription() << STD_ENDL;
448                 HxEnvironment::instance()->flush();
449             }
450         }
451         reader->closeFile();
452     }
453     else
454     {
```

```
455         HxEnvironment::instance()->errorStream()
456             << "" << reader->errorDescription() << STD_ENDL;
457         HxEnvironment::instance()->flush();
458     }
459
460     return imgs;
461 }
```

The documentation for this class was generated from the following files:

- **HxImageFactory.h**
- HxImageFactory.c

6.41 HxImageList Class Reference

A list of images.

`#include <HxImageList.h>`

Public Types

- typedef std::list< **HxImageRep** >::const_iterator **const_iterator**
- typedef std::list< **HxImageRep** >::iterator **iterator**

Public Methods

- **HxImageList** ()

  Constructor.

- **HxImageList** (**HxImageRep** e)
- **HxImageList** (**HxImageRep** e1, **HxImageRep** e2)
- **HxImageList** (**HxImageRep** e1, **HxImageRep** e2, **HxImageRep** e3)
- **HxImageList** (**HxImageRep** e1, **HxImageRep** e2, **HxImageRep** e3, **HxImageRep** e4)
- **HxImageList** (**HxImageRep** e1, **HxImageRep** e2, **HxImageRep** e3, **HxImageRep** e4, **HxImageRep** e5)
- **HxImageList** (const HxImageList &l)

  Copy Constructor.

- ~**HxImageList** ()

  Destructor.

- HxImageList & **operator=** (const HxImageList &l)
- **HxImageRep** & **operator[ ]** (const int i)
- **HxImageData** ** **pointees** () const

  The entry points of the list.

- int **nImages** () const
- int **size** () const
- iterator **begin** ()
- iterator **end** ()

- const_iterator **begin** () const
- const_iterator **end** () const
- void **operator+=** (const **HxImageRep** &e)
- HxImageList **unaryPixOp** (**HxString** upoName, **HxTagList** &tags=HxMakeTagList()) const

  Apply a unaryPixOp to all elements.

- HxImageList **binaryPixOp** (const **HxImageRep** arg, **HxString** bpoName, **HxTagList** &tags=HxMakeTagList()) const

  Apply a binaryPixOp to all elements.

- HxImageList **binaryPixOp** (const **HxValue** arg, **HxString** bpoName, **HxTagList** &tags=HxMakeTagList()) const

  Apply a binaryPixOp with value to all elements.

- **HxImageRep multiPixOp** (**HxString** mpoName, **HxTagList** &tags=HxMakeTagList())

  Apply a multiPixOp to the list, thereby contracting to an **HxImageRep** (p. 313).

- HxImageList **MNPixOp** (**HxString** mpoName, **HxTagList** &tags=HxMakeTagList())

  Apply a MNPixOp to the list, thereby contracting to an HxImageList.

Friends

- HxImageList **operator+** (const HxImageList &l, const **HxImageRep** &e)
- HxImageList **operator+** (const **HxImageRep** &e, const HxImageList &l)
- HxImageList **operator+** (const HxImageList &l1, const HxImageList &l2)

6.41.1 Detailed Description

A list of images.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 HxImageList::HxImageList () [inline]

Constructor.

```
87 {
88 }
```

6.41.2.2 HxImageList::HxImageList (const HxImageList & *l*) [inline]

Copy Constructor.

```
134 {
135     _ims = l._ims;
136 }
```

6.41.2.3 HxImageList::~HxImageList () [inline]

Destructor.

```
140 {
141 }
```

6.41.3 Member Function Documentation

6.41.3.1 HxImageData** HxImageList::pointees () const

The entry points of the list.

6.41.3.2 HxImageList HxImageList::unaryPixOp (HxString *upoName*, HxTagList & *tags* = HxMakeTagList()) const [inline]

Apply a unaryPixOp to all elements.

```
201 {
202     HxImageList::const_iterator i;
203     HxImageList res;
204 
205     for (i = _ims.begin(); i != _ims.end(); i++)
206         res += (*i).unaryPixOp(upoName, tags);
207 
208     return res;
209 }
```

6.41.3.3 HxImageList HxImageList::binaryPixOp (const HxImageRep *arg*, HxString *bpoName*, HxTagList & *tags* = HxMakeTagList()) const [inline]

Apply a binaryPixOp to all elements.

```
214 {
215     HxImageList::const_iterator i;
216     HxImageList res;
217 
218     for (i = _ims.begin(); i != _ims.end(); i++)
219         res += (*i).binaryPixOp(arg, bpoName, tags);
220 
221     return res;
222 }
```

6.41.3.4 HxImageList HxImageList::binaryPixOp (const HxValue *arg*, HxString *bpoName*, HxTagList & *tags* = HxMakeTagList()) const [inline]

Apply a binaryPixOp with value to all elements.

```
227 {
228     HxImageList::const_iterator i;
229     HxImageList res;
230 
```

```
231     for (i = _ims.begin(); i != _ims.end(); i++)
232         res += (*i).binaryPixOp(arg, bpoName, tags);
233 
234     return res;
235 }
```

6.41.3.5 HxImageRep HxImageList::multiPixOp (HxString *mpoName*, HxTagList & *tags* = HxMakeTagList()) [inline]

Apply a multiPixOp to the list, thereby contracting to an **HxImageRep** (p. 313).

```
239 {
240     HxImageRep im, res;
241 
242     im = _ims.front();
243     _ims.pop_front();
244 
245     res = im.multiPixOp(*this, mpoName, tags);
246 
247     _ims.push_front(im);
248 
249     return res;
250 }
```

6.41.3.6 HxImageList HxImageList::MNPixOp (HxString *mpoName*, HxTagList & *tags* = HxMakeTagList()) [inline]

Apply a MNPixOp to the list, thereby contracting to an HxImageList.

```
254 {
255     HxImageRep im;
256     HxImageList res;
257 
258     im = _ims.front();
259     _ims.pop_front();
260 
261     res = im.MNPixOp(*this, mpoName, tags);
262 
263     _ims.push_front(im);
264 
265     return res;
266 }
```

The documentation for this class was generated from the following file:

- **HxImageList.h**

6.42 HxImageRep Class Reference

Class definition of Horus image representation.

`#include <HxImageRep.h>`

Constructors

- **HxImageRep** ()

  Null image.

- **HxImageRep** (const HxImageRep &)

  Copy constructor.

Destructor

- ~**HxImageRep** ()

  Destructor.

Operators

- HxImageRep & **operator=** (const HxImageRep &)

  Assignment operator.

- int **operator==** (const HxImageRep &arg) const

  Equality.

- int **isNull** () const

  Indicates wether this is a valid image.

- **operator int** () const

  Indicates wether this is a valid image.

Inquiry

- int **ident** () const

  The unique identifier of the image.

- **HxString name** () const

  The (not necessarily unique) name of the image.

- void **name** (**HxString** s)

  Sets the name of the image.

- int **dimensionality** () const

  The number of dimensions of the image.

- int **dimensionSize** (int i) const

  The size of the image in the i-th dimension.

- **HxSizes sizes** () const

  The dimension sizes of the image.

- int **numberOfPixels** () const
  The total number of pixels in the image.
- int **pixelDimensionality** () const
  The number of dimensions of one pixel.
- **HxValueType pixelType** () const
  The pixel range value type.
- int **pixelPrecision** () const
  The pixel size in bits.
- **HxImageSignature signature** () const
  The signature of the image.

Unary pixel operations

- HxImageRep **unaryPixOp** (**HxString** upoName, **HxTagList** &tags=HxMakeTagList()) const
  Unary pixel operation.

Binary pixel operations

- HxImageRep **binaryPixOp** (const **HxValue** arg, **HxString** bpoName, **HxTagList** &tags=HxMakeTagList()) const
  Binary pixel operation.
- HxImageRep **binaryPixOp** (const HxImageRep arg, **HxString** bpoName, **HxTagList** &tags=HxMakeTagList()) const
  Binary pixel operation.

Multi pixel operations.

- HxImageRep **multiPixOp** (const **HxImageList** &args, **HxString** mpoName, **HxTagList** &tags=HxMakeTagList()) const
  Multi pixel operation.
- **HxImageList MNPixOp** (const **HxImageList** &args, **HxString** mpoName, **HxTagList** &tags=HxMakeTagList()) const
  M output N input pixel operation.

Reduce operations

- **HxValue reduceOp** (**HxString** op, **HxTagList** &tags=HxMakeTagList()) const
  Reduce operation.

Generalized convolution operations.

- HxImageRep **generalizedConvolution** (HxImageRep kerImg, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation.

- HxImageRep **generalizedConvolutionK1d** (int dimension, HxImageRep kerImg, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation in one dimension.

- HxImageRep **genConvSeparated** (HxImageRep kernel, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation separated for each dimension.

- HxImageRep **genConvSeparated** (int dimension, HxImageRep kernel1, HxImageRep kernel2, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation separated for each dimension.

Neighbourhood operations.

- HxImageRep **neighbourhoodOp** (**HxString** ngbName, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation.

- HxImageRep **neighbourhoodOp** (HxImageRep kernel, **HxString** ngbName, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation with kernel.

- HxImageRep **recursiveNeighOp** (HxImageRep kerImg, **HxString** gMul, **HxString** gAdd, **HxTagList** &tags=HxMakeTagList(), **ResultPrecision** resPrec=DEFAULT_PREC) const

  Recursive neighbourhood operation.

Geometric operations.

- HxImageRep **geometricOp2d** (**HxMatrix** func, **HxGeoIntType** gi=LINEAR, HxGeoTransType gt=FORWARD, int adjustSize=1, **HxValue** background=**HxValue**(0)) const

  Geometric operation in 2D.

Sample operations

- **HxValue sampleIdentMask** (const HxImageRep mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc) const

  Sample the image with the given identification mask.

- void **sampleIdentMask** (const HxImageRep mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc, **HxValueListBackInserter** res) const

Sample the image with the given identification mask.

- **HxValue sampleWeightMask** (const HxImageRep mask, **HxPoint** p, **HxString** sFunc) const

  Sample the image with the given weight mask.

Misc operations

- void **exportOp** (**HxString** exportName, **HxTagList** &tags=HxMakeTagList()) const

  Survey operations.

- void **setAt** (const **HxValue** v, int x, int y=0, int z=0)

  setAt is to be removed.

- **HxValue getAt** (int x, int y=0, int z=0) const

  Return pixel value at given position.

- STD_OSTREAM & **printInfo** (STD_OSTREAM &os, int doData=0)

  Print information.

Output/display operations

- void **getRgbPixels2d** (int *pixels, **HxString** displayMode, int resWidth=-1, int resHeight=-1, **HxGeoIntType** gi=NEAREST) const

  The getRgbPixels functions fill an externally allocated buffer(pixels) with pixelvalues of the (2d) image.

- void **getRgbPixels2d** (int *pixels, **HxString** displayMode, int bufWidth, int bufHeight, int VX, int VY, int VW, int VH, double SX, double SY, double scaleX, double scaleY, **HxGeoIntType** gi) const

  Partial display for large images.

- void **getRgbPixels3d** (int *pixels, **HxString** displayMode, int dimension, int coordinate, int resWidth=-1, int resHeight=-1, **HxGeoIntType** gi=NEAREST) const

For testing purposes only :-)

- **HxImageData** * **dirty** ()
- **HxString ref** () const
- void **setObjectObserver** (const HxObjectObserver &)
- void **setImageDataObserver** (const HxObjectObserver &)

Public Types

- enum **ResultPrecision** { **DEFAULT_PREC**, **SOURCE_PREC**, **ARITH_PREC**, **SMALL_PREC** }

  For some operations a precision needs to be specified.

Static Public Methods

- void **setDefaultResultPrecision** (**ResultPrecision** resPrec)

  Set the value for DEFAULT_PREC.

- **ResultPrecision getResultPrecision** (**ResultPrecision** resPrec=DEFAULT_PREC)

  Get the value for the result precision.

Friends

- class **HxImageFactory**
- class **HxImageRepInit**
- HxImageRep L_HXIMAGEREP **HxProjectRange** (HxImageRep im, int dimension)

  Projection of the pixel range.

- HxImageRep L_HXIMAGEREP **HxInverseProjectRange** (HxImageRep im, int dimension, HxImageRep arg)

  Inverse projection of the pixel range.

- HxImageRep L_HXIMAGEREP **HxRestrict** (HxImageRep img, **HxPoint** begin, **HxPoint** end)

  Restriction of domain.

- HxImageRep L_HXIMAGEREP **HxExtend** (HxImageRep img, HxImageRep background, **HxPoint** begin)

  Extension of domain.

- HxImageRep L_HXIMAGEREP **HxExtendVal** (HxImageRep img, **HxSizes** newSize, **HxValue** background, **HxPoint** begin)

  Extension of domain.

- void L_HXIMAGEREP **HxGetValues** (HxImageRep img, **HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter** valPtr)
- **HxValue** L_HXIMAGEREP **HxIdentMaskMean** (HxImageRep im, HxImageRep mask, **HxPoint** p, **HxSizes** size, int label)
- **HxValue** L_HXIMAGEREP **HxIdentMaskStDev** (HxImageRep im, HxImageRep mask, **HxPoint** p, **HxSizes** size, int label)
- **HxValue** L_HXIMAGEREP **HxIdentMaskSum** (HxImageRep im, HxImageRep mask, **HxPoint** p, **HxSizes** size, int label)
- **HxValue** L_HXIMAGEREP **HxMaskSum** (HxImageRep im, HxImageRep mask, **HxPoint** p)
- HxImageRep L_HXIMAGEREP **HxMakeFromSi** (IMAGE *im)

  Make an HxImageRep with from the given image in ScilImage format.

- L_HXIMAGEREP IMAGE * **HxExportSi** (HxImageRep img)

  Export image data to a ScilImage image.

- void L_HXIMAGEREP **HxExportMatlabPixels** (HxImageRep img, double *pixels)

  Export image data to array of int.

6.42.1 Detailed Description

Class definition of Horus image representation.

6.42.2 Member Enumeration Documentation

6.42.2.1 enum HxImageRep::ResultPrecision

For some operations a precision needs to be specified.

The type of the result image will be set according to the specified precision:

SOURCE_PREC The result type is the same as the object type. ARITH_PREC The result type is the same as the type of the kernel after the kernel has been converted. SMALL_PREC The result type will be the same as above but with a smaller pixel precision. If the kernel has an integral pixel type the result pixel precision is that of short. If the kernel has a real pixel type the result pixel precision is that of float.

```
231                                     {
232                                 DEFAULT_PREC, SOURCE_PREC,
233                                 ARITH_PREC, SMALL_PREC};
```

6.42.3 Constructor & Destructor Documentation

6.42.3.1 HxImageRep::HxImageRep ()

Null image.

```
116                         : _pointee(0) {
117     if (_objectObserver)
118         _objectObserver->constructed(name());
119 }
```

6.42.3.2 HxImageRep::HxImageRep (const HxImageRep & *rhs*)

Copy constructor.

```
122     : _pointee(rhs.pointee())
123 {
124     if (_objectObserver)
125         _objectObserver->constructed(name());
126 }
```

6.42.3.3 HxImageRep::~HxImageRep ()

Destructor.

```
344 {
345     if (_objectObserver)
346         _objectObserver->destructed(name());
347 }
```

6.42.4 Member Function Documentation

6.42.4.1 HxImageRep & HxImageRep::operator= (const HxImageRep & *rhs*)

Assignment operator.

```
354 {
355     _pointee = rhs._pointee;
356     return *this;
357 }
```

6.42.4.2 int HxImageRep::operator== (const HxImageRep & *arg*) const

Equality.

```
361 {
362     return ident() == arg.ident();
363 }
```

6.42.4.3 int HxImageRep::isNull () const

Indicates wether this is a valid image.

```
367 {
368     return !int(_pointee);
369 }
```

6.42.4.4 HxImageRep::operator int () const

Indicates wether this is a valid image.

```
372 {
373     return int(_pointee);
374 }
```

6.42.4.5 int HxImageRep::ident () const

The unique identifier of the image.

```
381 {
382     return pointee() ? pointee()->ident() : 0;
383 }
```

6.42.4.6 HxString HxImageRep::name () const

The (not necessarily unique) name of the image.

```
387 {
388     return pointee() ? pointee()->name() : HxString("");
389 }
```

6.42.4.7 void HxImageRep::name (HxString *s*)

Sets the name of the image.

```
393 {
394     if (pointee())
395         pointee()->name(s);
396 }
```

6.42.4.8 int HxImageRep::dimensionality () const

The number of dimensions of the image.

```
400 {
401     return pointee() ? pointee()->dimensionality() : 0;
402 }
```

6.42.4.9 int HxImageRep::dimensionSize (int *i*) const

The size of the image in the i-th dimension.

The first dimension has i = 1.

```
406 {
407     return pointee() ? pointee()->dimensionSize(i) : 0;
408 }
```

6.42.4.10 HxSizes HxImageRep::sizes () const

The dimension sizes of the image.

```
442 {
443     return pointee() ? pointee()->sizes() : HxSizes(0, 0, 0);
444 }
```

6.42.4.11 int HxImageRep::numberOfPixels () const

The total number of pixels in the image.

```
412 {
413     return pointee() ? pointee()->numberOfPixels() : 0;
414 }
```

6.42.4.12 int HxImageRep::pixelDimensionality () const

The number of dimensions of one pixel.

```
418 {
419     return pointee() ? pointee()->pixelDimensionality() : 0;
420 }
```

6.42.4.13 HxValueType HxImageRep::pixelType () const

The pixel range value type.

INT_VALUE or REAL_VALUE or COMPLEX_VALUE.

```
424 {
425     return pointee() ? pointee()->pixelType() : INT_VALUE;
426 }
```

6.42.4.14 int HxImageRep::pixelPrecision () const

The pixel size in bits.

If the pixel has more than one dimension the size of a pixel element is returned.

```
430 {
431     return pointee() ? pointee()->pixelPrecision() : 0;
432 }
```

6.42.4.15 HxImageSignature HxImageRep::signature () const

The signature of the image.

```
436 {
437     return pointee() ? pointee()->signature() : HxImageSignature();
438 }
```

6.42.4.16 HxImageRep HxImageRep::unaryPixOp (HxString *upoName*, HxTagList & *tags* = HxMakeTagList()) const

Unary pixel operation.

The result is obtained by applying the pixel functor designated by string ”op” to all pixels in this image.

```
451 {
452     if (!pointee())
453         return HxImageRep();
454     HxMfUpo methodFrame(pointee(), upoName);
455     methodFrame.result()->unaryPixOp(methodFrame.object(), upoName, tags);
456     return HxImageRep(methodFrame.result());
457 }
```

6.42.4.17 HxImageRep HxImageRep::binaryPixOp (const HxValue *val*, HxString *bpoName*, HxTagList & *tags* = HxMakeTagList()) const

Binary pixel operation.

The result is obtained by applying the pixel functor designated by string ”bpoName” to all pairs of pixels in this image and the argument.

```
495 {
496     if (!pointee())
497         return HxImageRep();
498     HxMfUpo methodFrame(pointee(), bpoName);
499     HxAddTag(tags, "value", val);
500     methodFrame.result()->unaryPixOp(methodFrame.object(), bpoName, tags);
501     return HxImageRep(methodFrame.result());
502 }
```

6.42.4.18 HxImageRep HxImageRep::binaryPixOp (const HxImageRep *arg*, HxString *bpoName*, HxTagList & *tags* = HxMakeTagList()) const

Binary pixel operation.

The result is obtained by applying the pixel functor designated by string ”bpoName” to all pairs of pixels in this image and the argument.

```
507 {
508     if (!pointee())
509         return HxImageRep();
510     HxMfBpo methodFrame(pointee(), arg.pointee(), bpoName);
511     if (methodFrame.preOpIsOk())
512         methodFrame.result()->binaryPixOp(methodFrame.source1(),
513                                 methodFrame.source2(), bpoName, tags);
514     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
515                                     : HxImageRep();
516 }
```

6.42.4.19 HxImageRep HxImageRep::multiPixOp (const HxImageList & *args*, HxString *mpoName*, HxTagList & *tags* = HxMakeTagList()) const

Multi pixel operation.

The result is obtained by applying the pixel functor designated by string ”mpoName” to corresponding pixels in this image and all argument images.

```
524 {
525     if (!pointee())
526         return HxImageRep();
527
528     HxImageData **imsPointee = new HxImageData * [args.size()+1];
529     HxImageList::const_iterator i;
530     int n = 0;
531
532     imsPointee[n++] = pointee();
533     for (i = args.begin(); i != args.end(); i++)
534         imsPointee[n++] = (*i).pointee();
535
536     HxMfMpo methodFrame(imsPointee, n, mpoName);
537
538     methodFrame.result()->multiPixOp(methodFrame.sources(),
539                             methodFrame.nSources(), mpoName, tags);
540
541     delete [] imsPointee;
542
543     return HxImageRep(methodFrame.result());
544 }
```

6.42.4.20 HxImageList HxImageRep::MNPixOp (const HxImageList & *args*, HxString *mpoName*, HxTagList & *tags* = HxMakeTagList()) const

M output N input pixel operation.

The results is obtained by applying the pixel functor designated by string ”mpoName” to corresponding pixels in this image and all argument images.

```
552 {
553     if (!pointee())
554         return HxImageRep();
555 
556     typedef HxImageData** HxImgDataPtrArray;
557 
558     HxImageData **srcPtrs = new HxImageData * [args.size()+1];
559     HxImageList::const_iterator i;
560     int srcCnt = 0;
561 
562     srcPtrs[srcCnt++] = pointee();
563     for (i = args.begin(); i != args.end(); i++)
564         srcPtrs[srcCnt++] = (*i).pointee();
565 
566     HxMfMNpo methodFrame(srcPtrs, srcCnt, mpoName);
567 
568     delete [] srcPtrs;
569 
570     HxImageData::MNPixOp(
571         methodFrame.results(), methodFrame.resultCnt(),
572         methodFrame.sources(), methodFrame.sourceCnt(), mpoName, tags);
573 
574     HxImageList result;
575 
576     for (int n = 0; n < methodFrame.resultCnt(); n++) {
577     HxImageRep temp(methodFrame.results(n));
578         result += temp;
579     }
580 
581     return result;
582 }
```

6.42.4.21 HxValue HxImageRep::reduceOp (HxString *op*, HxTagList & *tags* = HxMakeTagList()) const

Reduce operation.

The result is obtained by reducing all pixel values in this image to a single value by applying the pixel functor designated by string ”op” repeatedly to a combination of 2 values.

```
589 {
590     exportOp(op, tags);
591     HxValue result = HxGetTag(tags, "result", HxValue(0));
592     return result;
593 }
```

6.42.4.22 void HxImageRep::setDefaultResultPrecision (ResultPrecision *resPrec*) [static]

Set the value for DEFAULT_PREC.

```
47 {
48     _defaultResPrec = resPrec;
49 }
```

6.42.4.23 HxImageRep::ResultPrecision HxImageRep::getResultPrecision (ResultPrecision *resPrec* = DEFAULT_PREC) [static]

Get the value for the result precision.

```
53 {
54     return resPrec == DEFAULT_PREC ? _defaultResPrec : resPrec;
55 }
```

6.42.4.24 HxImageRep HxImageRep::generalizedConvolution (HxImageRep *kernel*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation.

The result is obtained by sliding the kernel over this image and at each position combining the values of the kernel with the underlying values of this image by means of the pixel punctor designated by ”gMul”, and reducing this set of values to a single value by means of the pixel functor designated by ”gAdd”.

```
602 {
603     resPrec = getResultPrecision(resPrec);
604 
605     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec);
606     if (methodFrame.preOpIsOk())
607         methodFrame.object()->generalizedConvolution(
608                                 methodFrame.source(),
609                                 methodFrame.kernel(),
610                                 gMul, gAdd, "stdKernel", tags);
611     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
612                                     : HxImageRep();
613 }
```

6.42.4.25 HxImageRep HxImageRep::generalizedConvolutionK1d (int *dimension*, HxImageRep *kernel*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation in one dimension.

The result is obtained by sliding the kernel over this image and at each position combining the values of the kernel with the underlying values of this image by means of the pixel punctor designated by ”gMul”, and reducing this set of values to a single value by means of the pixel functor designated by ”gAdd”.

The kernel image should have the same dimensionality as the object image. The convolution is performed in the specified dimension only. The precision of the result type is as specified by resPrec.

```
619 {
620     resPrec = getResultPrecision(resPrec);
621 
622     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec, true);
623     if (methodFrame.preOpIsOk())
```

```
624         methodFrame.object()->generalizedConvolutionK1d(
625                                 methodFrame.source(),
626                                 dimension, methodFrame.kernel(),
627                                 gMul, gAdd, "stdKernel", tags);
628     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
629                                     : HxImageRep();
630 }
```

6.42.4.26 HxImageRep HxImageRep::genConvSeparated (HxImageRep *kernel*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation separated for each dimension.

The result is obtained by sliding the kernel over this image and at each position combining the values of the kernel with the underlying values of this image by means of the pixel punctor designated by ”gMul”, and reducing this set of values to a single value by means of the pixel functor designated by ”gAdd”.

The kernel image should have the same dimensionality as the object image. The convolution is performed separately in all dimensions using the same kernel. The precision of the result type is as specified by resPrec.

```
636 {
637     resPrec = getResultPrecision(resPrec);
638
639     HxMfGenConv methodFrame(
640                     pointee(), kernel.pointee(), resPrec, true, ARITH_PREC);
641
642     if (!methodFrame.preOpIsOk())
643         return HxImageRep();
644
645     for (int i=1; i<=dimensionality(); i++)
646     {
647         methodFrame.object()->generalizedConvolutionK1d(
648                                 methodFrame.source(),
649                                 i, methodFrame.kernel(),
650                                 gMul, gAdd, "stdKernel", tags);
651         methodFrame.setObjectAsSource();
652     }
653
654     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
655                                     : HxImageRep();
656 }
```

6.42.4.27 HxImageRep HxImageRep::genConvSeparated (int *dimension*, HxImageRep *kernel1*, HxImageRep *kernel2*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation separated for each dimension.

The result is obtained by sliding the kernel over this image and at each position combining the values of the kernel with the underlying values of this image by means of the pixel punctor designated by ”gMul”, and reducing this set of values to a single value by means of the pixel functor designated by ”gAdd”.

The kernel images should be two dimensional and the size in the second dimension should be 1. The convolution is performed using kernel1 in the specified dimension and kernel2 in all other dimensions. The precision of the result type is as specified by resPrec.

```
663 {
```

```
664     resPrec = getResultPrecision(resPrec);
665
666     HxMfGenConv methodFrame(
667                     pointee(), kernel1.pointee(), kernel2.pointee(),
668                     resPrec, true, ARITH_PREC);
669
670     if (!methodFrame.preOpIsOk())
671         return HxImageRep();
672
673     int i;
674     for (i=1; i<=dimensionality(); i++)
675     {
676         if (i == dimension)
677             methodFrame.object()->generalizedConvolutionK1d(
678                                 methodFrame.source(),
679                                 i, methodFrame.kernel(),
680                                 gMul, gAdd, "stdKernel", tags);
681         else
682             methodFrame.object()->generalizedConvolutionK1d(
683                                 methodFrame.source(),
684                                 i, methodFrame.kernel2(),
685                                 gMul, gAdd, "stdKernel", tags);
686         methodFrame.setObjectAsSource();
687     }
688
689     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
690                                     : HxImageRep();
691 }
```

6.42.4.28 HxImageRep HxImageRep::neighbourhoodOp (HxString *ngbName*, HxTagList & *tags* = HxMakeTagList()) const

Neighbourhood operation.

Slides a ”neighbourhood” over this image and offers all pixels in the neighbourhood to the functor designated by ”ngbName”. The result at each position is determined by ”ngbName”.

```
698 {
699     HxMfNgb methodFrame(pointee(), ngbName, tags);
700     if (methodFrame.preOpIsOk())
701         methodFrame.result()->neighbourhoodOp(
702                                 methodFrame.source(), ngbName, tags);
703     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
704                                     : HxImageRep();
705 }
```

6.42.4.29 HxImageRep HxImageRep::neighbourhoodOp (HxImageRep *kernel*, HxString *ngbName*, HxTagList & *tags* = HxMakeTagList()) const

Neighbourhood operation with kernel.

Slides a ”neighbourhood” over this image and offers all pixels in the neighbourhood to the functor designated by ”ngbName”. The result at each position is determined by ”ngbName”.

```
710 {
711     HxMfKernelNgb methodFrame(pointee(), kernel.pointee(), ngbName, tags);
712     if (methodFrame.preOpIsOk())
713         methodFrame.result()->neighbourhoodOp(
```

```
714                                 methodFrame.source(), methodFrame.kernel(),
715                                 ngbName, tags);
716     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
717                                     : HxImageRep();
718 }
```

6.42.4.30 HxImageRep HxImageRep::recursiveNeighOp (HxImageRep *kernel*, HxString *gMul*, HxString *gAdd*, HxTagList & *tags* = HxMakeTagList(), ResultPrecision *resPrec* = DEFAULT_PREC) const

Recursive neighbourhood operation.

THIS SHOULD BE CALLED RECURSIVE GENERALIZED CONVOLUTION!!

```
724 {
725     if (resPrec == DEFAULT_PREC)
726         resPrec = _defaultResPrec;
727     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec);
728     if (methodFrame.preOpIsOk())
729         methodFrame.object()->recursiveNeighOp(
730                                 methodFrame.source(), methodFrame.kernel(),
731                                 gMul, gAdd, tags);
732     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
733                                     : HxImageRep();
734 }
```

6.42.4.31 HxImageRep HxImageRep::geometricOp2d (HxMatrix *func*, HxGeoIntType *gi* = LINEAR, HxGeoTransType *gt* = FORWARD, int *adjustSize* = 1, HxValue *background* = HxValue(0)) const

Geometric operation in 2D.

The result is obtained by applying the given transformation to the position of each pixel in this image. By default, the actual transformation is based on the inverse of the given transformation matrix M: f(x) = M.i() $*$ x (postfix multiplication). Beware that the image coordinate system has a different orientation than the cartesian coordinate system used in specification of matrix M.

The matrix needs to be 3x3 (homogeneous coordinates)

```
743 {
744     if ((func.nCol() != 3) || (func.nRow() != 3))
745         return errorIm("geometricOp2d: matrix needs to be 3x3");
746
747     HxMatrix funcForw = (gt == FORWARD) ? func : func.i();
748     HxMatrix funcBack = (gt == FORWARD) ? func.i() : func;
749     if (!funcForw.valid() || !funcBack.valid())
750         return errorIm("geometricOp2d: matrix is not valid");
751
752     HxSizes newSize = sizes();
753     HxVec3Double translate = HxVec3Double(0,0,0);
754     if (adjustSize) {
755         HxVec3Double ulP = funcForw * HxVec3Double(0,0,1);
756         HxVec3Double llP = funcForw * HxVec3Double(0, dimensionSize(2), 1);
757         HxVec3Double urP = funcForw * HxVec3Double(dimensionSize(1), 0, 1);
758         HxVec3Double lrP = funcForw * sizes();
759         ulP /= HxScalarDouble(ulP.z()); // homogeneous coordinates
760         llP /= HxScalarDouble(llP.z());
761         urP /= HxScalarDouble(urP.z());
```

```
762         lrP /= HxScalarDouble(lrP.z());
763
764         HxVec3Double maerB = ulP.inf(llP).inf(urP).inf(lrP);
765         HxVec3Double maerE = ulP.sup(llP).sup(urP).sup(lrP);
766         newSize = HxSizes(maerE.x() - maerB.x() + 0.5,
767             maerE.y() - maerB.y() + 0.5, 1);
768         translate = HxVec3Double(maerB.x(), maerB.y(), 0);
769     }
770
771     HxMfResize methodFrame(pointee(), newSize);
772     methodFrame.object()->geometricOp2d(pointee(), funcBack, gi, translate,
773         background);
774     return HxImageRep(methodFrame.result());
775 }
```

6.42.4.32 HxValue HxImageRep::sampleIdentMask (const HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*, HxString *sFunc*) const

Sample the image with the given identification mask.

”mask” is assumed to be an identification image with pixel type short. For points within the area starting at point ”p” with given ”size” the function denoted by ”sFunc” is called if the pixel value is equal to ”label”. ”sFunc” should reside in HxSampleFunTable.

```
886 {
887     if (!pointee())
888         return HxValue(0); // ADB 14 Feb 2001, was HxImageRep();
889     HxMfReqMaskPixType methodFrame(mask.pointee(), 1, INT_VALUE, 16);
890     return pointee()->sampleIdentMask(methodFrame.mask(), p, size, label, sFunc);
891 }
```

6.42.4.33 void HxImageRep::sampleIdentMask (const HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*, HxString *sFunc*, HxValueListBackInserter *res*) const

Sample the image with the given identification mask.

”mask” is assumed to be an identification image with pixel type short. For points within the area starting at point ”p” with given ”size” the functor denoted by ”sFunc” is called if the pixel value is equal to ”label”. ”sFunc” should reside in HxSampleFunctorTable.

```
896 {
897     if (!pointee())
898         return;
899     HxMfReqMaskPixType methodFrame(mask.pointee(), 1, INT_VALUE, 16);
900     pointee()->sampleIdentMask(methodFrame.mask(), p, size, label, sFunc, res);
901 }
```

6.42.4.34 HxValue HxImageRep::sampleWeightMask (const HxImageRep *mask*, HxPoint *p*, HxString *sFunc*) const

Sample the image with the given weight mask.

The type of mask is adjusted to match the type of the image. For points within the area starting at point p with the size of the ”mask” the function denoted by ”sFunc” is called.

```
905 {
906     if (!pointee())
907         return HxValue(0); // ADB 14 Feb 2001, was HxImageRep();
908     HxMfReqMaskPixType methodFrame(mask.pointee(), pointee());
909     return pointee()->sampleWeightMask(methodFrame.mask(), p, sFunc);
910 }
```

6.42.4.35 void HxImageRep::exportOp (HxString *exportName*, HxTagList & *tags* = HxMakeTagList()) const

Survey operations.

```
917 {
918     if (HxImgFtorRuleBase::instance().getIsModifying("inout", exportName))
919     {
920         HxEnvironment::instance()->errorStream()
921             << "Error: " << exportName << " is an image data modifying "
922             << "operator" << STD_ENDL;
923         return;
924     }
925     if (pointee())
926         pointee()->inout(exportName, tags);
927 }
```

6.42.4.36 void HxImageRep::setAt (const HxValue *v*, int *x*, int *y* = 0, int *z* = 0)

setAt is to be removed.

```
931 {
932     if (_pointee) {
933         _pointee.getUnshared();
934         _pointee->setAt(x, y, z, v);
935     }
936 }
```

6.42.4.37 HxValue HxImageRep::getAt (int *x*, int *y* = 0, int *z* = 0) const

Return pixel value at given position.

```
940 {
941     return _pointee ? _pointee->getAt(x, y, z) : HxValue(HxScalarInt(0));
942 }
```

6.42.4.38 STD_OSTREAM & HxImageRep::printInfo (STD_OSTREAM & *os*, int *doData* = 0)

Print information.

```
946 {
947     return _pointee ? pointee()->printInfo(os, doData) : os ;
948 }
```

6.42.4.39 void HxImageRep::getRgbPixels2d (int * *pixels*, HxString *displayMode*, int *resWidth* = -1, int *resHeight* = -1, HxGeoIntType *gi* = NEAREST) const

The getRgbPixels functions fill an externally allocated buffer(pixels) with pixelvalues of the (2d) image.

The values are stored in the buffer according to the format used in Java. Conversion of pixel values in the image to the Java format is done via a HxRgbFunctor (a function object). By default, the size of the pixels buffer matches the image sizes. However, the user may request pixels at a different resolution by giving a resWidth and resHeight. Then, for each pixel in the buffer the value is obtained from the relative position (resWidth is mapped onto the image width, etc.). The value at the relative position is obtained through the given geometric interpolation type gi (and passed to the RgbFunctor). See also: \Ref{Color conversion/display}

```
985 {
986     if (!pointee())
987         return;
988 
989     HxTagList tags;
990     HxAddTag(tags, "pixels", pixels);
991     HxAddTag(tags, "resWidth", resWidth);
992     HxAddTag(tags, "resHeight", resHeight);
993     HxAddTag(tags, "gi", gi);
994 
995     if (displayMode == HxString("LogMagnitude")) {
996         HxImageRep n2 = HxNorm2(*this);
997         double lowVal = ((HxScalarDouble)n2.reduceOp("minAssign")).x();
998         double highVal = ((HxScalarDouble)n2.reduceOp("maxAssign")).x();
999         HxAddTag(tags, "lowVal", lowVal);
1000          HxAddTag(tags, "highVal", highVal);
1001 
1002          n2.pointee()->rgbOp(displayMode, tags);
1003          return;
1004     }
1005 
1006     if (displayMode == HxString("Stretch")) {
1007         double lowVal = ((HxVec3Double) reduceOp("minAssign")).min().x();
1008         double highVal = ((HxVec3Double) reduceOp("maxAssign")).max().x();
1009         HxAddTag(tags, "lowVal", lowVal);
1010         HxAddTag(tags, "highVal", highVal);
1011     }
1012 
1013     pointee()->rgbOp(displayMode, tags);
1014 }
```

6.42.4.40 void HxImageRep::getRgbPixels2d (int * *pixels*, HxString *displayMode*, int *bufWidth*, int *bufHeight*, int *VX*, int *VY*, int *VW*, int *VH*, double *SX*, double *SY*, double *scaleX*, double *scaleY*, HxGeoIntType *gi*) const

Partial display for large images.

```
1020 {
1021     if (!pointee())
1022         return;
1023     pointee()->getRgbPixels2d(pixels, displayMode, bufWidth, bufHeight,
1024             VX, VY, VW, VH, SX, SY, scaleX, scaleY, gi);
1025 }
```

6.42.5 Friends And Related Function Documentation

6.42.5.1 HxImageRep L_HXIMAGEREP HxProjectRange (HxImageRep *im*, int *dimension*) [friend]

Projection of the pixel range.

The function computes the projection (see **Pixels** (p. 5)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 9)). Dimension starts at 1.

```
13 {
14     return im.projectRange(dimension);
15 }
```

6.42.5.2 HxImageRep L_HXIMAGEREP HxInverseProjectRange (HxImageRep *im*, int *dimension*, HxImageRep *arg*) [friend]

Inverse projection of the pixel range.

The function projects (see **Pixels** (p. 5)) all pixels of image im on the given dimension of image arg via a unary pixel operation (see **Images** (p. 9)). Dimension starts at 1.

```
13 {
14     return im.inverseProjectRange(dimension, arg);
15 }
```

6.42.5.3 HxImageRep L_HXIMAGEREP HxRestrict (HxImageRep *img*, HxPoint *begin*, HxPoint *end*) [friend]

Restriction of domain.

Restrict the domain of the image to the region specified by the given points. Points are treated as pixel coordinates (integers).

```
13 {
14     return img.restrict(begin, end);
15 }
```

6.42.5.4 HxImageRep L_HXIMAGEREP HxExtend (HxImageRep *img*, HxImageRep *background*, HxPoint *begin*) [friend]

Extension of domain.

Extend the domain of the image to the size of the background image. The image is put at the position indicated by begin. Points are treated as pixel coordinates (integers).

```
13 {
14     return img.extend(background, begin);
15 }
```

6.42.5.5 HxImageRep L_HXIMAGEREP HxExtendVal (HxImageRep *img*, HxSizes *newSize*, HxValue *background*, HxPoint *begin*) [friend]

Extension of domain.

Extend the domain of the image to the given size. The image is put at the position indicated by begin. Points are treated as pixel coordinates (integers).

```
14 {
15     return img.extend(newSize, background, begin);
16 }
```

6.42.5.6 HxImageRep L_HXIMAGEREP HxMakeFromSi (IMAGE * *im*) [friend]

Make an HxImageRep with from the given image in ScilImage format.

```
13 {
14     return HxImageRep(im);
15 }
```

6.42.5.7 L_HXIMAGEREP IMAGE* HxExportSi (HxImageRep *img*) [friend]

Export image data to a ScilImage image.

```
13 {
14     return img.toImageSi();
15 }
```

6.42.5.8 void L_HXIMAGEREP HxExportMatlabPixels (HxImageRep *img*, double * *pixels*) [friend]

Export image data to array of int.

The size of the (pre-allocated) array must be equal to the dimensionality of the pixel values times the number of pixels in the image.

```
13 {
14     img.getDoublePixels(pixels);
15 }
```

The documentation for this class was generated from the following files:

- **HxImageRep.h**
- HxImageRep.c

6.43 HxImageSeq Class Reference

Class definition for a sequence of **HxImageRep** (p. 313)'s.

`#include <HxImageSeq.h>`

Public Methods

- **HxImageSeq** ()

  Constructor.

- **HxImageSeq** (**HxString** filename, int bufSize=0)

  Construct sequence from given file.

- **HxImageSeq** (const HxImageSeq &rhs)

  Copy constructor.

- virtual ~**HxImageSeq** ()

  Destructor.

- HxImageSeq & **operator=** (const HxImageSeq &rhs)

  Assignment operator.

- int **ident** () const

  The identity of the sequence.

- int **isNull** () const

  Indicates wether this is a valid sequence.

- int **frameWidth** () const

  The frame width.

- int **frameHeight** () const

  The frame height.

- int **frameDepth** () const

  The frame depth.

- int **nrFrames** () const

  The number of frames.

- **HxImageRep getFrame** (int nr) const

  Get the specified frame.

- void **getRgb2d** (int nr, int *pixels, **HxString** displayMode) const

  Deprecated?

- void **getRgbPixels2d** (int nr, int *pixels, **HxString** displayMode, int resWidth=-1, int resHeight=-1, **HxGeoIntType** gi=NEAREST) const
- **HxImageSeqIter begin** ()

  An iterator pointing to the first frame.

- **HxImageSeqIter end** ()

  An iterator pointing beyond the last frame.

- int **writeFile** (std::vector< int > frameList, **HxString** filename, int format)

Write the frame from the list to the given file in the given format.

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Put some information on the given stream.

Static Public Methods

- HxImageSeq **constructMpegSoft** (**HxString** fileName, int bufSize=0, **HxString** indexFilename="")

  Construct a sequence from the given mpeg file using the software decoder.

Static Public Attributes

- const int **MPEG_F** = 1
- const int **MIR_F** = 2
- const int **AVI_F** = 3

Friends

- class **HxImageSeqIter**

6.43.1 Detailed Description

Class definition for a sequence of **HxImageRep** (p. 313)'s.

Supported types:

- Mpeg video: Both Mpeg 1 and 2 are supported.
- MIR video: A simple uncompressed format. A sequence of images (**HxImageRep** (p. 313)) are stacked in a MIR file. MIR stands for Motion (Hx)ImageRep.
- AVI video: Standard video for windows format.

We use the file extension to recognize the video format:

- .mir -> MIR format
- .avi -> AVI format
- default -> Mpeg format

6.43.2 Constructor & Destructor Documentation

6.43.2.1 HxImageSeq::HxImageSeq ()

Constructor.

```
29                          : _pointee(0)
30 {
31 }
```

6.43.2.2 HxImageSeq::HxImageSeq (HxString *filename*, int *bufSize* = 0)

Construct sequence from given file.

bufSize: the size of the internal buffer. The buffer stores frames converted to **HxImageRep** (p. 313)'s for later references. In the current implementation only contiguous frames are kept. The default is no buffering.

```
33                                                      : _pointee(0)
34 {
35 #ifndef __GNUC__
36     int pos = filename.rfind(".");
37     if (pos >=0 ) {
38         HxString ext = filename.substr(pos);
39         for (HxString::iterator i = ext.begin(); i!= ext.end(); i++)
40             *i = toupper(*i);
41         if (ext.compare(".MIR")==0)
42             _pointee = new HxImageSeqMir(filename, bufSize);
43         else {
44             HxImageSeqDXMedia* seq = new HxImageSeqDXMedia(filename, bufSize);
45             if(seq->valid())
46                 _pointee = seq;
47             else
48                 delete seq;
49         }
50     }else {
51         HxEnvironment::instance()->errorStream()
52             << "No extension found in " << filename << STD_ENDL;
53         HxEnvironment::instance()->flush();
54     }
55 #endif
56 }
```

6.43.2.3 HxImageSeq::HxImageSeq (const HxImageSeq & *rhs*)

Copy constructor.

```
69                                          : _pointee(rhs.pointee())
70 {
71 }
```

6.43.2.4 HxImageSeq::~HxImageSeq () [virtual]

Destructor.

```
74 {
75 }
```

6.43.3 Member Function Documentation

6.43.3.1 HxImageSeq HxImageSeq::constructMpegSoft (HxString *filename*, int *bufSize* = 0, HxString *indexFilename* = "") [static]

Construct a sequence from the given mpeg file using the software decoder.

indexFilename: an index file is required. If no index file is found at the specified location an index file will be created. Default: the index file is in the same directory as the video file.

```
61 {
62     HxImageSeq seq;
63 #ifndef __GNUC__
64     seq._pointee = new HxImageSeqMpeg(filename, bufSize, indexFilename);
65 #endif
66     return seq;
67 }
```

6.43.3.2 HxImageSeq & HxImageSeq::operator= (const HxImageSeq & *rhs*)

Assignment operator.

```
79 {
80     _pointee = rhs._pointee;
81     return *this;
82 }
```

6.43.3.3 int HxImageSeq::ident () const

The identity of the sequence.

```
92 {
93     return pointee() ? pointee()->ident() : 0;
94 }
```

6.43.3.4 int HxImageSeq::isNull () const

Indicates wether this is a valid sequence.

```
86 {
87     return !int(_pointee);
88 }
```

6.43.3.5 int HxImageSeq::frameWidth () const

The frame width.

```
98 {
99     return pointee() ? pointee()->frameWidth() : 0;
100 }
```

6.43.3.6 int HxImageSeq::frameHeight () const

The frame height.

```
104 {
105     return pointee() ? pointee()->frameHeight() : 0;
106 }
```

6.43.3.7 int HxImageSeq::frameDepth () const

The frame depth.

```
110 {
111     return pointee() ? pointee()->frameDepth() : 0;
112 }
```

6.43.3.8 int HxImageSeq::nrFrames () const

The number of frames.

```
116 {
117     return pointee() ? pointee()->nrFrames() : 0;
118 }
```

6.43.3.9 HxImageRep HxImageSeq::getFrame (int *nr*) const

Get the specified frame.

```
122 {
123     return pointee() ? pointee()->getFrame(nr) : HxImageRep();
124 }
```

6.43.3.10 void HxImageSeq::getRgb2d (int *nr*, int * *pixels*, HxString *displayMode*) const

Deprecated?

```
128 {
129     if (!pointee()) return;
130 
131     pointee()->getRgb2d(nr, pixels, displayMode);
132 }
```

6.43.3.11 HxImageSeqIter HxImageSeq::begin ()

An iterator pointing to the first frame.

```
145 {
146     return HxImageSeqIter(this, 0);
147 }
```

6.43.3.12 HxImageSeqIter HxImageSeq::end ()

An iterator pointing beyond the last frame.

```
151 {
152     return HxImageSeqIter(this, pointee()->nrFrames());
153 }
```

6.43.3.13 int HxImageSeq::writeFile (std::vector< int > *frameList*, HxString *filename*, int *format*)

Write the frame from the list to the given file in the given format.

Currently, only MIR format is implemented.

```
157 {
158
159 #ifndef __GNUC__
160
161     if (format==MIR_F) {
162         Header_Type* header = new Header_Type;
163         strcpy(header->id,"MIR");
164         header->width = frameWidth();
165         header->height = frameHeight();
166         header->depth = frameDepth();
167         header->no_of_frames = frameList.size();
168
169         FILE *fp = fopen(filename.c_str(),"w+b");
170         if (!fp) {
171             HxEnvironment::instance()->errorStream()
172                 << "Cannot open file for writing:" << filename << STD_ENDL;
173             HxEnvironment::instance()->flush();
174             return -1;
175         }
176
177         int* buf = new int[header->width * header->height];
178         int block_size =  header->width * header->height * sizeof(int);
179
180         fwrite(header,sizeof(Header_Type),1,fp);
181
182         for (int i=0; i<frameList.size(); i++) {
183             HxImageRep im = pointee()->frame2HxImageRep(frameList[i]);
184             im.getRgbPixels2d(buf,"Direct");
185             fwrite(buf, block_size, 1, fp);
186         }
187         fclose(fp);
188
189         delete buf;
190
191         return 0;
192     }
193     else {
194         HxEnvironment::instance()->errorStream()
195             << "Writefile for this format not implemented yet" << STD_ENDL;
196         HxEnvironment::instance()->flush();
197         return -1;
198     }
199
200 #else
201
202     HxEnvironment::instance()->errorStream()
203     << "No non DX codec" << STD_ENDL;
204     HxEnvironment::instance()->flush();
205     return -1;
206
207 // __GNUC__
208 #endif
209 }
```

6.43.3.14 STD_OSTREAM & HxImageSeq::put (STD_OSTREAM & *os*) const

Put some information on the given stream.

```
213 {
214     os << "Number of Frames: " << nrFrames()
215        << ", frame sizes : " << frameWidth() << " x " << frameHeight()
216        << STD_ENDL;
217     return os;
218 }
```

The documentation for this class was generated from the following files:

- **HxImageSeq.h**
- HxImageSeq.c

6.44 HxImageSeqIter Class Reference

Class definition for an iterator over an **HxImageSeq** (p. 333).

`#include <HxImageSeqIter.h>`

Public Methods

- **HxImageSeqIter** ()

  Constructor.

- **HxImageSeqIter** (**HxImageSeq** *hisf, int framenum)

  Constructor for random access.

- **HxImageSeqIter** (const HxImageSeqIter &rhs)

  Copy constrcutor.

- virtual ~**HxImageSeqIter** ()

  Destructor.

- HxImageSeqIter & **operator=** (const HxImageSeqIter &rhs)

  Assignment.

- HxImageSeqIter & **operator++** ()

  Increment (prefix).

- HxImageSeqIter & **operator++** (int)

  Increment (postfix).

- HxImageSeqIter & **operator–** ()

  Decrement (prefix).

- HxImageSeqIter & **operator–** (int)

  Decrement (postfix).

- HxImageSeqIter & **operator+=** (int)
  Increment with value (may be negative).
- **HxImageRep operator** ∗ ()
  Dereferencing: get the current frame.
- HxImageSeqIter ∗ **clone** () const
  Make a copy.
- bool **operator==** (const HxImageSeqIter &)
  Equal.
- bool **operator!=** (const HxImageSeqIter &)
  Not equal.

6.44.1 Detailed Description

Class definition for an iterator over an **HxImageSeq** (p. 333).

6.44.2 Constructor & Destructor Documentation

6.44.2.1 HxImageSeqIter::HxImageSeqIter ()

Constructor.

```
18                                  : _sequence(0), _framenum(0)
19 {
20 }
```

6.44.2.2 HxImageSeqIter::HxImageSeqIter (HxImageSeq ∗ *hisf*, int *framenum*)

Constructor for random access.

```
23     : _sequence(hisf->pointee())
24 {
25     _framenum = hisf->nrFrames();
26
27     if ((framenum >= 0) && (framenum < _framenum))
28         _framenum = framenum;
29
30 }
```

6.44.2.3 HxImageSeqIter::HxImageSeqIter (const HxImageSeqIter & *rhs*)

Copy constrcutor.

```
33     : _sequence(rhs._sequence), _framenum(rhs._framenum)
34 {
35 }
```

6.44.2.4 HxImageSeqIter::~HxImageSeqIter () [virtual]

Destructor.

```
38 {
39 }
```

6.44.3 Member Function Documentation

6.44.3.1 HxImageSeqIter & HxImageSeqIter::operator= (const HxImageSeqIter & *rhs*)

Assignment.

```
43 {
44      _sequence = rhs._sequence;
45      _framenum = rhs._framenum;
46      return *this;
47 }
```

6.44.3.2 HxImageSeqIter & HxImageSeqIter::operator++ ()

Increment (prefix).

```
51 {
52     _framenum++;
53     if (_framenum > _sequence->nrFrames())
54         _framenum = _sequence->nrFrames();
55     return *this;
56 }
```

6.44.3.3 HxImageSeqIter & HxImageSeqIter::operator++ (int)

Increment (postfix).

```
60 {
61     _framenum++;
62     if (_framenum > _sequence->nrFrames())
63         _framenum = _sequence->nrFrames();
64     return *this;
65 }
```

6.44.3.4 HxImageSeqIter & HxImageSeqIter::operator– ()

Decrement (prefix).

```
69 {
70     _framenum--;
71     if (_framenum < 0)
72         _framenum = 0;
73     return *this;
74 }
```

6.44.3.5 HxImageSeqIter & HxImageSeqIter::operator– (int)

Decrement (postfix).

```
78 {
79     _framenum--;
80     if (_framenum < 0)
81         _framenum = 0;
82     return *this;
83 }
```

6.44.3.6 HxImageSeqIter & HxImageSeqIter::operator+= (int *t*)

Increment with value (may be negative).

```
87 {
88     _framenum += t;
89     if (_framenum > _sequence->nrFrames())
90         _framenum = _sequence->nrFrames();
91     if (_framenum < 0)
92         _framenum = 0;
93     return *this;
94 }
```

6.44.3.7 HxImageRep HxImageSeqIter::operator ∗ ()

Dereferencing: get the current frame.

```
98 {
99     return _sequence->getFrame(_framenum);
100 }
```

6.44.3.8 HxImageSeqIter ∗ HxImageSeqIter::clone () const

Make a copy.

```
104 {
105     return new HxImageSeqIter (*this);
106 }
```

6.44.3.9 bool HxImageSeqIter::operator== (const HxImageSeqIter & *x*)

Equal.

```
110 {
111     return _framenum == (x._framenum);
112 }
```

6.44.3.10 bool HxImageSeqIter::operator!= (const HxImageSeqIter & *x*)

Not equal.

```
116 {
117     return _framenum != (x._framenum);
118 }
```

The documentation for this class was generated from the following files:

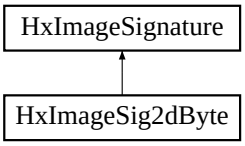
- **HxImageSeqIter.h**
- HxImageSeqIter.c

6.45 HxImageSig2dByte Class Reference

Signature for a 2D image with pixels represented by an HxByte.

`#include <HxImageSig2dByte.h>`

Inheritance diagram for HxImageSig2dByte::



Public Types

- typedef HxByte **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr2dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dByte **ProjectDomainImageSigType**
- typedef **HxImageSig2dInt ArithImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig2dByte** ()

6.45.1 Detailed Description

Signature for a 2D image with pixels represented by an HxByte.

The documentation for this class was generated from the following file:

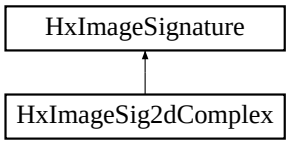
- **HxImageSig2dByte.h**

6.46 HxImageSig2dComplex Class Reference

Signature for a 2D image with pixels represented by an **HxComplex** (p. 239).

`#include <HxImageSig2dComplex.h>`

Inheritance diagram for HxImageSig2dComplex::



Public Types

- typedef **HxComplex PixelType**
- typedef **HxComplex ArithType**
- typedef **HxComplex ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dComplex **ProjectDomainImageSigType**
- typedef HxImageSig2dComplex **ArithImageSigType**
- typedef HxImageSig2dComplex **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, COMPLEX_VALUE, sizeof(double) << 3) }

Public Methods

- **HxImageSig2dComplex** ()

6.46.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxComplex** (p. 239).

The documentation for this class was generated from the following file:

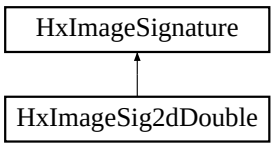
- **HxImageSig2dComplex.h**

6.47 HxImageSig2dDouble Class Reference

Signature for a 2D image with pixels represented by a double.

`#include <HxImageSig2dDouble.h>`

Inheritance diagram for HxImageSig2dDouble::



Public Types

- typedef double **PixelType**
- typedef **HxScalarDouble ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr2dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dDouble **ProjectRangeImageSigType**
- typedef HxImageSig2dDouble **ProjectDomainImageSigType**
- typedef HxImageSig2dDouble **ArithImageSigType**
- typedef HxImageSig2dDouble **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, REAL_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig2dDouble** ()

6.47.1 Detailed Description

Signature for a 2D image with pixels represented by a double.

The documentation for this class was generated from the following file:

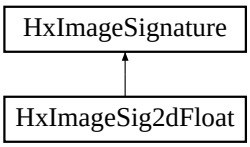
- **HxImageSig2dDouble.h**

6.48 HxImageSig2dFloat Class Reference

Signature for a 2D image with pixels represented by a float.

`#include <HxImageSig2dFloat.h>`

Inheritance diagram for HxImageSig2dFloat::



Public Types

- typedef HxDataPtr2dFloat **DataPtrType**
- typedef float **PixelType**
- typedef **HxScalarDouble ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxImageSig2dFloat **ProjectRangeImageSigType**
- typedef HxImageSig2dFloat **ProjectDomainImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigTypeDouble**
- typedef HxPixelAllocator< PixelType > **Allocator**
- enum { **ID** = SIG_ID(2, 1, REAL_VALUE, sizeof(float) << 3) }

Public Methods

- **HxImageSig2dFloat** ()

6.48.1 Detailed Description

Signature for a 2D image with pixels represented by a float.

The documentation for this class was generated from the following file:

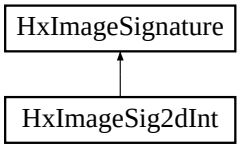
- **HxImageSig2dFloat.h**

6.49 HxImageSig2dInt Class Reference

Signature for a 2D image with pixels represented by an int.

`#include <HxImageSig2dInt.h>`

Inheritance diagram for HxImageSig2dInt::



Public Types

- typedef int **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr2dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dInt **ProjectDomainImageSigType**
- typedef HxImageSig2dInt **ArithImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig2dInt** ()

6.49.1 Detailed Description

Signature for a 2D image with pixels represented by an int.

The documentation for this class was generated from the following file:

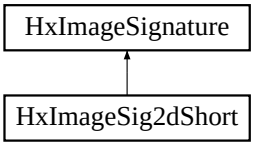
- **HxImageSig2dInt.h**

6.50 HxImageSig2dShort Class Reference

Signature for a 2D image with pixels represented by a short.

`#include <HxImageSig2dShort.h>`

Inheritance diagram for HxImageSig2dShort::



Public Types

- typedef short **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr2dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dShort **ProjectDomainImageSigType**
- typedef **HxImageSig2dInt ArithImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig2dShort** ()

6.50.1 Detailed Description

Signature for a 2D image with pixels represented by a short.

The documentation for this class was generated from the following file:

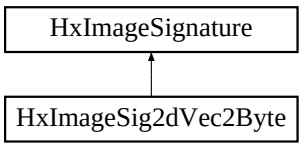
- **HxImageSig2dShort.h**

6.51 HxImageSig2dVec2Byte Class Reference

Signature for a 2D image with pixels represented by an HxVec2Byte.

`#include <HxImageSig2dVec2Byte.h>`

Inheritance diagram for HxImageSig2dVec2Byte::



Public Types

- typedef HxVec2Byte **PixelType**
- typedef **HxVec2Int ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec2Byte **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec2Int ArithImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, INT_VALUE, sizeof(char) << 3) }

Public Methods

- **HxImageSig2dVec2Byte** ()

6.51.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec2Byte.

The documentation for this class was generated from the following file:

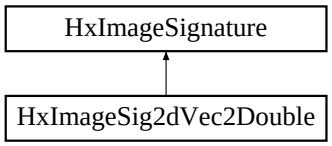
- **HxImageSig2dVec2Byte.h**

6.52 HxImageSig2dVec2Double Class Reference

Signature for a 2D image with pixels represented by an **HxVec2Double** (p. 766).

`#include <HxImageSig2dVec2Double.h>`

Inheritance diagram for HxImageSig2dVec2Double::



Public Types

- typedef **HxVec2Double PixelType**
- typedef **HxVec2Double ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec2Double **ProjectDomainImageSigType**
- typedef HxImageSig2dVec2Double **ArithImageSigType**
- typedef HxImageSig2dVec2Double **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, REAL_VALUE, sizeof(double) << 3) }

Public Methods

- **HxImageSig2dVec2Double** ()

6.52.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec2Double** (p. 766).

The documentation for this class was generated from the following file:

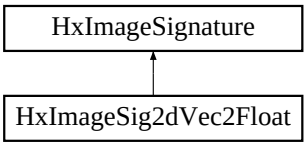
- **HxImageSig2dVec2Double.h**

6.53 HxImageSig2dVec2Float Class Reference

Signature for a 2D image with pixels represented by an HxVec2Float.

`#include <HxImageSig2dVec2Float.h>`

Inheritance diagram for HxImageSig2dVec2Float::



Public Types

- typedef HxVec2Float **PixelType**
- typedef **HxVec2Double ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec2Float **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, REAL_VALUE, sizeof(float) << 3) }

Public Methods

- **HxImageSig2dVec2Float** ()

6.53.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec2Float.

The documentation for this class was generated from the following file:

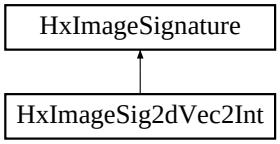
- **HxImageSig2dVec2Float.h**

6.54 HxImageSig2dVec2Int Class Reference

Signature for a 2D image with pixels represented by an **HxVec2Int** (p. 785).

`#include <HxImageSig2dVec2Int.h>`

Inheritance diagram for HxImageSig2dVec2Int::



Public Types

- typedef **HxVec2Int PixelType**
- typedef **HxVec2Int ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxImageSig2dVec2Int **ProjectDomainImageSigType**
- typedef HxImageSig2dVec2Int **ArithImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigTypeDouble**
- typedef HxPixelAllocator< PixelType > **Allocator**
- enum { **ID** = SIG_ID(2, 2, INT_VALUE, sizeof(int) << 3) }

Public Methods

- **HxImageSig2dVec2Int** ()

6.54.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec2Int** (p. 785).

The documentation for this class was generated from the following file:

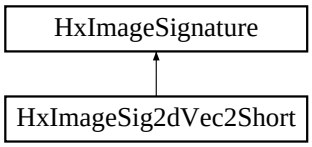
- **HxImageSig2dVec2Int.h**

6.55 HxImageSig2dVec2Short Class Reference

Signature for a 2D image with pixels represented by an HxVec2Short.

`#include <HxImageSig2dVec2Short.h>`

Inheritance diagram for HxImageSig2dVec2Short::



Public Types

- typedef HxVec2Short **PixelType**
- typedef **HxVec2Int ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec2Short **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec2Int ArithImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, INT_VALUE, sizeof(short) << 3) }

Public Methods

- **HxImageSig2dVec2Short** ()

6.55.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec2Short.

The documentation for this class was generated from the following file:

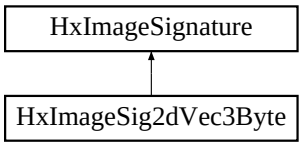
- **HxImageSig2dVec2Short.h**

6.56 HxImageSig2dVec3Byte Class Reference

Signature for a 2D image with pixels represented by an HxVec3Byte.

`#include <HxImageSig2dVec3Byte.h>`

Inheritance diagram for HxImageSig2dVec3Byte::



Public Types

- typedef HxVec3Byte **PixelType**
- typedef **HxVec3Int ArithType**
- typedef **HxVec3Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec3Byte **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec3Int ArithImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 3, INT_VALUE, sizeof(char) << 3) }

Public Methods

- **HxImageSig2dVec3Byte** ()

6.56.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec3Byte.

The documentation for this class was generated from the following file:

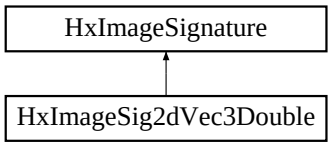
- **HxImageSig2dVec3Byte.h**

6.57 HxImageSig2dVec3Double Class Reference

Signature for a 2D image with pixels represented by an **HxVec3Double** (p. 804).

`#include <HxImageSig2dVec3Double.h>`

Inheritance diagram for HxImageSig2dVec3Double::



Public Types

- typedef **HxVec3Double PixelType**
- typedef **HxVec3Double ArithType**
- typedef **HxVec3Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec3Double **ProjectDomainImageSigType**
- typedef HxImageSig2dVec3Double **ArithImageSigType**
- typedef HxImageSig2dVec3Double **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 3, REAL_VALUE, sizeof(double) << 3) }

Public Methods

- **HxImageSig2dVec3Double** ()

6.57.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec3Double** (p. 804).

The documentation for this class was generated from the following file:

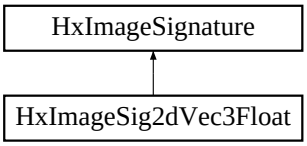
- **HxImageSig2dVec3Double.h**

6.58 HxImageSig2dVec3Float Class Reference

Signature for a 2D image with pixels represented by an HxVec3Float.

`#include <HxImageSig2dVec3Float.h>`

Inheritance diagram for HxImageSig2dVec3Float::



Public Types

- typedef HxVec3Float **PixelType**
- typedef **HxVec3Double ArithType**
- typedef **HxVec3Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec3Float **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 3, REAL_VALUE, sizeof(float) << 3) }

Public Methods

- **HxImageSig2dVec3Float** ()

6.58.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec3Float.

The documentation for this class was generated from the following file:

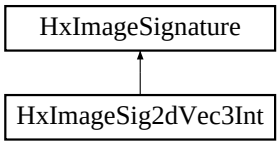
- **HxImageSig2dVec3Float.h**

6.59 HxImageSig2dVec3Int Class Reference

Signature for a 2D image with pixels represented by an **HxVec3Int** (p. 824).

`#include <HxImageSig2dVec3Int.h>`

Inheritance diagram for HxImageSig2dVec3Int::



Public Types

- typedef **HxVec3Int PixelType**
- typedef **HxVec3Int ArithType**
- typedef **HxVec3Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef HxImageSig2dVec3Int **ProjectDomainImageSigType**
- typedef HxImageSig2dVec3Int **ArithImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 3, INT_VALUE, sizeof(int) << 3) }

Public Methods

- **HxImageSig2dVec3Int** ()

6.59.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec3Int** (p. 824).

The documentation for this class was generated from the following file:

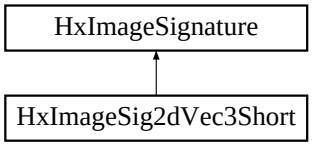
- **HxImageSig2dVec3Int.h**

6.60 HxImageSig2dVec3Short Class Reference

Signature for a 2D image with pixels represented by an HxVec3Short.

`#include <HxImageSig2dVec3Short.h>`

Inheritance diagram for HxImageSig2dVec3Short::



Public Types

- typedef HxVec3Short **PixelType**
- typedef **HxVec3Int ArithType**
- typedef **HxVec3Double ArithTypeDouble**
- typedef HxDataPtr2dTem< PixelType, ArithType > **DataPtrType**
- typedef HxImageSig2dVec3Short **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec3Int ArithImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigTypeDouble**
- typedef HxPixelAllocator< PixelType > **Allocator**
- enum { **ID** = SIG_ID(2, 3, INT_VALUE, sizeof(short) << 3) }

Public Methods

- **HxImageSig2dVec3Short** ()

6.60.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec3Short.

The documentation for this class was generated from the following file:

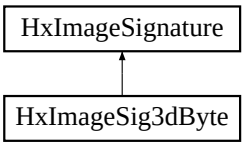
- **HxImageSig2dVec3Short.h**

6.61 HxImageSig3dByte Class Reference

Signature for a 3D image with pixels represented by an HxByte.

`#include <HxImageSig3dByte.h>`

Inheritance diagram for HxImageSig3dByte::



Public Types

- typedef HxByte **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr3dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef **HxImageSig2dByte ProjectDomainImageSigType**
- typedef **HxImageSig3dInt ArithImageSigType**
- typedef **HxImageSig3dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(3, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig3dByte** ()

6.61.1 Detailed Description

Signature for a 3D image with pixels represented by an HxByte.

The documentation for this class was generated from the following file:

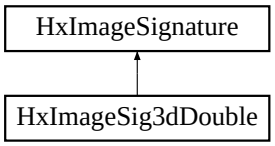
- **HxImageSig3dByte.h**

6.62 HxImageSig3dDouble Class Reference

Signature for a 3D image with pixels represented by a double.

`#include <HxImageSig3dDouble.h>`

Inheritance diagram for HxImageSig3dDouble::



Public Types

- typedef double **PixelType**
- typedef **HxScalarDouble ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr3dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef **HxImageSig2dDouble ProjectDomainImageSigType**
- typedef HxImageSig3dDouble **ArithImageSigType**
- typedef HxImageSig3dDouble **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(3, 1, REAL_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig3dDouble** ()

6.62.1 Detailed Description

Signature for a 3D image with pixels represented by a double.

The documentation for this class was generated from the following file:

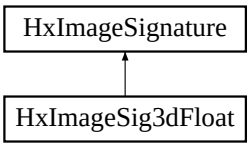
- **HxImageSig3dDouble.h**

6.63 HxImageSig3dFloat Class Reference

Signature for a 3D image with pixels represented by a float.

`#include <HxImageSig3dFloat.h>`

Inheritance diagram for HxImageSig3dFloat::



Public Types

- typedef float **PixelType**
- typedef **HxScalarDouble ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr3dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef **HxImageSig2dFloat ProjectDomainImageSigType**
- typedef **HxImageSig3dDouble ArithImageSigType**
- typedef **HxImageSig3dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(3, 1, REAL_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig3dFloat** ()

6.63.1 Detailed Description

Signature for a 3D image with pixels represented by a float.

The documentation for this class was generated from the following file:

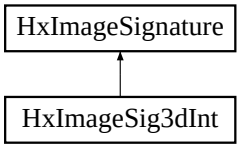
- **HxImageSig3dFloat.h**

6.64 HxImageSig3dInt Class Reference

Signature for a 3D image with pixels represented by an int.

`#include <HxImageSig3dInt.h>`

Inheritance diagram for HxImageSig3dInt::



Public Types

- typedef int **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr3dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef **HxImageSig2dInt ProjectDomainImageSigType**
- typedef HxImageSig3dInt **ArithImageSigType**
- typedef **HxImageSig3dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(3, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig3dInt** ()

6.64.1 Detailed Description

Signature for a 3D image with pixels represented by an int.

The documentation for this class was generated from the following file:

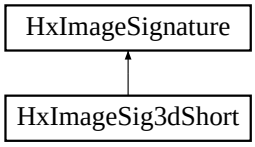
- **HxImageSig3dInt.h**

6.65 HxImageSig3dShort Class Reference

Signature for a 3D image with pixels represented by a short.

`#include <HxImageSig3dShort.h>`

Inheritance diagram for HxImageSig3dShort::



Public Types

- typedef short **PixelType**
- typedef **HxScalarInt ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef HxDataPtr3dScalarTem< PixelType, ArithType > **DataPtrType**
- typedef HxPixelAllocator< PixelType > **Allocator**
- typedef **HxImageSig2dShort ProjectDomainImageSigType**
- typedef **HxImageSig3dInt ArithImageSigType**
- typedef **HxImageSig3dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(3, 1, INT_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig3dShort** ()

6.65.1 Detailed Description

Signature for a 3D image with pixels represented by a short.

The documentation for this class was generated from the following file:

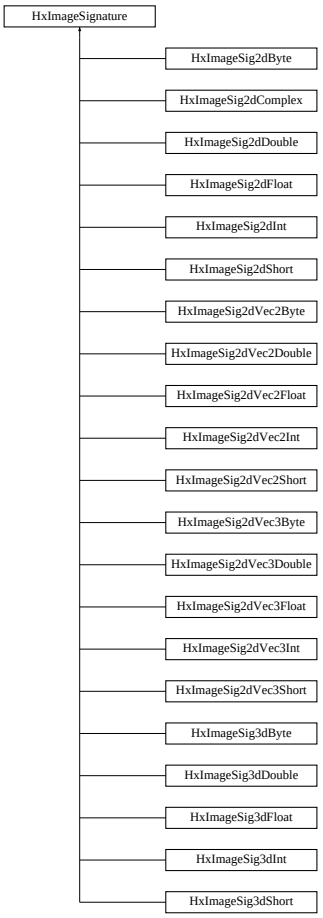
- **HxImageSig3dShort.h**

6.66 HxImageSignature Class Reference

Class definition image signature.

`#include <HxImageSignature.h>`

Inheritance diagram for HxImageSignature::



Constructors

- **HxImageSignature** ()

  Default constructor.

- **HxImageSignature** (const HxImageSignature &)

  Copy constructor.

- **HxImageSignature** (int imageDimensionality, int pixelDimensionality, **HxValueType** pixelType, int pixelPrecision)

  Construct signature with given description.

Operators

- HxImageSignature & **operator=** (const HxImageSignature &)
  Assignment operator.
- bool **isEqual** (const HxImageSignature &) const
  Equality for signatures.
- HxImageSignature **broadest** (const HxImageSignature &) const
  Broadest signature.
- bool **operator==** (const HxImageSignature &) const
  Equal.
- int **operator!=** (const HxImageSignature &) const
  Not equal.
- bool **operator<** (const HxImageSignature &) const
  Smaller.

Inquiry

- int **imageDimensionality** () const
  The dimensionality of the image (1, 2, or 3).
- int **pixelDimensionality** () const
  The dimensionality of the pixel values in the image (1: scalar value, 2: vector of 2 scalars, 3: vector of 3 scalars).
- **HxValueType pixelType** () const
  The type of the pixel values.
- int **pixelPrecision** () const
  The number of bits used in the representation of a pixel value (8, 16, 32, or 64).

Modification

- void **setImageDimensionality** (int)
  Set the image dimensionality.
- void **setPixelDimensionality** (int)
  Set the dimensionality of the pixel values.
- void **setPixelType** (**HxValueType**)
  Set the type of the pixel values.
- void **setPixelPrecision** (int)
  Set the number of bits used in the representation.

Output

- virtual STD_OSTREAM & **put** (STD_OSTREAM &) const
  Print value on stream.
- virtual **HxString toString** () const
  Value as a string.
- HxImageSignature **NameToSignature** (**HxString** name)
  Make signature from name.

Public Methods

- int **ident** () const

Protected Attributes

- int **_imageDimensionality**
- int **_pixelDimensionality**
- **HxValueType _pixelType**
- int **_pixelPrecision**

6.66.1 Detailed Description

Class definition image signature.

The signature gives a description of the characteristics of an image type (class).

6.66.2 Constructor & Destructor Documentation

6.66.2.1 HxImageSignature::HxImageSignature () [inline]

Default constructor.

```
203 {
204     _imageDimensionality = 2;
205     _pixelDimensionality = 1;
206     _pixelType = INT_VALUE;
207     _pixelPrecision = sizeof(short) << 3;
208 }
```

6.66.2.2 HxImageSignature::HxImageSignature (const HxImageSignature & *rhs*) [inline]

Copy constructor.

```
222     :   _imageDimensionality(rhs._imageDimensionality),
223         _pixelDimensionality(rhs._pixelDimensionality),
224         _pixelType(rhs._pixelType),
225         _pixelPrecision(rhs._pixelPrecision)
226 {
227 }
```

6.66.2.3 HxImageSignature::HxImageSignature (int *imgDim*, int *pixDim*, HxValueType *pixType*, int *pixPrec*) [inline]

Construct signature with given description.

```
213 {
214     _imageDimensionality = imgDim;
215     _pixelDimensionality = pixDim;
216     _pixelType = pixType;
217     _pixelPrecision = pixPrec;
218 }
```

6.66.3 Member Function Documentation

6.66.3.1 HxImageSignature & HxImageSignature::operator= (const HxImageSignature & *rhs*) [inline]

Assignment operator.

```
232 {
233     _imageDimensionality = rhs._imageDimensionality;
234     _pixelDimensionality = rhs._pixelDimensionality;
235     _pixelType = rhs._pixelType;
236     _pixelPrecision = rhs._pixelPrecision;
237     return *this;
238 }
```

6.66.3.2 bool HxImageSignature::isEqual (const HxImageSignature & *sig2*) const

Equality for signatures.

Same as operator==

```
126 {
127     if (_imageDimensionality != sig2._imageDimensionality)
128         return false;
129     if (_pixelDimensionality != sig2._pixelDimensionality)
130         return false;
131     if (_pixelType != sig2._pixelType)
132         return false;
133     if (_pixelPrecision != sig2._pixelPrecision)
134         return false;
135     return true;
136 }
```

6.66.3.3 HxImageSignature HxImageSignature::broadest (const HxImageSignature & *sig2*) const

Broadest signature.

Defined as the ”piecewise max” where max of integer and real equals real. In case one of the arguments has a pixel value type integer and the broadest signature has a pixel type real then the pixel precision of the result is doubled before computing the max to compensate for the loss of precision that occurs when integers are represented as floating point values.

```
158 {
159     HxImageSignature m;
160 
161     m._imageDimensionality = std::max(  _imageDimensionality,
162                                         sig2._imageDimensionality);
163     m._pixelDimensionality = std::max(  _pixelDimensionality,
164                                         sig2._pixelDimensionality);
165     m._pixelType = std::max(    _pixelType,
166                                 sig2._pixelType);
167 
168     int p1 = _pixelPrecision;
169     int p2 = sig2._pixelPrecision;
170     if ((m._pixelType == REAL_VALUE) || (m._pixelType == COMPLEX_VALUE))
171     {
172         if (_pixelType == INT_VALUE)
173             p1 = ((p1 << 1) <= doubleSize) ? (p1 << 1) : p1;
174         if (sig2._pixelType == INT_VALUE)
175             p2 = ((p2 << 1) <= doubleSize) ? (p2 << 1) : p2;
176     }
177     m._pixelPrecision = std::max(p1, p2);
178 
179     return m;
180 }
```

6.66.3.4 bool HxImageSignature::operator== (const HxImageSignature & *rhs*) const [inline]

Equal.

```
290 {
291     return isEqual(rhs);
292 }
```

6.66.3.5 int HxImageSignature::operator!= (const HxImageSignature & *rhs*) const [inline]

Not equal.

```
296 {
297     return !isEqual(rhs);
298 }
```

6.66.3.6 bool HxImageSignature::operator< (const HxImageSignature & *sig2*) const

Smaller.

Only for sorting purposes!

```
140 {
141     if (_imageDimensionality < sig2._imageDimensionality)
142         return true;
143     if (_imageDimensionality > sig2._imageDimensionality)
144         return false;
145     if (_pixelDimensionality < sig2._pixelDimensionality)
146         return true;
147     if (_pixelDimensionality > sig2._pixelDimensionality)
148         return false;
149     if (_pixelType < sig2._pixelType)
```

```
150         return true;
151     if (_pixelType > sig2._pixelType)
152         return false;
153     return (_pixelPrecision < sig2._pixelPrecision);
154 }
```

6.66.3.7 int HxImageSignature::imageDimensionality () const [inline]

The dimensionality of the image (1, 2, or 3).

```
242 {
243     return _imageDimensionality;
244 }
```

6.66.3.8 int HxImageSignature::pixelDimensionality () const [inline]

The dimensionality of the pixel values in the image (1: scalar value, 2: vector of 2 scalars, 3: vector of 3 scalars).

```
248 {
249     return _pixelDimensionality;
250 }
```

6.66.3.9 HxValueType HxImageSignature::pixelType () const [inline]

The type of the pixel values.

```
254 {
255     return _pixelType;
256 }
```

6.66.3.10 int HxImageSignature::pixelPrecision () const [inline]

The number of bits used in the representation of a pixel value (8, 16, 32, or 64).

```
260 {
261     return _pixelPrecision;
262 }
```

6.66.3.11 void HxImageSignature::setImageDimensionality (int *i*) [inline]

Set the image dimensionality.

```
266 {
267     _imageDimensionality = i;
268 }
```

6.66.3.12 void HxImageSignature::setPixelDimensionality (int *i*) [inline]

Set the dimensionality of the pixel values.

```
272 {
273     _pixelDimensionality = i;
274 }
```

6.66.3.13 void HxImageSignature::setPixelType (HxValueType *t*) [inline]

Set the type of the pixel values.

```
278 {
279     _pixelType = t;
280 }
```

6.66.3.14 void HxImageSignature::setPixelPrecision (int *i*) [inline]

Set the number of bits used in the representation.

```
284 {
285     _pixelPrecision = i;
286 }
```

6.66.3.15 STD_OSTREAM & HxImageSignature::put (STD_OSTREAM & *os*) const [virtual]

Print value on stream.

For global operator<<

```
191 {
192     os << "Image" << imageDimensionality() << "d";
193     if ((pixelDimensionality() > 1) && (pixelType() != COMPLEX_VALUE))
194         os << "Vec" << pixelDimensionality();
195     switch (pixelType()) {
196     case INT_VALUE:
197         os << "Int";
198         break;
199     case REAL_VALUE:
200         os << "Real";
201         break;
202     case COMPLEX_VALUE:
203         os << "Complex";
204         break;
205     }
206     os << pixelPrecision();
207     return os;
208 }
```

6.66.3.16 HxString HxImageSignature::toString () const [virtual]

Value as a string.

```
212 {
213     HxString s("Image");
214     s += makeString(imageDimensionality()) + "d";
215     if ((pixelDimensionality() > 1) && (pixelType() != COMPLEX_VALUE))
216         s += "Vec" + makeString(pixelDimensionality());
217     switch (pixelType()) {
218     case INT_VALUE:
219         s += "Int";
220         break;
221     case REAL_VALUE:
222         s += "Real";
223         break;
224     case COMPLEX_VALUE:
225         s += "Complex";
226         break;
227     }
228     s += makeString(pixelPrecision());
229     return s;
230 }
```

6.66.3.17 HxImageSignature HxImageSignature::NameToSignature (HxString *name*) [static]

Make signature from name.

```
271 {
272     static bool init = SignatureMapInit();
273     SignatureMap::iterator p = signatureMap.find(name);
274     return (p != signatureMap.end()) ?
275             (*p).second :
276             HxImageSignature(2, 1, INT_VALUE, 8);
277 }
```

The documentation for this class was generated from the following files:

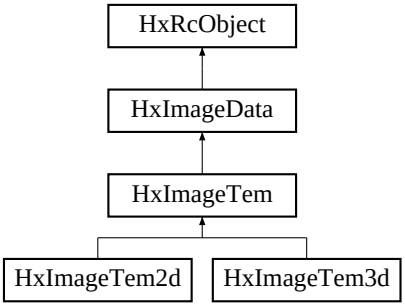
- **HxImageSignature.h**
- HxImageSignature.c

6.67 HxImageTem Class Template Reference

Template class for operations that are independent of image dimensionality.

`#include <HxImageTem.h>`

Inheritance diagram for HxImageTem::



Public Types

- typedef TYPENAME ImageSigT::ArithType **ArithType**
- typedef TYPENAME ImageSigT::ArithTypeDouble **ArithTypeDouble**
- typedef TYPENAME ImageSigT::DataPtrType **DataPtrType**
- typedef TYPENAME ImageSigT::ArithImageSigType **ArithImageSigType**
- typedef TYPENAME ImageSigT::ArithImageSigTypeDouble **ArithImageSigTypeDouble**

Public Methods

- **HxImageTem** ()
- **HxImageTem** (const HxImageTem &)
- virtual ~**HxImageTem** ()
- int **dimensionality** () const
- int **dimensionSize** (int i) const
- **HxSizes sizes** () const
- int **numberOfPixels** () const
- int **pixelDimensionality** () const
- **HxValueType pixelType** () const
- int **pixelPrecision** () const
- **HxImageSignature signature** () const
- virtual void **set** (double *pixels)
- virtual void **convertColor** (**HxVec3Double** scale1, **HxVec3Double** gamma1, HxUpoVec3Double step1, HxUpoVec3Double step2, HxUpoVec3Double step3, **HxVec3Double** gamma2, **HxVec3Double** scale2)
- virtual void **transpose** (const **HxImageData** *src)
- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)=0
- virtual void **setAt** (int x, int y, int z, const **HxValue** val)
- virtual **HxValue getAt** (int x, int y, int z) const
- virtual void **neighbourhoodOp** (const **HxImageData** *src, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation.

- virtual void **neighbourhoodOp** (const **HxImageData** *src, const **HxImageData** *kernel, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation with kernel.

- virtual void **getDoublePixels** (double *pixels)
- virtual STD_OSTREAM & **printInfo** (STD_OSTREAM &os, int doData=0) const

- virtual HxImageTem< ImageSigT > * **makeScratch** (**HxSizes** border) const
- virtual DataPtrType **dataPtrClone** () const=0

Protected Attributes

- int **_dimSizes** [3]

6.67.1 Detailed Description

template<class ImageSigT> class HxImageTem< ImageSigT >

Template class for operations that are independent of image dimensionality.

6.67.2 Member Function Documentation

6.67.2.1 template<class ImageSigT> void HxImageTem< ImageSigT >::neighbourhoodOp (const HxImageData * *src*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation.

Reimplemented from **HxImageData** (p. 296).

Reimplemented in **HxImageTem2d** (p. 371).

```
318 {
319     HxEnvironment::instance()->errorStream()
320         << "neighbourhoodOp(): operation not implemented for image type "
321         << signature() << STD_ENDL;
322     HxEnvironment::instance()->flush();
323 }
```

6.67.2.2 template<class ImageSigT> void HxImageTem< ImageSigT >::neighbourhoodOp (const HxImageData * *src*, const HxImageData * *kernel*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation with kernel.

Reimplemented from **HxImageData** (p. 296).

Reimplemented in **HxImageTem2d** (p. 371).

```
329 {
330     HxEnvironment::instance()->errorStream()
331         << "neighbourhoodOp(): operation not implemented for image type "
332         << signature() << STD_ENDL;
333     HxEnvironment::instance()->flush();
334 }
```

The documentation for this class was generated from the following files:

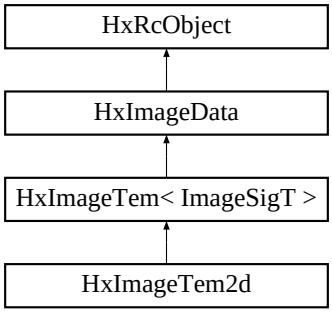
- **HxImageTem.h**
- HxImageTem.c

6.68 HxImageTem2d Class Template Reference

Template class for operations on 2D images.

`#include <HxImageTem2d.h>`

Inheritance diagram for HxImageTem2d::



Public Methods

- **HxImageTem2d** (int width=1, int height=1)
- **HxImageTem2d** (size_t *sizes)
- **HxImageTem2d** (const HxImageTem2d &)
- virtual ~**HxImageTem2d** ()
- int **width** () const
- int **height** () const
- virtual DataPtrType **dataPtrClone** () const
- virtual void **neighbourhoodOp** (const **HxImageData** *src, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation.

- virtual void **neighbourhoodOp** (const **HxImageData** *src, const **HxImageData** *kernel, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation with kernel.

- virtual void **geometricOp2d** (const **HxImageData** *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)
- virtual **HxImageData** * **projectDomain** (int dimension, int coordinate)
- virtual void **inverseProjectDomain** (int dimension, int coordinate, const **HxImageData** *arg)
- virtual void **transpose** (const **HxImageData** *src)
- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)
- virtual **HxValue sampleIdentMask** (const **HxImageData** *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc)
- virtual void **sampleIdentMask** (const **HxImageData** *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc, **HxValueListBackInserter** res)
- virtual **HxValue sampleWeightMask** (const **HxImageData** *mask, **HxPoint** p, **HxString** sFunc)
- virtual void **getRgbPixels2d** (int *pixels, **HxString** dispF, int bufWidth, int bufHeight, int VX, int VY, int VW, int VH, double SX, double SY, double scaleX, double scaleY, **HxGeoIntType** gi) const

6.68.1 Detailed Description

template<class ImageSigT> class HxImageTem2d< ImageSigT >

Template class for operations on 2D images.

6.68.2 Member Function Documentation

6.68.2.1 template<class ImageSigT> void HxImageTem2d< ImageSigT >::neighbourhoodOp (const HxImageData * *src*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation.

Reimplemented from **HxImageTem** (p. 369).

```
537 {
538     HxImgFtorNgbKey   funcKey(
539                           signature().toString(),
540                           src->signature().toString(), ngbName);
541
542     typedef HxImgFtorI2 FunctorType;
543
544     static HxImgFtorTableTem<FunctorType> funcTable;
545     FunctorType* func = funcTable.find(funcKey);
546
547     if (func) {
548         HxSizes borderSize = func->minimumBorderSize(tags);
549         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
550         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
551                                               src->signature(), scratchSize);
552         scratch->setPartImage(
553                         src, HxPointInt(0, 0, 0),
554                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
555         scratch->setBorder(borderSize, tags);
556         HxAddTag<HxSizes>(tags, "borderSize", borderSize);
557
558         func->callIt(this, scratch, tags);
559         delete scratch;
560     } else {
561         HxEnvironment::instance()->errorStream()
562             << "Can't find " << funcKey << STD_ENDL;
563         HxEnvironment::instance()->flush();
564     }
565 }
```

6.68.2.2 template<class ImageSigT> void HxImageTem2d< ImageSigT >::neighbourhoodOp (const HxImageData * *src*, const HxImageData * *kernel*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation with kernel.

Reimplemented from **HxImageTem** (p. 369).

```
572 {
573     HxImgFtorKernelNgbKey   funcKey(
574                                 signature().toString(),
575                                 src->signature().toString(),
576                                 kernel->signature().toString(),
```

```
577                                 ngbName);
578
579     typedef HxImgFtorI3 FunctorType;
580
581     static HxImgFtorTableTem<FunctorType> funcTable;
582     FunctorType* func = funcTable.find(funcKey);
583
584     if (func) {
585         HxAddTag(tags, "kernelSize", kernel->sizes());
586         HxSizes borderSize = func->minimumBorderSize(tags);
587         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
588         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
589                                                 src->signature(), scratchSize);
590         scratch->setPartImage(
591                         src, HxPointInt(0, 0, 0),
592                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
593         scratch->setBorder(borderSize, tags);
594         HxAddTag(tags, "borderSize", borderSize);
595
596         func->callIt(this, scratch, kernel, tags);
597         delete scratch;
598     } else {
599         HxEnvironment::instance()->errorStream()
600             << "Can't find " << funcKey << STD_ENDL;
601         HxEnvironment::instance()->flush();
602     }
603 }
```

The documentation for this class was generated from the following files:

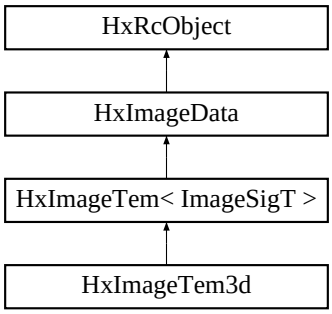
- **HxImageTem2d.h**
- HxImageTem2d.c

6.69 HxImageTem3d Class Template Reference

Template class for operations on 3D images.

`#include <HxImageTem3d.h>`

Inheritance diagram for HxImageTem3d::



Public Methods

- **HxImageTem3d** (int width=1, int height=1, int depth=1)
- **HxImageTem3d** (size_t *sizes)

- **HxImageTem3d** (const HxImageTem3d &)
- virtual ~**HxImageTem3d** ()
- int **width** () const
- int **height** () const
- int **depth** () const
- virtual DataPtrType **dataPtrClone** () const
- virtual void **neighbourhoodOp** (const **HxImageData** *kernel, **HxString** nfName)
- virtual void **geometricOp2d** (const **HxImageData** *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)
- virtual **HxImageData** * **projectDomain** (int dimension, int coordinate)
- virtual void **inverseProjectDomain** (int dimension, int coordinate, const **HxImageData** *arg)
- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)
- virtual **HxValue sampleIdentMask** (const **HxImageData** *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc)
- virtual void **sampleIdentMask** (const **HxImageData** *mask, **HxPoint** p, **HxSizes** size, int label, **HxString** sFunc, **HxValueListBackInserter** res)
- virtual **HxValue sampleWeightMask** (const **HxImageData** *mask, **HxPoint** p, **HxString** sFunc)

6.69.1 Detailed Description

template<class ImageSigT> class HxImageTem3d< ImageSigT >

Template class for operations on 3D images.

The documentation for this class was generated from the following files:

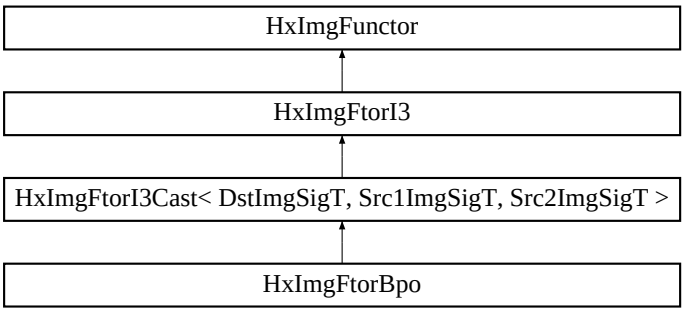
- **HxImageTem3d.h**
- HxImageTem3d.c

6.70 HxImgFtorBpo Class Template Reference

Instantiation of generic algorithm for binary pixel operations on images.

`#include <HxImgFtorBpo.h>`

Inheritance diagram for HxImgFtorBpo::



Public Types

- typedef **HxImgFtorBpoKey KeyType**

  The key type of this class.

- enum **RuleType** { **Src1IsKey**, **ArgsAreKey** }

Public Methods

- **HxImgFtorBpo** (RuleType ruleType=Src1IsKey)

  Constructor.

- virtual ~**HxImgFtorBpo** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **Src1DataPtrType** src1Ptr, **Src2DataPtrType** src2Ptr, **HxSizes** dstSize, **HxSizes** src1Size, **HxSizes** src2Size, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncBinaryPixOp** (p. 82) *to do the actual work.*

6.70.1 Detailed Description

template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> class HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >

Instantiation of generic algorithm for binary pixel operations on images.

6.70.2 Member Typedef Documentation

6.70.2.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> typedef HxImgFtorBpoKey HxImgFtorBpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.70.3 Constructor & Destructor Documentation

6.70.3.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::HxImgFtorBpo (RuleType *ruleType* = Src1IsKey) [inline]

Constructor.

```
24     : HxImgFtorI3Cast<DstImgSigT, Src1ImgSigT, Src2ImgSigT>(
25         HxImgFtorBpoKey(HxClassName<DstImgSigT>(), HxClassName<Src1ImgSigT>(),
```

```
26             HxClassName<Src2ImgSigT>(), HxClassName<BpoT>()))
27 {
28     switch (ruleType)
29     {
30     case Src1IsKey  :
31         HxImgFtorRuleBase::instance().setResultType(
32             HxClassName<DstImgSigT>(), "bpo",
33             HxClassName<Src1ImgSigT>(), HxClassName<BpoT>());
34         HxImgFtorRuleBase::instance().setArgumentType(
35             HxClassName<Src2ImgSigT>(), "bpo",
36             HxClassName<Src1ImgSigT>(), HxClassName<BpoT>());
37         break;
38     case ArgsAreKey :
39         HxImgFtorRuleBase::instance().setResultType(
40             HxClassName<DstImgSigT>(), "bpo",
41             HxClassName<Src1ImgSigT>(), HxClassName<Src2ImgSigT>(),
42             HxClassName<BpoT>());
43         break;
44     }
45 }
```

6.70.3.2 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::~HxImgFtorBpo () [virtual]

Destructor.

```
49 {
50 }
```

6.70.4 Member Function Documentation

6.70.4.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> void HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::doIt (DstDataPtrType *dstPtr*, Src1DataPtrType *src1Ptr*, Src2DataPtrType *src2Ptr*, HxSizes *dstSize*, HxSizes *src1Size*, HxSizes *src2Size*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, virtual]

Calls **HxFuncBinaryPixOp** (p. 82) to do the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 424).

```
58 {
59     BpoT bpo(tags);
60     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
61     HxFuncBinaryPixOp(
62         dstPtr, src1Ptr, src2Ptr, nPix, bpo);
63 }
```

The documentation for this class was generated from the following files:

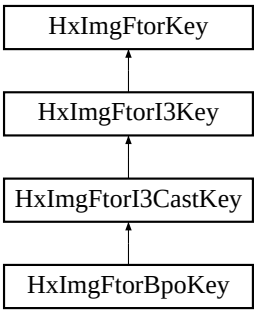
- **HxImgFtorBpo.h**
- HxImgFtorBpo.c

6.71 HxImgFtorBpoKey Class Reference

Key for **HxImgFtorBpo** (p. 373).

`#include <HxImgFtorBpoKey.h>`

Inheritance diagram for HxImgFtorBpoKey::



Public Methods

- **HxImgFtorBpoKey** (**HxString** dstImgSig, **HxString** src1ImgSig, **HxString** src2ImgSig, **HxString** bpoName)

  Constructor.

6.71.1 Detailed Description

Key for **HxImgFtorBpo** (p. 373).

6.71.2 Constructor & Destructor Documentation

6.71.2.1 HxImgFtorBpoKey::HxImgFtorBpoKey (HxString *dstImgSig*, HxString *src1ImgSig*, HxString *src2ImgSig*, HxString *bpoName*) [inline]

Constructor.

```
34     : HxImgFtorI3CastKey("HxImgFtorBpo", dstImgSig, src1ImgSig, src2ImgSig)
35 {
36     addArgument(bpoName);
37 }
```

The documentation for this class was generated from the following file:

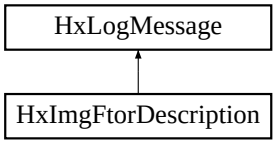
- **HxImgFtorBpoKey.h**

6.72 HxImgFtorDescription Class Reference

Image functor description.

`#include <HxImgFtorDescription.h>`

Inheritance diagram for HxImgFtorDescription::



Public Methods

- **HxImgFtorDescription** ()
- **HxImgFtorDescription** (const HxImgFtorDescription &)
- HxImgFtorDescription & **operator=** (const HxImgFtorDescription &)
- ~**HxImgFtorDescription** ()
- void **setKey** (const **HxImgFtorKey** &key)
- void **addArgument** (const **HxImageSignature** &, **HxSizes** size)
- void **setVariation** (**HxString**)
- void **setTags** (const **HxTagList** &tags)
- virtual **HxString toString** () const
- virtual STD_OSTREAM & **put** (STD_OSTREAM &) const
- bool **operator<** (const HxImgFtorDescription &) const
- bool **operator==** (const HxImgFtorDescription &) const
- bool **operator!=** (const HxImgFtorDescription &) const

6.72.1 Detailed Description

Image functor description.

The documentation for this class was generated from the following files:

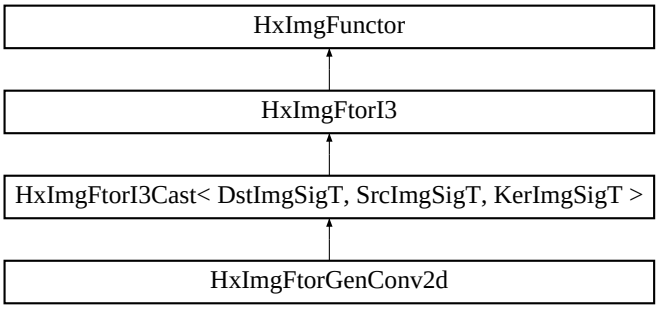
- **HxImgFtorDescription.h**
- HxImgFtorDescription.c

6.73 HxImgFtorGenConv2d Class Template Reference

Instantiation of generic algorithm for generalized convolution operations on 2d images.

`#include <HxImgFtorGenConv2d.h>`

Inheritance diagram for HxImgFtorGenConv2d::



Public Types

- typedef **HxImgFtorGenConvKey KeyType**

  The key type of this class.

- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source image.

- typedef KerImgSigT::DataPtrType **KerDataPtrType**

  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorGenConv2d** ()

  Constructor.

- virtual ~**HxImgFtorGenConv2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.73.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> class HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >

Instantiation of generic algorithm for generalized convolution operations on 2d images.

6.73.2 Member Typedef Documentation

6.73.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef HxImgFtorGenConvKey HxImgFtorGenConv2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.73.2.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef SrcImgSigT::DataPtrType HxImgFtorGenConv2d::SrcDataPtrType

The data pointer type of the source image.

6.73.2.3 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef KerImgSigT::DataPtrType HxImgFtorGenConv2d::KerDataPtrType

The data pointer type of the kernel image.

6.73.3 Constructor & Destructor Documentation

6.73.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv2d () [inline]

Constructor.

```
32         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
33             HxImgFtorGenConvKey(HxClassName<DstImgSigT>(),
34                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
35                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
36                 HxClassName<KernelT>()))
37 {
38 #ifdef CD_TRACE
39     HxEnvironment::instance()->outputStream()
40         << "HxImgFtorGenConv2d::HxImgFtorGenConv2d()" << STD_ENDL;
41 #endif
42 }
```

6.73.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv2d () [virtual]

Destructor.

```
48 {
49 #ifdef CD_TRACE
50     HxEnvironment::instance()->outputStream()
51         << "HxImgFtorGenConv2d::~HxImgFtorGenConv2d()" << STD_ENDL;
52 #endif
53 }
```

6.73.4 Member Function Documentation

6.73.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KerDataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, virtual]

The actual operation.

```
64 {
65     int imgWidth = dstSize.x();
66     int imgHeight = dstSize.y();
67     int kerWidth = kerSize.x();
68     int kerHeight = kerSize.y();
69     int x, y, i, j;
70 
71     PixOpT pixOp(tags);
72     RedOpT redOp(tags);
73     KernelT kernel(kerPtr, kerSize, tags);
74 
75     typedef typename KerImgSigT::ArithType ArithType;
76     ArithType result, tmpVal, neutralElement(RedOpT::neutralElement());
77 
78     for (y=0 ; y<imgHeight ; y++) {
79         DstDataPtrType dPtr = dstPtr;
80         dPtr.incY(y);
81         for (x=0 ; x<imgWidth ; x++) {
82             SrcDataPtrType sPtr = srcPtr;
83             sPtr.incXYZ(x, y);
84             result = neutralElement;
85             for (j=0 ; j<kerHeight ; j++) {
86                 for (i=0 ; i<kerWidth ; i++) {
87                     tmpVal = pixOp.doIt(sPtr.readIncX(), kernel(i,j));
88                     redOp.doIt(result, tmpVal);
89                 }
90                 sPtr.decX(kerWidth);
91                 sPtr.incY();
92             }
93             dPtr.writeIncX(result);
94         }
95     }
96 }
```

The documentation for this class was generated from the following files:

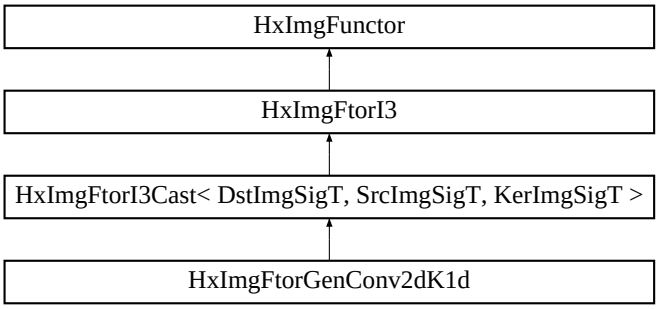
- **HxImgFtorGenConv2d.h**
- HxImgFtorGenConv2d.c

6.74 HxImgFtorGenConv2dK1d Class Template Reference

Instantiation of generic algorithm for generalized convolution operations on 2d images with a 1d kernel.

`#include <HxImgFtorGenConv2dK1d.h>`

Inheritance diagram for HxImgFtorGenConv2dK1d::



Public Types

- typedef **HxImgFtorGenConvK1dKey KeyType**

  The key type of this class.

- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source image.

- typedef KerImgSigT::DataPtrType **KerDataPtrType**

  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorGenConv2dK1d** ()

  Constructor.

- virtual ∼**HxImgFtorGenConv2dK1d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** ∗=0)

  Calls convolutionX, convolutionY, convolutionXIp, or convolutionYIp to do the actual work based on the "dimension" and "inplace" tags.

- void **convolutionX** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)
- void **convolutionY** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)
- void **convolutionXIp** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)
- void **convolutionYIp** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)

6.74.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> class HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >

Instantiation of generic algorithm for generalized convolution operations on 2d images with a 1d kernel.

6.74.2 Member Typedef Documentation

6.74.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef HxImgFtorGenConvK1dKey HxImgFtorGenConv2dK1d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.74.2.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef SrcImgSigT::DataPtrType HxImgFtorGenConv2dK1d::SrcDataPtrType

The data pointer type of the source image.

6.74.2.3 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef KerImgSigT::DataPtrType HxImgFtorGenConv2dK1d::KerDataPtrType

The data pointer type of the kernel image.

6.74.3 Constructor & Destructor Documentation

6.74.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv2dK1d () [inline]

Constructor.

```
24         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
25             HxImgFtorGenConvK1dKey(HxClassName<DstImgSigT>(),
26                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
27                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
28                 HxClassName<KernelT>()))
29 {
30 #ifdef CD_TRACE
31     HxEnvironment::instance()->outputStream()
32         << "HxImgFtorGenConv2dK1d::HxImgFtorGenConv2dK1d()" << STD_ENDL;
33 #endif
34 }
```

6.74.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv2dK1d () [virtual]

Destructor.

```
40 {
41 #ifdef CD_TRACE
42     HxEnvironment::instance()->outputStream()
43         << "HxImgFtorGenConv2dK1d::~HxImgFtorGenConv2dK1d()" << STD_ENDL;
44 #endif
45 }
```

6.74.4 Member Function Documentation

6.74.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KerDataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls convolutionX, convolutionY, convolutionXIp, or convolutionYIp to do the actual work based on the ”dimension” and ”inplace” tags.

```
56 {
57     int dimension = HxGetTag<int>(tags, "dimension");
58     bool inplace = HxGetTag(tags, "inplace", false);
59
60     if (inplace && description)
61         description->setVariation("inplace");
62
63     switch (dimension) {
64     case 1  :
65         if (inplace)
66             convolutionXIp(
67                 dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
68         else
69             convolutionX(
70                 dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
71         break;
72     case 2  :
73         if (inplace)
74             convolutionYIp(
75                 dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
76         else
77             convolutionY(
78                 dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
79         break;
80     default :
81         HxEnvironment::instance()->errorStream()
82             << "Generalized convolution (2d image, 1d kernel): "
83             << "cannot execute convolution in dimension " << dimension
84             << STD_ENDL;
85         HxEnvironment::instance()->flush();
86     }
87 }
```

The documentation for this class was generated from the following files:

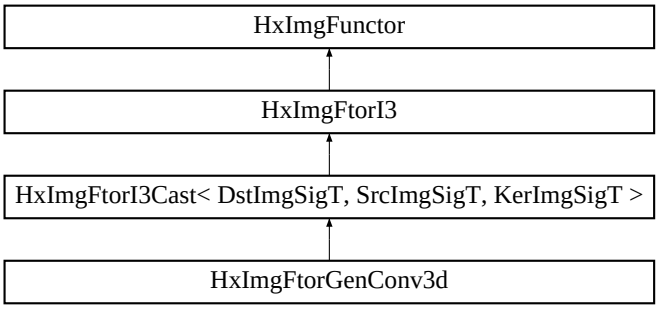
- **HxImgFtorGenConv2dK1d.h**
- HxImgFtorGenConv2dK1d.c

6.75 HxImgFtorGenConv3d Class Template Reference

Instantiation of generic algorithm for generalized convolution operations on 3d images.

`#include <HxImgFtorGenConv3d.h>`

Inheritance diagram for HxImgFtorGenConv3d::



Public Types

- typedef **HxImgFtorGenConvKey KeyType**

  The key type of this class.

- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source image.

- typedef KerImgSigT::DataPtrType **KerDataPtrType**

  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorGenConv3d** ()

  Constructor.

- virtual ~**HxImgFtorGenConv3d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.75.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> class HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >

Instantiation of generic algorithm for generalized convolution operations on 3d images.

6.75.2 Member Typedef Documentation

6.75.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef HxImgFtorGenConvKey HxImgFtorGenConv3d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.75.2.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef SrcImgSigT::DataPtrType HxImgFtorGenConv3d::SrcDataPtrType

The data pointer type of the source image.

6.75.2.3 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef KerImgSigT::DataPtrType HxImgFtorGenConv3d::KerDataPtrType

The data pointer type of the kernel image.

6.75.3 Constructor & Destructor Documentation

6.75.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv3d () [inline]

Constructor.

```
32         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
33             HxImgFtorGenConvKey(HxClassName<DstImgSigT>(),
34                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
35                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
36                 HxClassName<KernelT>()))
37 {
38 #ifdef CD_TRACE
39     HxEnvironment::instance()->outputStream()
40         << "HxImgFtorGenConv3d::HxImgFtorGenConv3d()" << STD_ENDL;
41 #endif
42 }
```

6.75.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv3d () [virtual]

Destructor.

```
48 {
49 #ifdef CD_TRACE
50     HxEnvironment::instance()->outputStream()
51         << "HxImgFtorGenConv3d::~HxImgFtorGenConv3d()" << STD_ENDL;
52 #endif
53 }
```

6.75.4 Member Function Documentation

6.75.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KerDataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, virtual]

The actual operation.

```
64 {
65     int imgWidth = dstSize.x();
66     int imgHeight = dstSize.y();
67     int imgDepth = dstSize.z();
68     int kerWidth = kerSize.x();
69     int kerHeight = kerSize.y();
70     int kerDepth = kerSize.z();
71     int x, y, z, i, j, k;
72 
73     PixOpT pixOp(tags);
74     RedOpT redOp(tags);
75     KernelT kernel(kerPtr, kerSize, tags);
76 
77     typedef typename KerImgSigT::ArithType ArithType;
78     ArithType result, tmpVal;
79 
80     for (z=0; z<imgDepth; z++) {
81         for (y=0 ; y<imgHeight ; y++) {
82             DstDataPtrType dPtr = dstPtr;
83             dPtr.incXYZ(0, y, z);
84             for (x=0 ; x<imgWidth ; x++) {
85                 SrcDataPtrType sPtr = srcPtr;
86                 sPtr.incXYZ(x, y, z);
87                 result = RedOpT::neutralElement();
88                 for (k=0; k<kerDepth; k++) {
89                     for (j=0 ; j<kerHeight ; j++) {
90                         for (i=0 ; i<kerWidth ; i++) {
91                             tmpVal = pixOp.doIt(sPtr.read(), kernel(i,j,k));
92                             redOp.doIt(result, tmpVal);
93                             sPtr.incX();
94                         }
95                         sPtr.decX(kerWidth);
96                         sPtr.incY();
97                     }
98                     sPtr.decY(kerHeight);
99                     sPtr.incZ();
100                 }
```

```
101                 dPtr.write(result);
102                 dPtr.incX();
103             }
104         }
105     }
106 }
```

The documentation for this class was generated from the following files:

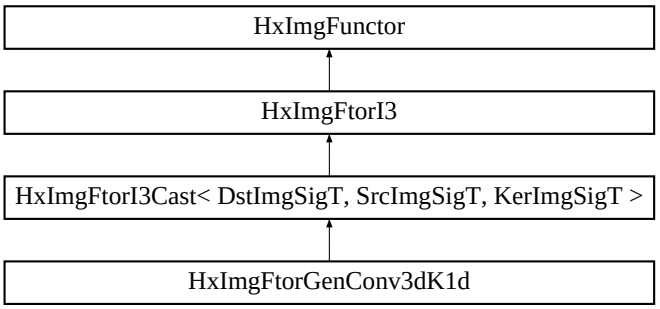
- **HxImgFtorGenConv3d.h**
- HxImgFtorGenConv3d.c

6.76 HxImgFtorGenConv3dK1d Class Template Reference

Instantiation of generic algorithm for generalized convolution operations on 3d images with a 1d kernel.

`#include <HxImgFtorGenConv3dK1d.h>`

Inheritance diagram for HxImgFtorGenConv3dK1d::



Public Types

- typedef **HxImgFtorGenConvK1dKey KeyType**
  The key type of this class.
- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**
  The data pointer type of the source image.
- typedef KerImgSigT::DataPtrType **KerDataPtrType**
  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorGenConv3dK1d** ()
  Constructor.
- virtual ~**HxImgFtorGenConv3dK1d** ()
  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls convolutionX, convolutionY, or convolutionZ to do the actual work based on the "dimension" tag.

- void **convolutionX** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)
- void **convolutionY** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)
- void **convolutionZ** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags)

6.76.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> class HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >

Instantiation of generic algorithm for generalized convolution operations on 3d images with a 1d kernel.

6.76.2 Member Typedef Documentation

6.76.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef HxImgFtorGenConvK1dKey HxImgFtorGenConv3dK1d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.76.2.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef SrcImgSigT::DataPtrType HxImgFtorGenConv3dK1d::SrcDataPtrType

The data pointer type of the source image.

6.76.2.3 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef KerImgSigT::DataPtrType HxImgFtorGenConv3dK1d::KerDataPtrType

The data pointer type of the kernel image.

6.76.3 Constructor & Destructor Documentation

6.76.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv3dK1d () [inline]

Constructor.

```
32         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
33             HxImgFtorGenConvK1dKey(
34                 HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
35                 HxClassName<KerImgSigT>(),
36                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
37                 HxClassName<KernelT>()))
38 {
39 #ifdef CD_TRACE
40     HxEnvironment::instance()->outputStream()
41         << "HxImgFtorGenConv3dK1d::HxImgFtorGenConv3dK1d()" << STD_ENDL;
42 #endif
43 }
```

6.76.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv3dK1d () [virtual]

Destructor.

```
49 {
50 #ifdef CD_TRACE
51     HxEnvironment::instance()->outputStream()
52         << "HxImgFtorGenConv3dK1d::~HxImgFtorGenConv3dK1d()" << STD_ENDL;
53 #endif
54 }
```

6.76.4 Member Function Documentation

6.76.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KerDataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, virtual]

Calls convolutionX, convolutionY, or convolutionZ to do the actual work based on the ”dimension” tag.

```
65 {
66     int dimension = HxGetTag<int>(tags, "dimension");
67
68     switch (dimension) {
69     case 1  :
70         convolutionX(dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
71         break;
72     case 2  :
73         convolutionY(dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
74         break;
75     case 3  :
76         convolutionZ(dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
77         break;
78     default :
79         convolutionX(dstPtr, srcPtr, kerPtr, dstSize, srcSize, kerSize, tags);
80     }
81 }
```

The documentation for this class was generated from the following files:

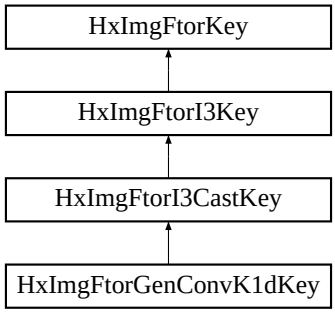
- **HxImgFtorGenConv3dK1d.h**
- HxImgFtorGenConv3dK1d.c

6.77 HxImgFtorGenConvK1dKey Class Reference

Key for HxImgFtorGenConvK1d.

`#include <HxImgFtorGenConvK1dKey.h>`

Inheritance diagram for HxImgFtorGenConvK1dKey::



Public Methods

- **HxImgFtorGenConvK1dKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** krnImgSig, **HxString** pixOpName, **HxString** redOpName, **HxString** kerName)

  Constructor.

6.77.1 Detailed Description

Key for HxImgFtorGenConvK1d.

6.77.2 Constructor & Destructor Documentation

6.77.2.1 HxImgFtorGenConvK1dKey::HxImgFtorGenConvK1dKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *krnImgSig*, HxString *pixOpName*, HxString *redOpName*, HxString *kerName*) [inline]

Constructor.

```
34     : HxImgFtorI3CastKey("HxImgFtorGenConvK1d", dstImgSig, srcImgSig, krnImgSig)
35 {
36     addArgument(pixOpName);
37     addArgument(redOpName);
38     addArgument(kerName);
39 }
```

The documentation for this class was generated from the following file:

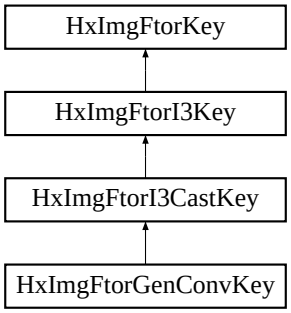
- **HxImgFtorGenConvK1dKey.h**

6.78 HxImgFtorGenConvKey Class Reference

Key for HxImgFtorGenConv.

`#include <HxImgFtorGenConvKey.h>`

Inheritance diagram for HxImgFtorGenConvKey::



Public Methods

- **HxImgFtorGenConvKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** krnImgSig, **HxString** pixOpName, **HxString** redOpName, **HxString** kerName)

  Constructor.

6.78.1 Detailed Description

Key for HxImgFtorGenConv.

6.78.2 Constructor & Destructor Documentation

6.78.2.1 HxImgFtorGenConvKey::HxImgFtorGenConvKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *krnImgSig*, HxString *pixOpName*, HxString *redOpName*, HxString *kerName*) [inline]

Constructor.

```
35     : HxImgFtorI3CastKey("HxImgFtorGenConv", dstImgSig, srcImgSig, krnImgSig)
36 {
37     addArgument(pixOpName);
38     addArgument(redOpName);
39     addArgument(kerName);
40 }
```

The documentation for this class was generated from the following file:

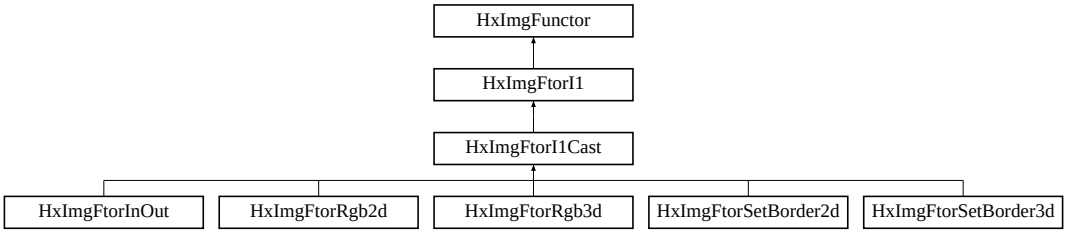
- **HxImgFtorGenConvKey.h**

6.79 HxImgFtorI1 Class Reference

Base class for image functors with one image parameter.

`#include <HxImgFtorI1.h>`

Inheritance diagram for HxImgFtorI1::



Public Types

- typedef **HxImgFtorI1Key KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorI1** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI1** ()

  Destructor.

- virtual void **callIt** (const **HxImageData** *img, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorI1Cast** (p. 394).

6.79.1 Detailed Description

Base class for image functors with one image parameter.

6.79.2 Member Typedef Documentation

6.79.2.1 typedef HxImgFtorI1Key HxImgFtorI1::KeyType

The key type of this class.

Reimplemented in **HxImgFtorI1Cast** (p. 395), **HxImgFtorInOut** (p. 441), **HxImgFtorRgb2d** (p. 464), **HxImgFtorRgb3d** (p. 467), **HxImgFtorSetBorder2d** (p. 475), **HxImgFtorSetBorder3d** (p. 478), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 441),

HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > > (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutHistogram< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467).

6.79.3 Constructor & Destructor Documentation

6.79.3.1 HxImgFtorI1::HxImgFtorI1 (const KeyType & *key*) [inline]

Constructor.

```
45                                              : HxImgFunctor(key)
46 {
47 }
```

6.79.3.2 HxImgFtorI1::~HxImgFtorI1 () [inline, virtual]

Destructor.

```
51 {
52 }
```

6.79.4 Member Function Documentation

6.79.4.1 virtual void HxImgFtorI1::callIt (const HxImageData * *img*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI1Cast** (p. 394).

Reimplemented in **HxImgFtorI1Cast** (p. 397).

The documentation for this class was generated from the following file:

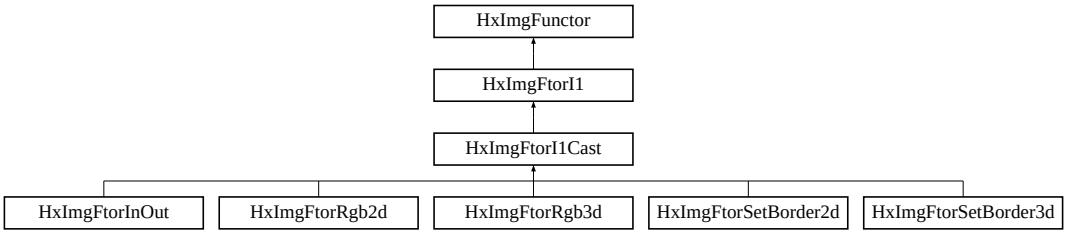
- **HxImgFtorI1.h**

6.80 HxImgFtorI1Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI1** (p. 392) to (statically typed) image data pointers.

`#include <HxImgFtorI1Cast.h>`

Inheritance diagram for HxImgFtorI1Cast::



Public Types

- typedef **HxImgFtorI1CastKey KeyType**

The key type of this class.

- typedef ImgSigT::DataPtrType **ImgDataPtrType**

  The data pointer type of the image.

Public Methods

- **HxImgFtorI1Cast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI1Cast** ()

  Destructor.

- virtual void **callIt** (const **HxImageData** *img, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** ptr, **HxSizes** size, **HxTagList** &tags, **HxImgFtorDescription** *=0)=0

  doIt is implemented by derived image functors.

6.80.1 Detailed Description

template<class ImgSigT> class HxImgFtorI1Cast< ImgSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI1** (p. 392) to (statically typed) image data pointers.

6.80.2 Member Typedef Documentation

6.80.2.1 template<class ImgSigT> typedef HxImgFtorI1CastKey HxImgFtorI1Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1** (p. 392).

Reimplemented in **HxImgFtorInOut** (p. 441), **HxImgFtorRgb2d** (p. 464), **HxImgFtorRgb3d** (p. 467), **HxImgFtorSetBorder2d** (p. 475), **HxImgFtorSetBorder3d** (p. 478), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType,**

typename ImgSigT::ArithType > > > (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutHistogram< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 464), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 467).

6.80.2.2 template<class ImgSigT> typedef ImgSigT::DataPtrType HxImgFtorI1Cast::ImgDataPtrType

The data pointer type of the image.

6.80.3 Constructor & Destructor Documentation

6.80.3.1 template<class ImgSigT> HxImgFtorI1Cast< ImgSigT >::HxImgFtorI1Cast (const KeyType & *key*) [inline]

Constructor.

```
56                                                          : HxImgFtorI1(key)
57 {
58 }
```

6.80.3.2 template<class ImgSigT> HxImgFtorI1Cast< ImgSigT >::~HxImgFtorI1Cast () [virtual]

Destructor.

```
20 {
21 }
```

6.80.4 Member Function Documentation

6.80.4.1 template<class ImgSigT> void HxImgFtorI1Cast< ImgSigT >::callIt (const HxImageData * *img*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI1** (p. 394).

```
26 {
27     TYPENAME ImgSigT::DataPtrType ptr
28         = HxMakeDataPtr<typename ImgSigT::DataPtrType>(img);
29
30     HxImgFtorDescription* description = getDescription();
31     if (description)
32     {
33         description->setTags(tags);
34         description->addArgument(img->signature(), img->sizes());
35         description->startTime();
36     }
37
38     doIt(ptr, img->sizes(), tags, description);
39     if (description)
40         description->stopTime();
41
42 }
```

6.80.4.2 template<class ImgSigT> virtual void HxImgFtorI1Cast< ImgSigT >::doIt (ImgDataPtrType *imgPtr*, HxSizes *imgSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors.

Reimplemented in **HxImgFtorInOut** (p. 441), **HxImgFtorRgb2d** (p. 465), **HxImgFtorRgb3d** (p. 468), **HxImgFtorSetBorder2d** (p. 476), **HxImgFtorSetBorder3d** (p. 478), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxInOutHistogram< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 441), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 441), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 465), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 468).

The documentation for this class was generated from the following files:

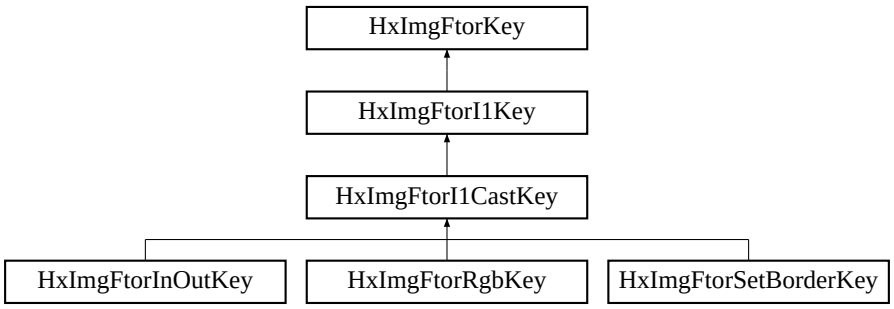
- **HxImgFtorI1Cast.h**
- HxImgFtorI1Cast.c

6.81 HxImgFtorI1CastKey Class Reference

Key for **HxImgFtorI1Cast** (p. 394).

`#include <HxImgFtorI1CastKey.h>`

Inheritance diagram for HxImgFtorI1CastKey::



Public Methods

- **HxImgFtorI1CastKey** (**HxString** className, **HxString** imgSig)

  Constructor.

6.81.1 Detailed Description

Key for **HxImgFtorI1Cast** (p. 394).

6.81.2 Constructor & Destructor Documentation

6.81.2.1 HxImgFtorI1CastKey::HxImgFtorI1CastKey (HxString *className*, HxString *imgSig*) [inline]

Constructor.

```
26     : HxImgFtorI1Key(className, imgSig)
27 {
28 }
```

The documentation for this class was generated from the following file:

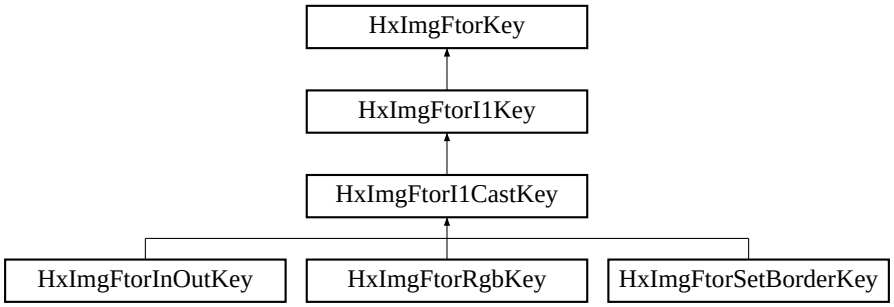
- **HxImgFtorI1CastKey.h**

6.82 HxImgFtorI1Key Class Reference

Key for **HxImgFtorI1** (p. 392).

`#include <HxImgFtorI1Key.h>`

Inheritance diagram for HxImgFtorI1Key::



Public Methods

- **HxImgFtorI1Key** (**HxString** className, **HxString** imgSig)
  Constructor.

6.82.1 Detailed Description

Key for **HxImgFtorI1** (p. 392).

6.82.2 Constructor & Destructor Documentation

6.82.2.1 HxImgFtorI1Key::HxImgFtorI1Key (HxString *className*, HxString *imgSig*) [inline]

Constructor.

```
26     : HxImgFtorKey(className)
27 {
28     addArgument(imgSig);
29 }
```

The documentation for this class was generated from the following file:

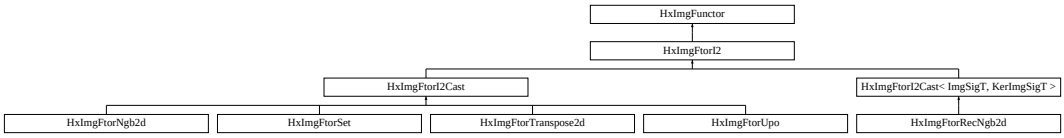
- **HxImgFtorI1Key.h**

6.83 HxImgFtorI2 Class Reference

Base class for image functors with two image parameters.

`#include <HxImgFtorI2.h>`

Inheritance diagram for HxImgFtorI2::



Public Types

- typedef **HxImgFtorI2Key KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorI2** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI2** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, const **HxImageData** *srcImg, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorI2Cast** (p. 405).

6.83.1 Detailed Description

Base class for image functors with two image parameters.

6.83.2 Member Typedef Documentation

6.83.2.1 typedef HxImgFtorI2Key HxImgFtorI2::KeyType

The key type of this class.

Reimplemented in **HxImgFtorI2Cast** (p. 407), **HxImgFtorNgb2d** (p. 457), **HxImgFtorRecNgb2d** (p. 461), **HxImgFtorSet** (p. 473), **HxImgFtorTranspose2d** (p. 484), **HxImgFtorUpo** (p. 487), **HxImgFtorI2Cast< DstSigT, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dFloat, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dByte, SrcSigT >** (p. 407), **HxImgFtorI2Cast< ImgSigT, ImgSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig3dDouble, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig3dByte, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec3Float, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec2Float, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dDouble, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig3dInt, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec2Byte, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec3Byte, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dInt, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec3Double, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec3Short, SrcSigT >** (p. 407), **HxImgFtorI2Cast< DstSigT, ImgSigT >** (p. 407), **HxImgFtorI2Cast< ResSigT, ImgSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec2Double, SrcSigT >** (p. 407), **HxImgFtorI2Cast< ImgSigT, KerSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig3dShort, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dShort, SrcSigT >** (p. 407), **HxImgFtorI2Cast< SrcSigT, SrcSigT >** (p. 407),

HxImgFtorI2Cast< HxImageSig2dVec2Int, SrcSigT > (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec3Int, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dVec2Short, SrcSigT >** (p. 407), **HxImgFtorI2Cast< ImgSigT, KerImgSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig2dComplex, SrcSigT >** (p. 407), **HxImgFtorI2Cast< HxImageSig3dFloat, SrcSigT >** (p. 407), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 457), **HxImgFtorRecNgb2d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 461), **HxImgFtorRecNgb2d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 461), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dComplex, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dFloat, SrcSigT >** (p. 473), **HxImgFtorTranspose2d< ImgSigT, ImgSigT >** (p. 484), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val<**

typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dComplex, SrcSigT, HxUpoCopy< typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< type-

name HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2d-

Short, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > > (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487).

6.83.3 Constructor & Destructor Documentation

6.83.3.1 HxImgFtorI2::HxImgFtorI2 (const KeyType & *key*) [inline]

Constructor.

```
48                                              : HxImgFunctor(key)
49 {
50 }
```

6.83.3.2 HxImgFtorI2::~HxImgFtorI2 () [inline, virtual]

Destructor.

```
54 {
55 }
```

6.83.4 Member Function Documentation

6.83.4.1 virtual void HxImgFtorI2::callIt (HxImageData * *dstImg*, const HxImageData * *srcImg*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI2Cast** (p. 405).

Reimplemented in **HxImgFtorI2Cast** (p. 410), **HxImgFtorI2Cast< DstSigT, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dFloat, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dByte, SrcSigT >** (p. 410), **HxImgFtorI2Cast< ImgSigT, ImgSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig3dDouble, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig3dByte, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec3Float, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec2Float, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dDouble, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig3dInt, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec2Byte, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec3Byte, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dInt, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec3Double, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec3Short, SrcSigT >** (p. 410), **HxImgFtorI2Cast< DstSigT, ImgSigT >** (p. 410), **HxImgFtorI2Cast< ResSigT, ImgSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec2Double, SrcSigT >** (p. 410), **HxImgFtorI2Cast< ImgSigT, KerSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig3dShort, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dShort, SrcSigT >** (p. 410), **HxImgFtorI2Cast< SrcSigT, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec2Int, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec3Int, SrcSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dVec2Short, SrcSigT >** (p. 410), **HxImgFtorI2Cast< ImgSigT, KerImgSigT >** (p. 410), **HxImgFtorI2Cast< HxImageSig2dComplex, SrcSigT >** (p. 410), and **HxImgFtorI2Cast< HxImageSig3dFloat, SrcSigT >** (p. 410).

The documentation for this class was generated from the following file:

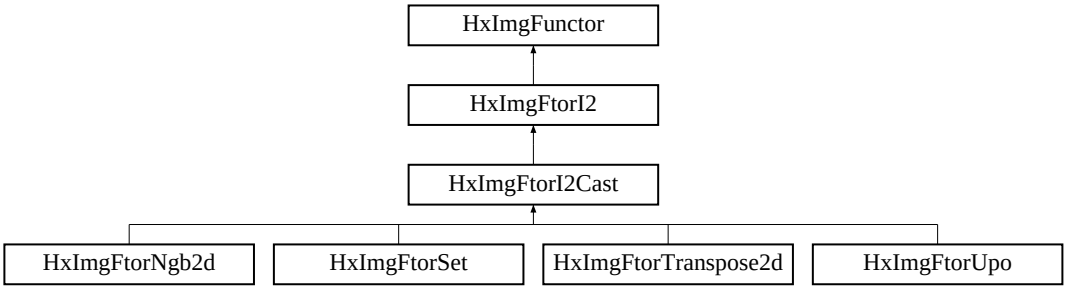
- **HxImgFtorI2.h**

6.84 HxImgFtorI2Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI2** (p. 400) to (statically typed) image data pointers.

`#include <HxImgFtorI2Cast.h>`

Inheritance diagram for HxImgFtorI2Cast::



Public Types

- typedef **HxImgFtorI2CastKey KeyType**

  The key type of this class.

- typedef DstImgSigT::DataPtrType **DstDataPtrType**

  The data pointer type of the destination image.

- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source image.

Public Methods

- **HxImgFtorI2Cast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI2Cast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, const **HxImageData** *srcImg, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)=0

  doIt is implemented by derived image functors.

6.84.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT> class HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI2** (p. 400) to (statically typed) image data pointers.

6.84.2 Member Typedef Documentation

6.84.2.1 template<class DstImgSigT, class SrcImgSigT> typedef HxImgFtorI2CastKey HxImgFtorI2Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2** (p. 401).

Reimplemented in **HxImgFtorNgb2d** (p. 457), **HxImgFtorRecNgb2d** (p. 461), **HxImgFtorSet** (p. 473), **HxImgFtorTranspose2d** (p. 484), **HxImgFtorUpo** (p. 487), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 457), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 457), **HxImgFtorRecNgb2d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 461), **HxImgFtorRecNgb2d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 461), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dComplex, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dFloat, SrcSigT >** (p. 473), **HxImgFtorTranspose2d< ImgSigT, ImgSigT >** (p. 484), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpo-**

Sub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dComplex, SrcSigT, HxUpoCopy< typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSig-**

T, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec2Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 487), HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > > (p. 487), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val<

typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 487), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< HxImageSig2dShort, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > >** (p. 487), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 487).

6.84.2.2 template<class DstImgSigT, class SrcImgSigT> typedef DstImgSigT::DataPtrType HxImgFtorI2Cast::DstDataPtrType

The data pointer type of the destination image.

6.84.2.3 template<class DstImgSigT, class SrcImgSigT> typedef SrcImgSigT::DataPtrType HxImgFtorI2Cast::SrcDataPtrType

The data pointer type of the source image.

6.84.3 Constructor & Destructor Documentation

6.84.3.1 template<class DstImgSigT, class SrcImgSigT> HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >::HxImgFtorI2Cast (const KeyType & *key*) [inline]

Constructor.

```
65             : HxImgFtorI2(key){}
```

6.84.3.2 template<class DstImgSigT, class SrcImgSigT> HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >::~HxImgFtorI2Cast () [virtual]

Destructor.

```
26 {
27 #ifdef CD_TRACE
28     HxEnvironment::instance()->outputStream()
29         << "HxImgFtorI2Cast<>::~HxImgFtorI2Cast()" << STD_ENDL;
30     HxEnvironment::instance()->flush();
31 #endif
32 }
```

6.84.4 Member Function Documentation

6.84.4.1 template<class DstImgSigT, class SrcImgSigT> void HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >::callIt (HxImageData * *dstImg*, const HxImageData * *srcImg*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI2** (p. 405).

```
38 {
39     TYPENAME DstImgSigT::DataPtrType dstPtr
40         = HxMakeDataPtr<typename DstImgSigT::DataPtrType>(dstImg);
41     TYPENAME SrcImgSigT::DataPtrType srcPtr
42         = HxMakeDataPtr<typename SrcImgSigT::DataPtrType>(srcImg);
43     HxImgFtorDescription* description = getDescription();
44     if (description)
45     {
46         description->setTags(tags);
47         description->addArgument(dstImg->signature(), dstImg->sizes());
48         description->addArgument(srcImg->signature(), srcImg->sizes());
49         description->startTime();
50     }
51
52     doIt(
53         dstPtr, srcPtr, dstImg->sizes(), srcImg->sizes(),
54         tags, description);
55
56     if (description)
57         description->stopTime();
58 }
```

6.84.4.2 template<class DstImgSigT, class SrcImgSigT> virtual void HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors.

Reimplemented in **HxImgFtorNgb2d** (p. 458), **HxImgFtorSet** (p. 473), **HxImgFtorTranspose2d** (p. 485), **HxImgFtorUpo** (p. 488), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 458), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > >** (p. 458), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 458), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 458), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dByte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dDouble, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dInt, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig3dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dShort, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT >** (p. 473), **HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT >** (p. 473), **HxImg-**

FtorSet< HxImageSig2dComplex, SrcSigT > (p. 473), **HxImgFtorSet< HxImageSig3dFloat, SrcSigT >** (p. 473), **HxImgFtorTranspose2d< ImgSigT, ImgSigT >** (p. 485), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2d-**

Complex, SrcSigT, HxUpoCopy< typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > > (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec2Int,**

SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > > (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 488), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< HxImageSig2dShort, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > >** (p. 488), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 488).

The documentation for this class was generated from the following files:

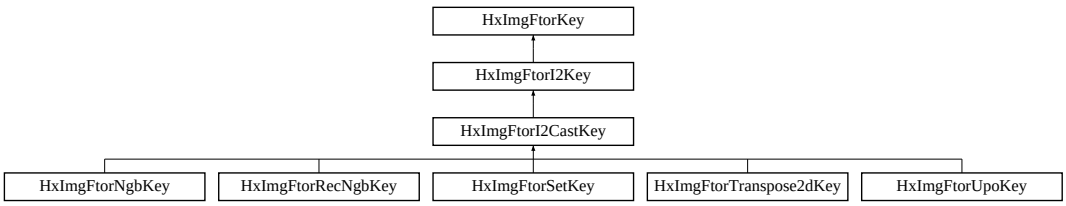
- **HxImgFtorI2Cast.h**
- HxImgFtorI2Cast.c

6.85 HxImgFtorI2CastKey Class Reference

Key for **HxImgFtorI2Cast** (p. 405).

`#include <HxImgFtorI2CastKey.h>`

Inheritance diagram for HxImgFtorI2CastKey::



Public Methods

- **HxImgFtorI2CastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

6.85.1 Detailed Description

Key for **HxImgFtorI2Cast** (p. 405).

6.85.2 Constructor & Destructor Documentation

6.85.2.1 HxImgFtorI2CastKey::HxImgFtorI2CastKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
30     : HxImgFtorI2Key(className, sig1Name, sig2Name)
31 {
32 }
```

The documentation for this class was generated from the following file:

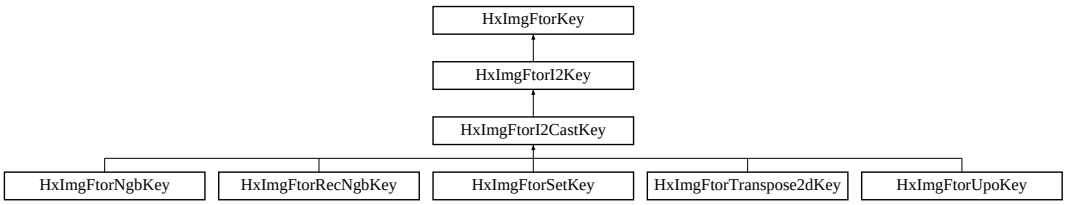
- **HxImgFtorI2CastKey.h**

6.86 HxImgFtorI2Key Class Reference

Key for **HxImgFtorI2** (p. 400).

`#include <HxImgFtorI2Key.h>`

Inheritance diagram for HxImgFtorI2Key::



Public Methods

- **HxImgFtorI2Key** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

6.86.1 Detailed Description

Key for **HxImgFtorI2** (p. 400).

6.86.2 Constructor & Destructor Documentation

6.86.2.1 HxImgFtorI2Key::HxImgFtorI2Key (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
29     : HxImgFtorKey(className)
30 {
31     HxStringList l;
32     l << sig1Name << sig2Name;
33     setArguments(l.begin(), l.end());
34 }
```

The documentation for this class was generated from the following file:

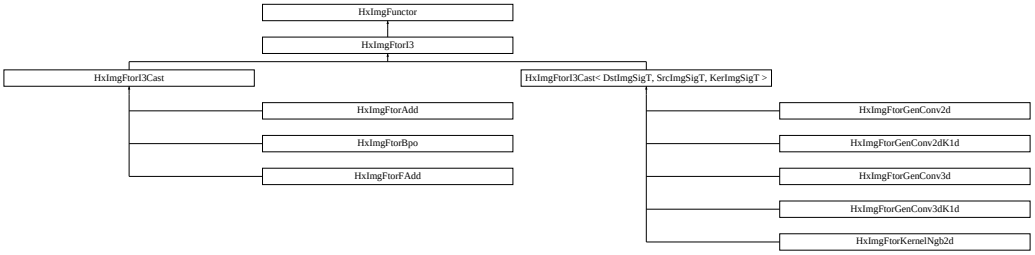
- **HxImgFtorI2Key.h**

6.87 HxImgFtorI3 Class Reference

Base class for image functors with three image parameters.

`#include <HxImgFtorI3.h>`

Inheritance diagram for HxImgFtorI3::



Public Types

- typedef **HxImgFtorI3Key KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorI3** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI3** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, const **HxImageData** *src1Img, const **HxImageData** *src2Img, **HxTagList** &tags)=0

callIt is implemented by **HxImgFtorI3Cast** (p. 419).

6.87.1 Detailed Description

Base class for image functors with three image parameters.

6.87.2 Member Typedef Documentation

6.87.2.1 typedef HxImgFtorI3Key HxImgFtorI3::KeyType

The key type of this class.

Reimplemented in **HxImgFtorBpo** (p. 374), **HxImgFtorGenConv2d** (p. 379), **HxImgFtorGenConv2dK1d** (p. 382), **HxImgFtorGenConv3d** (p. 385), **HxImgFtorGenConv3dK1d** (p. 388), **HxImgFtorI3Cast** (p. 420), **HxImgFtorKernelNgb2d** (p. 444), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, Img-**

SigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 374), **HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMaxAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoSupAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoInfAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoInfAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMaxAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoSupAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv3d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel3d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 385), **HxImgFtorGenConv3dK1d< DstSigT, SrcSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename SrcSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 388), **HxImgFtorI3Cast< DstSigT, SrcSigT, KerSigT >** (p. 420), **HxImgFtorI3Cast< DstSigT, ImgSigT, ImgSigT >** (p. 420), **HxImgFtorI3Cast< ImgSigT, ImgSigT, ImgSigT >** (p. 420), **HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, KerImgSigT**

> (p. 420), **HxImgFtorI3Cast< ImgSigT, KerSigT, KerSigT >** (p. 420), **HxImgFtorI3Cast< DstSigT, DstSigT, ImgSigT >** (p. 420), and **HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSigT, HxNgbNormCorrelation< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > >** (p. 444).

6.87.3 Constructor & Destructor Documentation

6.87.3.1 HxImgFtorI3::HxImgFtorI3 (const KeyType & *key*) [inline]

Constructor.

```
48                                              : HxImgFunctor(key)
49 {
50 }
```

6.87.3.2 HxImgFtorI3::~HxImgFtorI3 () [inline, virtual]

Destructor.

```
54 {
55 }
```

6.87.4 Member Function Documentation

6.87.4.1 virtual void HxImgFtorI3::callIt (HxImageData * *dstImg*, const HxImageData * *src1Img*, const HxImageData * *src2Img*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI3Cast** (p. 419).

Reimplemented in **HxImgFtorI3Cast** (p. 423), **HxImgFtorI3Cast< DstSigT, SrcSigT, KerSigT >** (p. 423), **HxImgFtorI3Cast< DstSigT, ImgSigT, ImgSigT >** (p. 423), **HxImgFtorI3Cast< ImgSigT, ImgSigT, ImgSigT >** (p. 423), **HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, KerImgSigT >** (p. 423), **HxImgFtorI3Cast< ImgSigT, KerSigT, KerSigT >** (p. 423), and **HxImgFtorI3Cast< DstSigT, DstSigT, ImgSigT >** (p. 423).

The documentation for this class was generated from the following file:

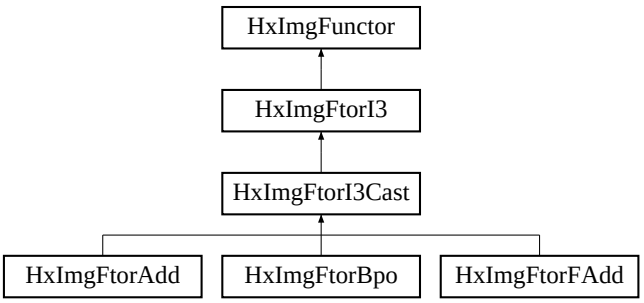
- **HxImgFtorI3.h**

6.88 HxImgFtorI3Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI3** (p. 416) to (statically typed) image data pointers.

`#include <HxImgFtorI3Cast.h>`

Inheritance diagram for HxImgFtorI3Cast::



Public Types

- typedef **HxImgFtorI3CastKey KeyType**

  The key type of this class.

- typedef DstImgSigT::DataPtrType **DstDataPtrType**

  The data pointer type of the destination image.

- typedef Src1ImgSigT::DataPtrType **Src1DataPtrType**

  The data pointer type of the first source image.

- typedef Src2ImgSigT::DataPtrType **Src2DataPtrType**

  The data pointer type of the second source image.

Public Methods

- **HxImgFtorI3Cast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI3Cast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, const **HxImageData** *src1Img, const **HxImageData** *src2Img, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **Src1DataPtrType** src1Ptr, **Src2DataPtrType** src2Ptr, **HxSizes** dstSize, **HxSizes** src1Size, **HxSizes** src2Size, **HxTagList** &tags, **HxImgFtorDescription** *=0)=0

  doIt is implemented by derived image functors.

6.88.1 Detailed Description

template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> class HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorI3** (p. 416) to (statically typed) image data pointers.

6.88.2 Member Typedef Documentation

6.88.2.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> typedef HxImgFtorI3CastKey HxImgFtorI3Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3** (p. 417).

Reimplemented in **HxImgFtorBpo** (p. 374), **HxImgFtorGenConv2d** (p. 379), **HxImgFtorGenConv2dK1d** (p. 382), **HxImgFtorGenConv3d** (p. 385), **HxImgFtorGenConv3dK1d** (p. 388), **HxImgFtorKernelNgb2d** (p. 444), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT,**

ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 374), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMaxAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoSupAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoInfAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 379), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoInfAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMaxAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoSupAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 382), **HxImgFtorGenConv3d< ImgSigT, KerSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel3d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 385), **HxImgFtorGenConv3dK1d< DstSigT, SrcSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename SrcSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > >** (p. 388), and **HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSig-**

T, HxNgbNormCorrelation< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > > (p. 444).

6.88.2.2 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> typedef DstImgSigT::DataPtrType HxImgFtorI3Cast::DstDataPtrType

The data pointer type of the destination image.

6.88.2.3 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> typedef Src1ImgSigT::DataPtrType HxImgFtorI3Cast::Src1DataPtrType

The data pointer type of the first source image.

6.88.2.4 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> typedef Src2ImgSigT::DataPtrType HxImgFtorI3Cast::Src2DataPtrType

The data pointer type of the second source image.

6.88.3 Constructor & Destructor Documentation

6.88.3.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >::HxImgFtorI3Cast (const KeyType & *key*) [inline]

Constructor.

```
69 : HxImgFtorI3(key){}
```

6.88.3.2 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >::~HxImgFtorI3Cast () [virtual]

Destructor.

```
25 {
26 #ifdef CD_TRACE
27     HxEnvironment::instance()->outputStream()
28         << "~HxImgFtorI3Cast()" << STD_ENDL;
29     HxEnvironment::instance()->flush();
30 #endif
31 }
```

6.88.4 Member Function Documentation

6.88.4.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> void HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >::callIt (HxImageData * *dstImg*, const HxImageData * *src1Img*, const HxImageData * *src2Img*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI3** (p. 419).

```
38 {
39     TYPENAME DstImgSigT::DataPtrType dstPtr
40         = HxMakeDataPtr<typename DstImgSigT::DataPtrType>(dstImg);
41     TYPENAME Src1ImgSigT::DataPtrType src1Ptr
42         = HxMakeDataPtr<typename Src1ImgSigT::DataPtrType>(src1Img);
43     TYPENAME Src2ImgSigT::DataPtrType src2Ptr
44         = HxMakeDataPtr<typename Src2ImgSigT::DataPtrType>(src2Img);
45
46     HxImgFtorDescription* description = getDescription();
47     if (description)
48     {
49         description->setTags(tags);
50         description->addArgument(dstImg->signature(), dstImg->sizes());
51         description->addArgument(src1Img->signature(), src1Img->sizes());
52         description->addArgument(src2Img->signature(), src2Img->sizes());
53         description->startTime();
54     }
55
56     doIt(dstPtr, src1Ptr, src2Ptr,
57         dstImg->sizes(), src1Img->sizes(), src2Img->sizes(), tags, description);
58
59     if (description)
60         description->stopTime();
61 }
```

6.88.4.2 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT> virtual void HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >::doIt (DstDataPtrType *dstPtr*, Src1DataPtrType *src1Ptr*, Src2DataPtrType *src2Ptr*, HxSizes *dstSize*, HxSizes *src1Size*, HxSizes *src2Size*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, pure virtual]

doIt is implemented by derived image functors.

Reimplemented in **HxImgFtorBpo** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::Arith-**

Type, typename ImgSigT::ArithType > > (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375), and **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 375).

The documentation for this class was generated from the following files:

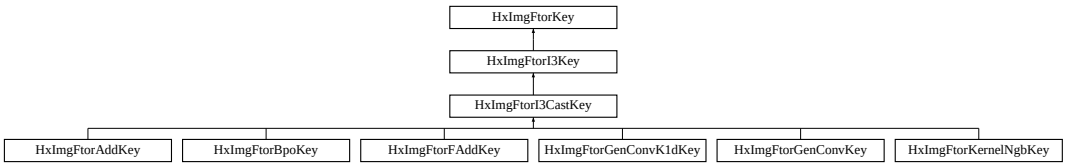
- **HxImgFtorI3Cast.h**
- HxImgFtorI3Cast.c

6.89 HxImgFtorI3CastKey Class Reference

Key for **HxImgFtorI3Cast** (p. 419).

`#include <HxImgFtorI3CastKey.h>`

Inheritance diagram for HxImgFtorI3CastKey::



Public Methods

- **HxImgFtorI3CastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name)

  Constructor.

6.89.1 Detailed Description

Key for **HxImgFtorI3Cast** (p. 419).

6.89.2 Constructor & Destructor Documentation

6.89.2.1 HxImgFtorI3CastKey::HxImgFtorI3CastKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*, HxString *sig3Name*) [inline]

Constructor.

```
32     : HxImgFtorI3Key(className, sig1Name, sig2Name, sig3Name)
33 {
34 }
```

The documentation for this class was generated from the following file:

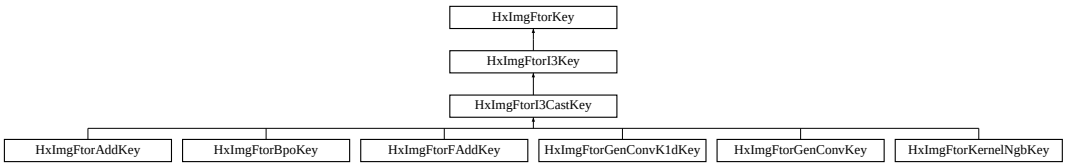
- **HxImgFtorI3CastKey.h**

6.90 HxImgFtorI3Key Class Reference

Key for **HxImgFtorI3** (p. 416).

`#include <HxImgFtorI3Key.h>`

Inheritance diagram for HxImgFtorI3Key::



Public Methods

- **HxImgFtorI3Key** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name)

  Constructor.

6.90.1 Detailed Description

Key for **HxImgFtorI3** (p. 416).

6.90.2 Constructor & Destructor Documentation

6.90.2.1 HxImgFtorI3Key::HxImgFtorI3Key (HxString *className*, HxString *sig1Name*, HxString *sig2Name*, HxString *sig3Name*) [inline]

Constructor.

```
32     : HxImgFtorKey(className)
33 {
34     HxStringList l;
35     l << sig1Name << sig2Name << sig3Name;
36     setArguments(l.begin(),l.end());
37 }
```

The documentation for this class was generated from the following file:

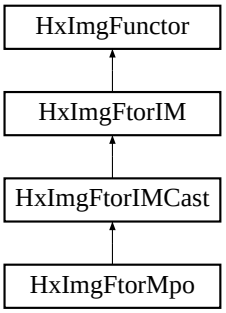
- **HxImgFtorI3Key.h**

6.91 HxImgFtorIM Class Reference

Base class for image functors with 1 + M image parameters.

`#include <HxImgFtorIM.h>`

Inheritance diagram for HxImgFtorIM::



Public Types

- typedef **HxImgFtorIMKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorIM** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorIM** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, **HxImageData** **srcImgs, const int nImgs, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorIMCast** (p. 428).

6.91.1 Detailed Description

Base class for image functors with 1 + M image parameters.

6.91.2 Member Typedef Documentation

6.91.2.1 typedef HxImgFtorIMKey HxImgFtorIM::KeyType

The key type of this class.

Reimplemented in **HxImgFtorIMCast** (p. 429), **HxImgFtorMpo** (p. 453), **HxImgFtorIMCast< DstSigT, SrcsSigT >** (p. 429), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 453).

6.91.3 Constructor & Destructor Documentation

6.91.3.1 HxImgFtorIM::HxImgFtorIM (const KeyType & *key*) [inline]

Constructor.

```
49                                          : HxImgFunctor(key)
50 {
51 }
```

6.91.3.2 HxImgFtorIM::~HxImgFtorIM () [inline, virtual]

Destructor.

```
55 {
56 }
```

6.91.4 Member Function Documentation

6.91.4.1 virtual void HxImgFtorIM::callIt (HxImageData * *dstImg*, HxImageData ** *srcImgs*, const int *nImgs*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorIMCast** (p. 428).

Reimplemented in **HxImgFtorIMCast** (p. 430), and **HxImgFtorIMCast< DstSigT, SrcsSigT >** (p. 430).

The documentation for this class was generated from the following file:

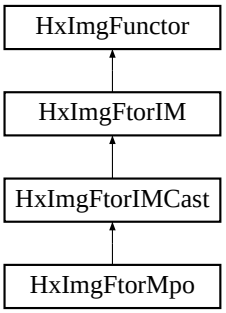
- **HxImgFtorIM.h**

6.92 HxImgFtorIMCast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorIM** (p. 427) to (statically typed) image data pointers.

`#include <HxImgFtorIMCast.h>`

Inheritance diagram for HxImgFtorIMCast::



Public Types

- typedef **HxImgFtorIMCastKey KeyType**

  The key type of this class.

- typedef DstImgSigT::DataPtrType **DstDataPtrType**

  The data pointer type of the destination image.

- typedef SrcImgsSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source images.

- typedef HxDataPtrArray< SrcImgsSigT > **SrcDataPtrArray**

  An array of data pointers to source images.

Public Methods

- **HxImgFtorIMCast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorIMCast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *dstImg, **HxImageData** **srcImgs, const int nImgs, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrArray** &srcPtrs, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)=0

  doIt is implemented by derived image functors.

6.92.1 Detailed Description

template<class DstImgSigT, class SrcImgsSigT> class HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorIM** (p. 427) to (statically typed) image data pointers.

6.92.2 Member Typedef Documentation

6.92.2.1 template<class DstImgSigT, class SrcImgsSigT> typedef HxImgFtorIMCastKey HxImgFtorIMCast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIM** (p. 427).

Reimplemented in **HxImgFtorMpo** (p. 453), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 453).

6.92.2.2 template<class DstImgSigT, class SrcImgsSigT> typedef DstImgSigT::DataPtrType HxImgFtorIMCast::DstDataPtrType

The data pointer type of the destination image.

6.92.2.3 template<class DstImgSigT, class SrcImgsSigT> typedef SrcImgsSigT::DataPtrType HxImgFtorIMCast::SrcDataPtrType

The data pointer type of the source images.

6.92.2.4 template<class DstImgSigT, class SrcImgsSigT> typedef HxDataPtrArray<SrcImgsSigT> HxImgFtorIMCast::SrcDataPtrArray

An array of data pointers to source images.

6.92.3 Constructor & Destructor Documentation

6.92.3.1 template<class DstImgSigT, class SrcImgsSigT> HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::HxImgFtorIMCast (const KeyType & *key*) [inline]

Constructor.

```
71             : HxImgFtorIM(key){}
```

6.92.3.2 template<class DstImgSigT, class SrcImgsSigT> HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::~HxImgFtorIMCast () [virtual]

Destructor.

```
27 {
28 #ifdef CD_TRACE
29     HxEnvironment::instance()->outputStream()
30         << "HxImgFtorIMCast<>::~HxImgFtorIMCast()" << STD_ENDL;
31     HxEnvironment::instance()->flush();
32 #endif
33 }
```

6.92.4 Member Function Documentation

6.92.4.1 template<class DstImgSigT, class SrcImgsSigT> void HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::callIt (HxImageData * *dstImg*, HxImageData ** *srcImgs*, const int *nImgs*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorIM** (p. 428).

```
39 {
40     TYPENAME DstImgSigT::DataPtrType dstPtr
41         = HxMakeDataPtr<typename DstImgSigT::DataPtrType>(dstImg);
42     SrcDataPtrArray srcPtrs(srcImgs, nImgs);
43
44     HxImgFtorDescription* description = getDescription();
45     if (description)
46     {
47         description->setTags(tags);
48         description->addArgument(dstImg->signature(), dstImg->sizes());
49         description->addArgument(srcImgs[0]->signature(), srcImgs[0]->sizes());
50         description->startTime();
51     }
52
53     doIt(
54         dstPtr, srcPtrs, dstImg->sizes(), srcImgs[0]->sizes(),
55         tags, description);
56
57     if (description)
58         description->stopTime();
59 }
```

6.92.4.2 template<class DstImgSigT, class SrcImgsSigT> virtual void HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors.

Reimplemented in **HxImgFtorMpo** (p. 454), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 454).

The documentation for this class was generated from the following files:

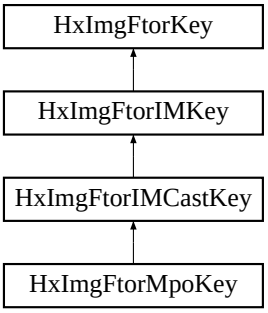
- **HxImgFtorIMCast.h**
- HxImgFtorIMCast.c

6.93 HxImgFtorIMCastKey Class Reference

Key for **HxImgFtorIMCast** (p. 428).

`#include <HxImgFtorIMCastKey.h>`

Inheritance diagram for HxImgFtorIMCastKey::



Public Methods

- **HxImgFtorIMCastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

6.93.1 Detailed Description

Key for **HxImgFtorIMCast** (p. 428).

6.93.2 Constructor & Destructor Documentation

6.93.2.1 HxImgFtorIMCastKey::HxImgFtorIMCastKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
31     : HxImgFtorIMKey(className, sig1Name, sig2Name)
32 {
33 }
```

The documentation for this class was generated from the following file:

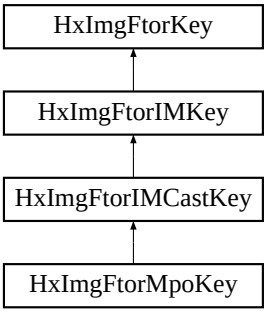
- **HxImgFtorIMCastKey.h**

6.94 HxImgFtorIMKey Class Reference

Key for **HxImgFtorIM** (p. 427).

`#include <HxImgFtorIMKey.h>`

Inheritance diagram for HxImgFtorIMKey::



Public Methods

- **HxImgFtorIMKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)

  Constructor.

6.94.1 Detailed Description

Key for **HxImgFtorIM** (p. 427).

6.94.2 Constructor & Destructor Documentation

6.94.2.1 HxImgFtorIMKey::HxImgFtorIMKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
30     : HxImgFtorKey(className)
31 {
32     HxStringList l;
33     l << sig1Name << sig2Name;
34     setArguments(l.begin(), l.end());
35 }
```

The documentation for this class was generated from the following file:

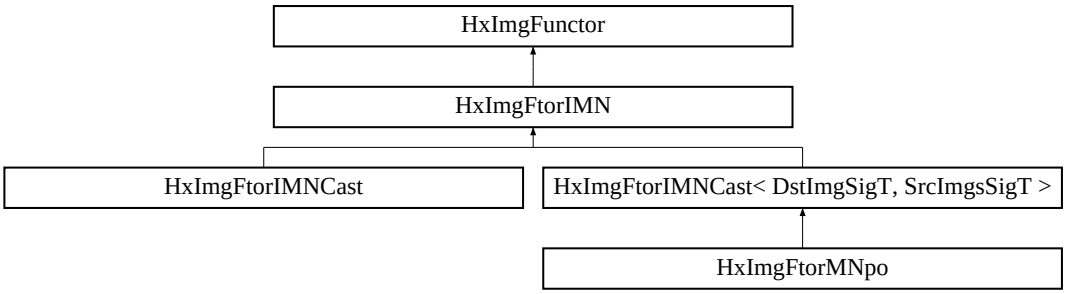
- **HxImgFtorIMKey.h**

6.95 HxImgFtorIMN Class Reference

Base class for image functors with M + N image parameters.

`#include <HxImgFtorIMN.h>`

Inheritance diagram for HxImgFtorIMN::



Public Types

- typedef **HxImgFtorIMNKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorIMN** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorIMN** ()

  Destructor.

- virtual void **callIt** (**HxImageData** **dstImgs, int dstCnt, **HxImageData** **srcImgs, int srcCnt, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorIMNCast** (p. 435).

6.95.1 Detailed Description

Base class for image functors with M + N image parameters.

6.95.2 Member Typedef Documentation

6.95.2.1 typedef HxImgFtorIMNKey HxImgFtorIMN::KeyType

The key type of this class.

Reimplemented in **HxImgFtorIMNCast** (p. 436), **HxImgFtorMNpo** (p. 450), and **HxImgFtorIMNCast< DstImgSigT, SrcImgsSigT >** (p. 436).

6.95.3 Constructor & Destructor Documentation

6.95.3.1 HxImgFtorIMN::HxImgFtorIMN (const KeyType & *key*) [inline]

Constructor.

```
50                                                  : HxImgFunctor(key)
51 {
52 }
```

6.95.3.2 HxImgFtorIMN::~HxImgFtorIMN () [inline, virtual]

Destructor.

```
56 {
57 }
```

6.95.4 Member Function Documentation

6.95.4.1 virtual void HxImgFtorIMN::callIt (HxImageData ** *dstImgs*, int *dstCnt*, HxImageData ** *srcImgs*, int *srcCnt*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorIMNCast** (p. 435).

Reimplemented in **HxImgFtorIMNCast** (p. 437), and **HxImgFtorIMNCast< DstImgSigT, SrcImgsSigT >** (p. 437).

The documentation for this class was generated from the following file:

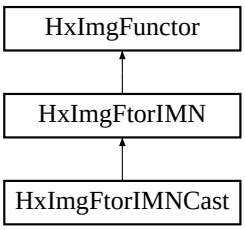
- **HxImgFtorIMN.h**

6.96 HxImgFtorIMNCast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorIMN** (p. 433) to (statically typed) image data pointers.

`#include <HxImgFtorIMNCast.h>`

Inheritance diagram for HxImgFtorIMNCast::



Public Types

- typedef **HxImgFtorIMNCastKey KeyType**

  The key type of this class.

- typedef DstImgsSigT::DataPtrType **DstDataPtrType**

  The data pointer type of the destination images.

- typedef SrcImgsSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source images.

- typedef HxDataPtrArray< SrcImgsSigT > **SrcDataPtrArray**

  An array of data pointers to source images.

- typedef HxDataPtrArray< DstImgsSigT > **DstDataPtrArray**

  An array of data pointers to destination images.

Public Methods

- **HxImgFtorIMNCast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorIMNCast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** **dstImgs, int dstCnt, **HxImageData** **srcImgs, int srcCnt, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**DstDataPtrArray** &dstPtrs, **SrcDataPtrArray** &srcPtrs, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)=0

  doIt is implemented by derived image functors.

6.96.1 Detailed Description

template<class DstImgsSigT, class SrcImgsSigT> class HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 288) parameters of **HxImgFtorIMN** (p. 433) to (statically typed) image data pointers.

6.96.2 Member Typedef Documentation

6.96.2.1 template<class DstImgsSigT, class SrcImgsSigT> typedef HxImgFtorIMNCastKey HxImgFtorIMNCast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMN** (p. 434).

Reimplemented in **HxImgFtorMNpo** (p. 450).

6.96.2.2 template<class DstImgsSigT, class SrcImgsSigT> typedef DstImgsSigT::DataPtrType HxImgFtorIMNCast::DstDataPtrType

The data pointer type of the destination images.

6.96.2.3 template<class DstImgsSigT, class SrcImgsSigT> typedef SrcImgsSigT::DataPtrType HxImgFtorIMNCast::SrcDataPtrType

The data pointer type of the source images.

6.96.2.4 template<class DstImgsSigT, class SrcImgsSigT> typedef HxDataPtrArray<SrcImgsSigT> HxImgFtorIMNCast::SrcDataPtrArray

An array of data pointers to source images.

6.96.2.5 template<class DstImgsSigT, class SrcImgsSigT> typedef HxDataPtrArray<DstImgsSigT> HxImgFtorIMNCast::DstDataPtrArray

An array of data pointers to destination images.

6.96.3 Constructor & Destructor Documentation

6.96.3.1 template<class DstImgsSigT, class SrcImgsSigT> HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >::HxImgFtorIMNCast (const KeyType & *key*) [inline]

Constructor.

```
74             : HxImgFtorIMN(key){}
```

6.96.3.2 template<class DstImgsSigT, class SrcImgsSigT> HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >::~HxImgFtorIMNCast () [virtual]

Destructor.

```
27 {
28 #ifdef CD_TRACE
29     HxEnvironment::instance()->outputStream()
30         << "HxImgFtorIMNCast<>::~HxImgFtorIMNCast()" << STD_ENDL;
31     HxEnvironment::instance()->flush();
32 #endif
33 }
```

6.96.4 Member Function Documentation

6.96.4.1 template<class DstImgsSigT, class SrcImgsSigT> void HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >::callIt (HxImageData ** *dstImgs*, int *dstCnt*, HxImageData ** *srcImgs*, int *srcCnt*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorIMN** (p. 434).

```
40 {
41     DstDataPtrArray dstPtrs(dstImgs, dstCnt);
42     SrcDataPtrArray srcPtrs(srcImgs, srcCnt);
43 
44     HxImgFtorDescription* description = getDescription();
45     if (description)
46     {
47         description->setTags(tags);
48         description->addArgument(dstImgs[0]->signature(), dstImgs[0]->sizes());
49         description->addArgument(srcImgs[0]->signature(), srcImgs[0]->sizes());
50         description->startTime();
51     }
52 
53     doIt(
54         dstPtrs, srcPtrs, dstImgs[0]->sizes(), srcImgs[0]->sizes(),
55         tags, description);
56 
57     if (description)
58         description->stopTime();
59 }
```

6.96.4.2 template<class DstImgsSigT, class SrcImgsSigT> virtual void HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >::doIt (DstDataPtrArray & *dstPtrs*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors.

Reimplemented in **HxImgFtorMNpo** (p. 451).

The documentation for this class was generated from the following files:

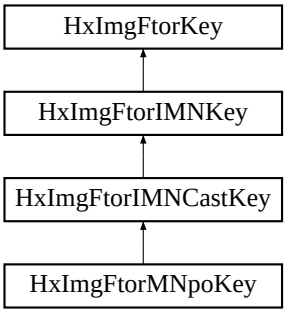
- **HxImgFtorIMNCast.h**
- HxImgFtorIMNCast.c

6.97 HxImgFtorIMNCastKey Class Reference

Key for **HxImgFtorIMNCast** (p. 435).

`#include <HxImgFtorIMNCastKey.h>`

Inheritance diagram for HxImgFtorIMNCastKey::



Public Methods

- **HxImgFtorIMNCastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)

  Constructor.

6.97.1 Detailed Description

Key for **HxImgFtorIMNCast** (p. 435).

6.97.2 Constructor & Destructor Documentation

6.97.2.1 HxImgFtorIMNCastKey::HxImgFtorIMNCastKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
31     : HxImgFtorIMNKey(className, sig1Name, sig2Name)
32 {
33 }
```

The documentation for this class was generated from the following file:

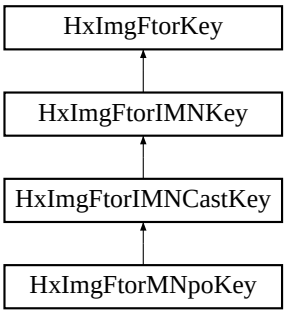
- **HxImgFtorIMNCastKey.h**

6.98 HxImgFtorIMNKey Class Reference

Key for **HxImgFtorIMN** (p. 433).

`#include <HxImgFtorIMNKey.h>`

Inheritance diagram for HxImgFtorIMNKey::



Public Methods

- **HxImgFtorIMNKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)

  Constructor.

6.98.1 Detailed Description

Key for **HxImgFtorIMN** (p. 433).

6.98.2 Constructor & Destructor Documentation

6.98.2.1 HxImgFtorIMNKey::HxImgFtorIMNKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*) [inline]

Constructor.

```
30     : HxImgFtorKey(className)
31 {
32     HxStringList l;
33     l << sig1Name << sig2Name;
34     setArguments(l.begin(), l.end());
35 }
```

The documentation for this class was generated from the following file:

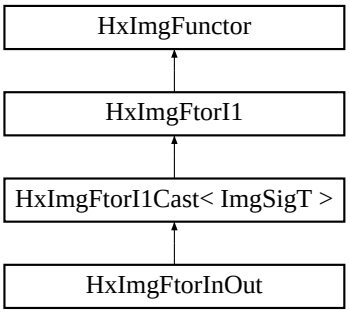
- **HxImgFtorIMNKey.h**

6.99 HxImgFtorInOut Class Template Reference

Instantiation of generic algorithm for in/out pixel operations on images.

`#include <HxImgFtorInOut.h>`

Inheritance diagram for HxImgFtorInOut::



Public Types

- typedef **HxImgFtorInOutKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorInOut** ()

Constructor.

- virtual ~**HxImgFtorInOut** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** ptr, **HxSizes** size, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncInOutPixOpInit***(HxSizes,HxTagList&)* (p. 88) *to dispatch the initialization phase and* **HxFuncInOutPixOp***(DataPtrT,HxSizes,PixOpT&)* (p. 86) *to dispatch the actual work.*

6.99.1 Detailed Description

template<class ImgSigT, class InOutT> class HxImgFtorInOut< ImgSigT, InOutT >

Instantiation of generic algorithm for in/out pixel operations on images.

6.99.2 Member Typedef Documentation

6.99.2.1 template<class ImgSigT, class InOutT> typedef HxImgFtorInOutKey HxImgFtorInOut::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 395).

6.99.3 Constructor & Destructor Documentation

6.99.3.1 template<class ImgSigT, class InOutT> HxImgFtorInOut< ImgSigT, InOutT >::HxImgFtorInOut () [inline]

Constructor.

```
35     : HxImgFtorI1Cast<ImgSigT>(
36         HxImgFtorInOutKey(HxClassName<ImgSigT>(), HxClassName<InOutT>()))
37 {
38     HxImgFtorRuleBase::instance().setIsModifying(
39         "inout", HxClassName<InOutT>(),
40         IsModifying(typename InOutT::DirectionCategory()));
41 }
```

6.99.3.2 template<class ImgSigT, class InOutT> HxImgFtorInOut< ImgSigT, InOutT >::~HxImgFtorInOut () [virtual]

Destructor.

```
45 {
46 }
```

6.99.4 Member Function Documentation

6.99.4.1 template<class ImgSigT, class InOutT> void HxImgFtorInOut< ImgSigT, InOutT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncInOutPixOpInit**(HxSizes,HxTagList&) (p. 88) to dispatch the initialization phase and **HxFuncInOutPixOp**(DataPtrT,HxSizes,PixOpT&) (p. 86) to dispatch the actual work.

Reimplemented from **HxImgFtorI1Cast** (p. 397).

```
53 {
54     if (description) {
55         HxString v(typename InOutT::DirectionCategory().toString());
56         v += ", ";
57         v += typename InOutT::TransVarianceCategory().toString();
58         v += ", ";
59         v += typename InOutT::PhaseCategory().toString();
60         description->setVariation(v);
61     }
62 
63     HxBoundingBox imgBb(size), regionBb(size);
64     regionBb = HxGetTag(tags, "boundingBox", imgBb);
65     regionBb = regionBb.intersect(imgBb);
66 
67     if (!regionBb.isEmpty())
68     {
69         InOutT pixOp =
70             HxFuncInOutPixOpInit<InOutT>(
71                 regionBb.size(), tags, typename InOutT::DirectionCategory(),
72                 typename InOutT::TransVarianceCategory(), typename InOutT::PhaseCategory());
73         ptr.incXYZ(
74             regionBb.begin().x(), regionBb.begin().y(), regionBb.begin().z());
75         HxFuncInOutPixOp(ptr, regionBb.size(), pixOp);
76     }
77 
78 }
```

The documentation for this class was generated from the following files:

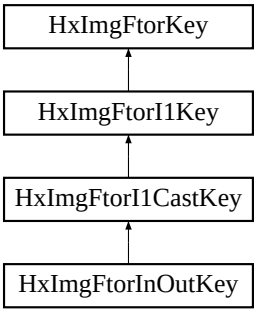
- **HxImgFtorInOut.h**
- HxImgFtorInOut.c

6.100 HxImgFtorInOutKey Class Reference

Key for **HxImgFtorInOut** (p. 440).

`#include <HxImgFtorInOutKey.h>`

Inheritance diagram for HxImgFtorInOutKey::



Public Methods

- **HxImgFtorInOutKey** (**HxString** imgSig, **HxString** inOutName)

  Constructor.

6.100.1 Detailed Description

Key for **HxImgFtorInOut** (p. 440).

6.100.2 Constructor & Destructor Documentation

6.100.2.1 HxImgFtorInOutKey::HxImgFtorInOutKey (HxString *imgSig*, HxString *pixOpName*) [inline]

Constructor.

```
29     : HxImgFtorI1CastKey("HxImgFtorInOut", imgSig)
30 {
31     addArgument(pixOpName);
32 }
```

The documentation for this class was generated from the following file:

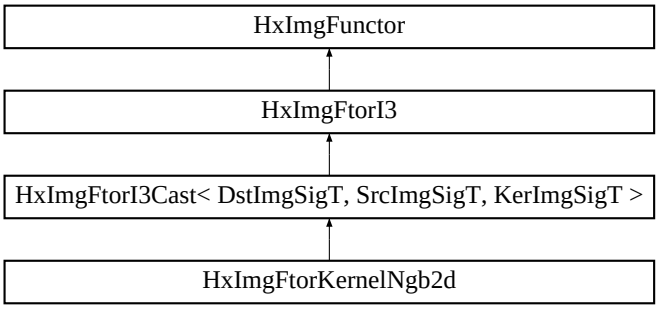
- **HxImgFtorInOutKey.h**

6.101 HxImgFtorKernelNgb2d Class Template Reference

Instantiation of generic algorithm for kernel based neighborhood operations on 2D images.

`#include <HxImgFtorKernelNgb2d.h>`

Inheritance diagram for HxImgFtorKernelNgb2d::



Public Types

- typedef **HxImgFtorKernelNgbKey KeyType**

  The key type of this class.

- typedef SrcImgSigT::DataPtrType **SrcDataPtrType**

  The data pointer type of the source image.

- typedef KerImgSigT::DataPtrType **KerDataPtrType**

  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorKernelNgb2d** ()

  Constructor.

- virtual ∼**HxImgFtorKernelNgb2d** ()

  Destructor.

- virtual **HxSizes minimumBorderSize** (**HxTagList** &tags) const

  Minimum border size.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **KerDataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** ∗=0)

  Calls **HxFuncKernelNgbOp2d** (p. 78) *to do the actual work.*

6.101.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> class HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >

Instantiation of generic algorithm for kernel based neighborhood operations on 2D images.

6.101.2 Member Typedef Documentation

6.101.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> typedef HxImgFtorKernelNgbKey HxImgFtorKernelNgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 420).

6.101.2.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> typedef SrcImgSigT::DataPtrType HxImgFtorKernelNgb2d::SrcDataPtrType

The data pointer type of the source image.

6.101.2.3 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> typedef KerImgSigT::DataPtrType HxImgFtorKernelNgb2d::KerDataPtrType

The data pointer type of the kernel image.

6.101.3 Constructor & Destructor Documentation

6.101.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::HxImgFtorKernelNgb2d () [inline]

Constructor.

```
23         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
24             HxImgFtorKernelNgbKey(
25                 HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
26                 HxClassName<KerImgSigT>(), HxClassName<NgbT>()))
27 {
28     HxImgFtorRuleBase::instance().setResultType(
29         HxClassName<DstImgSigT>(), "kernelNgb",
30         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
31     HxImgFtorRuleBase::instance().setKernelType(
32         HxClassName<KerImgSigT>(), "kernelNgb",
33         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
34 }
```

6.101.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::~HxImgFtorKernelNgb2d () [virtual]

Destructor.

```
39 {
40 }
```

6.101.4 Member Function Documentation

6.101.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> HxSizes HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::minimumBorderSize (HxTagList & *tags*) const [virtual]

Minimum border size.

Reimplemented from **HxImgFunctor** (p. 490).

```
47 {
48     NgbT ngb(tags);
49     return ngb.size() / HxSizes(2,2,2);
50 }
```

6.101.4.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> void HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KerDataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncKernelNgbOp2d** (p. 78) to do the actual work.

```
60 {
61     typedef HxKernel2d<KerDataPtrType, typename KerImgSigT::ArithType> KernelType;
62 
63     NgbT        ngb(tags);
64     KernelType  kernel(kerPtr, kerSize, tags);
65 
66     HxBreakPoint();
67 
68     if (description) {
69         HxString v(typename NgbT::IteratorCategory().toString());
70         v += ", ";
71         v += typename NgbT::PhaseCategory().toString();
72         v += ", ";
73         v += typename NgbT::TransVarianceCategory().toString();
74         description->setVariation(v);
75     }
76 
77     if (kerSize.inf(ngb.size()) != kerSize)
78     {
79         HxEnvironment::instance()->errorStream()
80             << "Error: Kernel neighborhood operator: kernel size smaller than "
81             << "neighborhood size." << STD_ENDL;
82         HxEnvironment::instance()->flush();
83         return;
84     }
85 
86     // pre-condition states that source should be large enough
87 
88     HxSizes borderSize = getBorderSize(
89         tags, ngb.size().sup(srcSize - dstSize)/HxSizes(2, 2, 2));
90 
91     srcPtr.incXYZ(borderSize.x(), borderSize.y(), borderSize.z());
92 
93     HxFuncKernelNgbOp2d(dstPtr, srcPtr, dstSize, ngb, kernel);
94 }
```

The documentation for this class was generated from the following files:

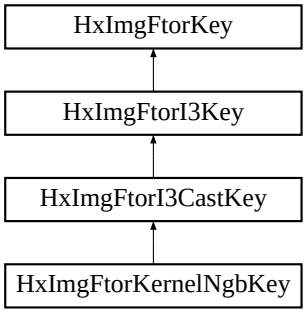
- **HxImgFtorKernelNgb2d.h**
- HxImgFtorKernelNgb2d.c

6.102 HxImgFtorKernelNgbKey Class Reference

Key for HxImgFtorKernelNgb.

`#include <HxImgFtorKernelNgbKey.h>`

Inheritance diagram for HxImgFtorKernelNgbKey::



Public Methods

- **HxImgFtorKernelNgbKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** kerImgSig, **HxString** ngbName)

  Constructor.

6.102.1 Detailed Description

Key for HxImgFtorKernelNgb.

6.102.2 Constructor & Destructor Documentation

6.102.2.1 HxImgFtorKernelNgbKey::HxImgFtorKernelNgbKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *kerImgSig*, HxString *ngbName*) [inline]

Constructor.

```
33     : HxImgFtorI3CastKey("HxImgFtorNgb", dstImgSig, srcImgSig, kerImgSig)
34 {
35     addArgument(ngbName);
36 }
```

The documentation for this class was generated from the following file:

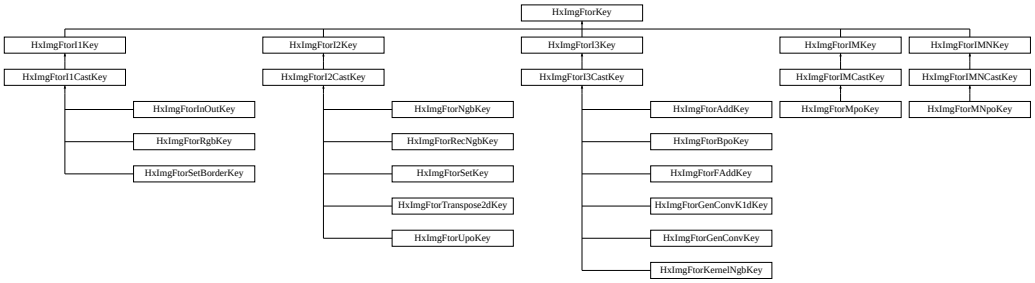
- **HxImgFtorKernelNgbKey.h**

6.103 HxImgFtorKey Class Reference

Base class for keys for image functors.

`#include <HxImgFtorKey.h>`

Inheritance diagram for HxImgFtorKey::



Public Methods

- **HxImgFtorKey** ()
- **HxImgFtorKey** (const HxImgFtorKey &)
- **HxImgFtorKey** (**HxString** className, **HxStringListIter** argListBegin, **HxStringListIter** argListEnd)
- **HxImgFtorKey** (**HxString** className)
- void **addArgument** (**HxString** argName)
- void **setArguments** (**HxStringListIter** argListBegin, **HxStringListIter** argListEnd)
- **HxString getArgument** (int) const
- **HxString getClassName** () const
- **HxString toString** () const
- int **compare** (const HxImgFtorKey &) const
- STD_OSTREAM & **put** (STD_OSTREAM &) const

6.103.1 Detailed Description

Base class for keys for image functors.

The documentation for this class was generated from the following files:

- **HxImgFtorKey.h**
- HxImgFtorKey.c

6.104 HxImgFtorKeyNameTable Class Reference

Table with keys (?).

`#include <HxImgFtorKeyNameTable.h>`

Public Types

- typedef size_t **SizeType**

Public Methods

- **HxString getClassName** (SizeType id)
- SizeType **getClassNameId** (**HxString** name)
- **HxString getName** (SizeType id)
- SizeType **getNameId** (**HxString** name)
- **HxString getTypeName** (SizeType id)
- SizeType **getTypeNameId** (**HxString** name)
- STD_OSTREAM & **put** (STD_OSTREAM &os) const
- **~HxImgFtorKeyNameTable** ()

Static Public Methods

- HxImgFtorKeyNameTable & **instance** ()

6.104.1 Detailed Description

Table with keys (?).

The documentation for this class was generated from the following files:

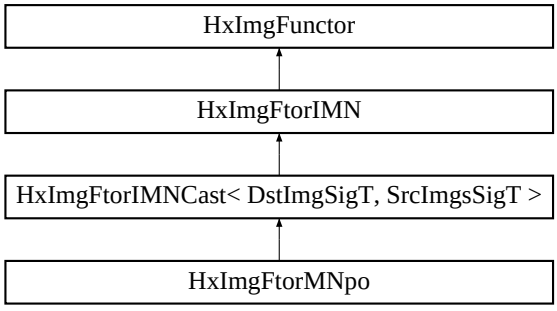
- **HxImgFtorKeyNameTable.h**
- HxImgFtorKeyNameTable.c

6.105 HxImgFtorMNpo Class Template Reference

Instantiation of generic algorithm for M output, N input pixel operations on images.

`#include <HxImgFtorMNpo.h>`

Inheritance diagram for HxImgFtorMNpo::



Public Types

- typedef **HxImgFtorMNpoKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorMNpo** ()

  Constructor.

- virtual bool **probeOp** (**HxTagList** &tags) const

  Probe Operation.

- virtual ~**HxImgFtorMNpo** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrArray** &dstPtrs, **SrcDataPtrArray** &srcPtrs, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncMNPixOp** (p. 82) *to do the actual work.*

6.105.1 Detailed Description

template<class DstImgSigT, class SrcImgsSigT, class MNpoT> class HxImgFtorMNpo< DstImgSigT, SrcImgsSigT, MNpoT >

Instantiation of generic algorithm for M output, N input pixel operations on images.

6.105.2 Member Typedef Documentation

6.105.2.1 template<class DstImgSigT, class SrcImgsSigT, class MNpoT> typedef HxImgFtorMNpoKey HxImgFtorMNpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMN** (p. 434).

6.105.3 Constructor & Destructor Documentation

6.105.3.1 template<class DstImgSigT, class SrcImgsSigT, class MNpoT> HxImgFtorMNpo< DstImgSigT, SrcImgsSigT, MNpoT >::HxImgFtorMNpo ()

Constructor.

```
21     : HxImgFtorIMNCast<DstImgSigT, SrcImgsSigT>(
22         HxImgFtorMNpoKey(HxClassName<DstImgSigT>(), HxClassName<SrcImgsSigT>(),
23             HxClassName<MNpoT>()))
24 {
25 #ifdef CD_TRACE
26     HxEnvironment::instance()->outputStream()
27         << "HxImgFtorMNpo::HxImgFtorMNpo()" << STD_ENDL;
28 #endif
29     static HxRegKey* mpoKey
```

```
30         = HxRegistry::instance().insertKey("/imagefunctortable/mpo");
31
32     HxRegKey* k = mpoKey->insertKey(HxClassName<MNpoT>());
33     k = k->insertKey("resulttype");
34     k->insertValue(
35         HxClassName<SrcImgsSigT>(), HxRegData(HxClassName<DstImgSigT>()));
36 }
```

6.105.3.2 template<class DstImgSigT, class SrcImgsSigT, class MNpoT> HxImgFtorMNpo< DstImgSigT, SrcImgsSigT, MNpoT >::~HxImgFtorMNpo () [virtual]

Destructor.

```
40 {
41 #ifdef CD_TRACE
42     HxEnvironment::instance()->outputStream()
43         << "HxImgFtorMNpo::~HxImgFtorMNpo()" << STD_ENDL;
44 #endif
45 }
```

6.105.4 Member Function Documentation

6.105.4.1 template<class DstImgSigT, class SrcImgsSigT, class MNpoT> bool HxImgFtorMNpo< DstImgSigT, SrcImgsSigT, MNpoT >::probeOp (HxTagList & *tags*) const [virtual]

Probe Operation.

Reimplemented from **HxImgFunctor** (p. 491).

```
50 {
51     MNpoT mpo(tags);
52
53     return HxGetTag(tags, "preOpIsOk", true);
54 }
```

6.105.4.2 template<class DstImgSigT, class SrcImgsSigT, class MNpoT> void HxImgFtorMNpo< DstImgSigT, SrcImgsSigT, MNpoT >::doIt (DstDataPtrArray & *dstPtrs*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncMNPixOp** (p. 82) to do the actual work.

Reimplemented from **HxImgFtorIMNCast** (p. 438).

```
61 {
62     HxAddTag(tags, "sourceCnt", srcPtrs.size());
63     MNpoT mpo(tags);
64
65     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
66     HxFuncMNPixOp(
67         dstPtrs, srcPtrs, nPix, mpo);
68 }
```

The documentation for this class was generated from the following files:

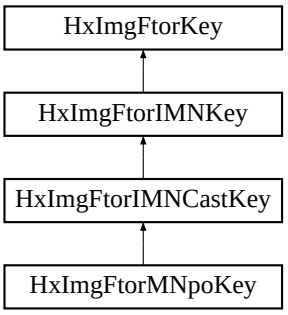
- **HxImgFtorMNpo.h**
- HxImgFtorMNpo.c

6.106 HxImgFtorMNpoKey Class Reference

Key for **HxImgFtorMNpo** (p. 449).

`#include <HxImgFtorMNpoKey.h>`

Inheritance diagram for HxImgFtorMNpoKey::



Public Methods

- **HxImgFtorMNpoKey** (**HxString** dstImgsSig, **HxString** srcImgsSig, **HxString** mpoName)

  Constructor.

6.106.1 Detailed Description

Key for **HxImgFtorMNpo** (p. 449).

6.106.2 Constructor & Destructor Documentation

6.106.2.1 HxImgFtorMNpoKey::HxImgFtorMNpoKey (HxString *dstImgsSig*, HxString *srcImgsSig*, HxString *mpoName*) [inline]

Constructor.

```
31     : HxImgFtorIMNCastKey("HxImgFtorMNpo", dstImgsSig, srcImgsSig)
32 {
33     addArgument(mpoName);
34 }
```

The documentation for this class was generated from the following file:

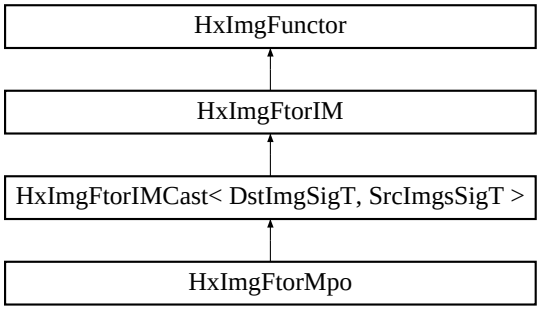
- **HxImgFtorMNpoKey.h**

6.107 HxImgFtorMpo Class Template Reference

Instantiation of generic algorithm for multi pixel operations on images.

`#include <HxImgFtorMpo.h>`

Inheritance diagram for HxImgFtorMpo::



Public Types

- typedef **HxImgFtorMpoKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorMpo** ()

  Constructor.

- virtual ~**HxImgFtorMpo** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrArray** &srcPtrs, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncMultiPixOp** (p. 82) *to do the actual work.*

6.107.1 Detailed Description

template<class DstImgSigT, class SrcImgsSigT, class MpoT> class HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >

Instantiation of generic algorithm for multi pixel operations on images.

6.107.2 Member Typedef Documentation

6.107.2.1 template<class DstImgSigT, class SrcImgsSigT, class MpoT> typedef HxImgFtorMpoKey HxImgFtorMpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMCast** (p. 429).

6.107.3 Constructor & Destructor Documentation

6.107.3.1 template<class DstImgSigT, class SrcImgsSigT, class MpoT> HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >::HxImgFtorMpo () [inline]

Constructor.

```
22     : HxImgFtorIMCast<DstImgSigT, SrcImgsSigT>(
23         HxImgFtorMpoKey(HxClassName<DstImgSigT>(), HxClassName<SrcImgsSigT>(),
24             HxClassName<MpoT>()))
25 {
26 #ifdef CD_TRACE
27     HxEnvironment::instance()->outputStream()
28         << "HxImgFtorMpo::HxImgFtorMpo()" << STD_ENDL;
29 #endif
30     static HxRegKey* mpoKey
31         = HxRegistry::instance().insertKey("/imagefunctortable/mpo");
32
33     HxRegKey* k = mpoKey->insertKey(HxClassName<MpoT>());
34     k = k->insertKey("resulttype");
35     k->insertValue(
36         HxClassName<SrcImgsSigT>(), HxRegData(HxClassName<DstImgSigT>()));
37 }
```

6.107.3.2 template<class DstImgSigT, class SrcImgsSigT, class MpoT> HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >::~HxImgFtorMpo () [virtual]

Destructor.

```
41 {
42 #ifdef CD_TRACE
43     HxEnvironment::instance()->outputStream()
44         << "HxImgFtorMpo::~HxImgFtorMpo()" << STD_ENDL;
45 #endif
46 }
```

6.107.4 Member Function Documentation

6.107.4.1 template<class DstImgSigT, class SrcImgsSigT, class MpoT> void HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncMultiPixOp** (p. 82) to do the actual work.

Reimplemented from **HxImgFtorIMCast** (p. 431).

```
53 {
54     HxAddTag(tags, "sourceCnt", srcPtrs.size());
55     MpoT mpo(tags);
56 
57     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
58     HxFuncMultiPixOp(
59         dstPtr, srcPtrs, nPix, mpo);
60 }
```

The documentation for this class was generated from the following files:

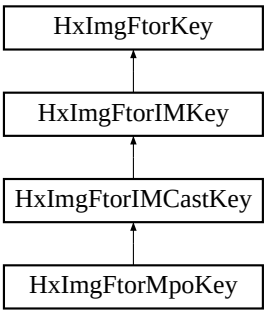
- **HxImgFtorMpo.h**
- HxImgFtorMpo.c

6.108 HxImgFtorMpoKey Class Reference

Key for **HxImgFtorMpo** (p. 452).

`#include <HxImgFtorMpoKey.h>`

Inheritance diagram for HxImgFtorMpoKey::



Public Methods

- **HxImgFtorMpoKey** (**HxString** dstImgSig, **HxString** srcImgsSig, **HxString** mpoName)

  Constructor.

6.108.1 Detailed Description

Key for **HxImgFtorMpo** (p. 452).

6.108.2 Constructor & Destructor Documentation

6.108.2.1 HxImgFtorMpoKey::HxImgFtorMpoKey (HxString *dstImgSig*, HxString *srcImgsSig*, HxString *mpoName*) [inline]

Constructor.

```
31     : HxImgFtorIMCastKey("HxImgFtorMpo", dstImgSig, srcImgsSig)
32 {
33     addArgument(mpoName);
34 }
```

The documentation for this class was generated from the following file:

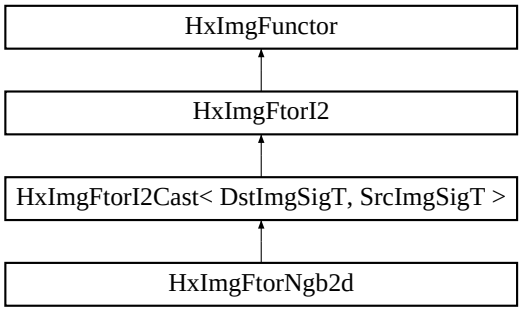
- **HxImgFtorMpoKey.h**

6.109 HxImgFtorNgb2d Class Template Reference

Instantiation of generic algorithm for neighborhood operations on 2D images.

`#include <HxImgFtorNgb2d.h>`

Inheritance diagram for HxImgFtorNgb2d::



Public Types

- typedef **HxImgFtorNgbKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorNgb2d** ()

  Constructor.

- virtual ~**HxImgFtorNgb2d** ()

  Destructor.

- virtual **HxSizes minimumBorderSize** (**HxTagList** &tags) const

  Minimum border size.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

Calls **HxFuncNgbOp2d***(DstDataPtrT,SrcDataPtrT,HxSizes,NgbT&)* (p. 79) *to do the actual work.*

6.109.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class NgbT> class HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >

Instantiation of generic algorithm for neighborhood operations on 2D images.

6.109.2 Member Typedef Documentation

6.109.2.1 template<class DstImgSigT, class SrcImgSigT, class NgbT> typedef HxImgFtorNgbKey HxImgFtorNgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 407).

6.109.3 Constructor & Destructor Documentation

6.109.3.1 template<class DstImgSigT, class SrcImgSigT, class NgbT> HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::HxImgFtorNgb2d () [inline]

Constructor.

```
21     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
22         HxImgFtorNgbKey(
23             HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
24             HxClassName<NgbT>()))
25 {
26     HxImgFtorRuleBase::instance().setResultType(
27         HxClassName<DstImgSigT>(), "ngb",
28         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
29
30 }
```

6.109.3.2 template<class DstImgSigT, class SrcImgSigT, class NgbT> HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::~HxImgFtorNgb2d () [virtual]

Destructor.

```
34 {
35 }
```

6.109.4 Member Function Documentation

6.109.4.1 template<class DstImgSigT, class SrcImgSigT, class NgbT> HxSizes HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::minimumBorderSize (HxTagList & *tags*) const [virtual]

Minimum border size.

Reimplemented from **HxImgFunctor** (p. 490).

```
41 {
42     NgbT ngb(tags);
43     return ngb.size() / HxSizes(2,2,2);
44 }
```

6.109.4.2 template<class DstImgSigT, class SrcImgSigT, class NgbT> void HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncNgbOp2d**(DstDataPtrT,SrcDataPtrT,HxSizes,NgbT&) (p. 79) to do the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 411).

```
52 {
53     NgbT ngb(tags);
54 
55     if (description) {
56         HxString v(typename NgbT::IteratorCategory().toString());
57         v += ", ";
58         v += typename NgbT::PhaseCategory().toString();
59         v += ", ";
60         v += typename NgbT::TransVarianceCategory().toString();
61         description->setVariation(v);
62     }
63 
64     HxSizes borderSize = getBorderSize(
65         tags, ngb.size().sup(srcSize - dstSize)/ HxSizes(2,2,2));
66 
67     srcPtr.incXYZ(borderSize.x(), borderSize.y(), borderSize.z());
68 
69     HxFuncNgbOp2d(dstPtr, srcPtr, dstSize, ngb);
70 }
```

The documentation for this class was generated from the following files:

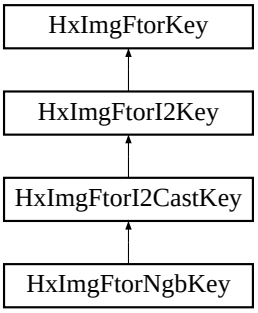
- **HxImgFtorNgb2d.h**
- HxImgFtorNgb2d.c

6.110 HxImgFtorNgbKey Class Reference

Key for HxImgFtorNgb.

`#include <HxImgFtorNgbKey.h>`

Inheritance diagram for HxImgFtorNgbKey::



Public Methods

- **HxImgFtorNgbKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** ngbName)

  Constructor.

6.110.1 Detailed Description

Key for HxImgFtorNgb.

6.110.2 Constructor & Destructor Documentation

6.110.2.1 HxImgFtorNgbKey::HxImgFtorNgbKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *ngbName*) [inline]

Constructor.

```
31     : HxImgFtorI2CastKey("HxImgFtorNgb2d", dstImgSig, srcImgSig)
32 {
33     addArgument(ngbName);
34 }
```

The documentation for this class was generated from the following file:

- **HxImgFtorNgbKey.h**

6.111 HxImgFtorObserver Class Reference

Image functor observer.

`#include <HxImgFtorObserver.h>`

Public Methods

- virtual int **inserted** (const **HxImgFtorKey** &, const **HxImgFunctor** &)=0

6.111.1 Detailed Description

Image functor observer.

The documentation for this class was generated from the following file:

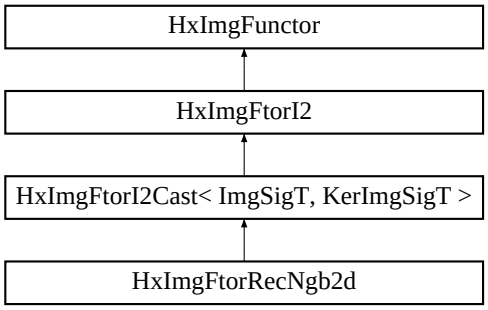
- **HxImgFtorObserver.h**

6.112 HxImgFtorRecNgb2d Class Template Reference

Instantiation of generic algorithm for recursive neighborhood operations on 2D images.

`#include <HxImgFtorRecNgb2d.h>`

Inheritance diagram for HxImgFtorRecNgb2d::



Public Types

- typedef **HxImgFtorRecNgbKey KeyType**

  The key type of this class.

- typedef ImgSigT::DataPtrType **DataPtrType**

  The data pointer type of the source image.

- typedef KerImgSigT::DataPtrType **KerDataPtrType**

  The data pointer type of the kernel image.

Public Methods

- **HxImgFtorRecNgb2d** ()

  Constructor.

- virtual ~**HxImgFtorRecNgb2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DataPtrType** imgPtr, **KerDataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls recNgb, recNgbKx, or recNgbKy to do the actual work based on the "dimension" tag.

- virtual void **recNgb** (**DataPtrType** imgPtr, **KerDataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags)
- virtual void **recNgbKx** (**DataPtrType** imgPtr, **KerDataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags)
- virtual void **recNgbKy** (**DataPtrType** imgPtr, **KerDataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags)

6.112.1 Detailed Description

template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> class HxImgFtorRecNgb2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >

Instantiation of generic algorithm for recursive neighborhood operations on 2D images.

6.112.2 Member Typedef Documentation

6.112.2.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> typedef HxImgFtorRecNgbKey HxImgFtorRecNgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 407).

6.112.2.2 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> typedef ImgSigT::DataPtrType HxImgFtorRecNgb2d::DataPtrType

The data pointer type of the source image.

6.112.2.3 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> typedef KerImgSigT::DataPtrType HxImgFtorRecNgb2d::KerDataPtrType

The data pointer type of the kernel image.

6.112.3 Constructor & Destructor Documentation

6.112.3.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecNgb2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::HxImgFtorRecNgb2d ()

Constructor.

```
27         : HxImgFtorI2Cast<ImgSigT, KerImgSigT>(
28             HxImgFtorRecNgbKey(
29                 HxClassName<ImgSigT>(), HxClassName<KerImgSigT>(),
```

```
30                 HxClassName<PixOpT>(), HxClassName<RedOpT>()))
31 {
32 #ifdef CD_TRACE
33     HxEnvironment::instance()->outputStream()
34         << "HxImgFtorRecNgb2d::HxImgFtorRecNgb2d()" << STD_ENDL;
35 #endif
36 }
```

6.112.3.2 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecNgb2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::~HxImgFtorRecNgb2d () [virtual]

Destructor.

```
40 {
41 #ifdef CD_TRACE
42     HxEnvironment::instance()->outputStream()
43         << "HxImgFtorRecNgb2d::~HxImgFtorRecNgb2d()" << STD_ENDL;
44 #endif
45 }
```

6.112.4 Member Function Documentation

6.112.4.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> void HxImgFtorRecNgb2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::doIt (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, HxSizes *imgSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * = 0) [protected, virtual]

Calls recNgb, recNgbKx, or recNgbKy to do the actual work based on the ”dimension” tag.

```
432 {
433     int dimension = HxGetTag(tags, "dimension", 0);
434
435     switch(dimension)
436     {
437     case 0  :
438         recNgb(imgPtr, kerPtr, imgSize, kerSize, tags);
439         break;
440     case 1  :
441         recNgbKx(imgPtr, kerPtr, imgSize, kerSize, tags);
442         break;
443     case 2  :
444         recNgbKy(imgPtr, kerPtr, imgSize, kerSize, tags);
445         break;
446     default :
447         HxEnvironment::instance()->errorStream()
448             << "Recursive generalized convolution (2d image, 1d kernel): "
449             << "cannot execute convolution in dimension " << dimension
450             << STD_ENDL;
451         HxEnvironment::instance()->flush();
452     }
453 }
```

The documentation for this class was generated from the following files:

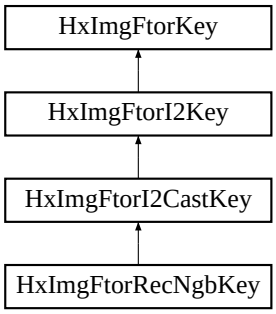
- **HxImgFtorRecNgb2d.h**
- HxImgFtorRecNgb2d.c

6.113 HxImgFtorRecNgbKey Class Reference

Key for HxImgFtorRecNgb.

`#include <HxImgFtorRecNgbKey.h>`

Inheritance diagram for HxImgFtorRecNgbKey::



Public Methods

- **HxImgFtorRecNgbKey** (**HxString** imgSig, **HxString** kerImgSig, **HxString** pixOpName, **HxString** redOpName)

  Constructor.

6.113.1 Detailed Description

Key for HxImgFtorRecNgb.

6.113.2 Constructor & Destructor Documentation

6.113.2.1 HxImgFtorRecNgbKey::HxImgFtorRecNgbKey (HxString *imgSig*, HxString *kerImgSig*, HxString *pixOpName*, HxString *redOpName*) [inline]

Constructor.

```
32         : HxImgFtorI2CastKey("HxImgFtorRecNgb", imgSig, kerImgSig)
33 {
34     addArgument(pixOpName);
35     addArgument(redOpName);
36 }
```

The documentation for this class was generated from the following file:

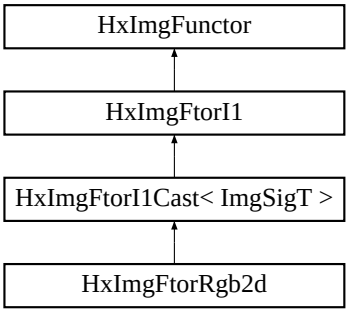
- **HxImgFtorRecNgbKey.h**

6.114 HxImgFtorRgb2d Class Template Reference

Instantiation of generic algorithm for display of 2d images.

`#include <HxImgFtorRgb2d.h>`

Inheritance diagram for HxImgFtorRgb2d::



Public Types

- typedef **HxImgFtorRgbKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorRgb2d** ()

  Constructor.

- virtual ~**HxImgFtorRgb2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** ptr, **HxSizes** size, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.114.1 Detailed Description

template<class ImgSigT, class RgbT> class HxImgFtorRgb2d< ImgSigT, RgbT >

Instantiation of generic algorithm for display of 2d images.

6.114.2 Member Typedef Documentation

6.114.2.1 template<class ImgSigT, class RgbT> typedef HxImgFtorRgbKey HxImgFtorRgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 395).

6.114.3 Constructor & Destructor Documentation

6.114.3.1 template<class ImgSigT, class RgbT> HxImgFtorRgb2d< ImgSigT, RgbT >::HxImgFtorRgb2d () [inline]

Constructor.

```
19     : HxImgFtorI1Cast<ImgSigT>(HxImgFtorRgbKey(HxClassName<ImgSigT>(),
20                                HxClassName<RgbT>()))
21 {
22 #ifdef CD_TRACE
23     HxEnvironment::instance()->outputStream()
24         << "HxImgFtorRgb2d::HxImgFtorRgb2d()" << STD_ENDL;
25 #endif
26     static HxRegKey* surKey
27         = HxRegistry::instance().insertKey("/imagefunctortable/rgb");
28
29     HxRegKey* k = surKey->insertKey(HxClassName<RgbT>());
30     k = k->insertKey("imagetype");
31     k->insertValue(
32         HxClassName<ImgSigT>(), HxRegData(HxClassName<ImgSigT>()));
33 }
```

6.114.3.2 template<class ImgSigT, class RgbT> HxImgFtorRgb2d< ImgSigT, RgbT >::~HxImgFtorRgb2d () [virtual]

Destructor.

```
37 {
38 #ifdef CD_TRACE
39     HxEnvironment::instance()->outputStream()
40         << "HxImgFtorRgb2d::~HxImgFtorRgb2d()" << STD_ENDL;
41 #endif
42 }
```

6.114.4 Member Function Documentation

6.114.4.1 template<class ImgSigT, class RgbT> void HxImgFtorRgb2d< ImgSigT, RgbT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

The actual operation.

Reimplemented from **HxImgFtorI1Cast** (p. 397).

```
48 {
49     int* pixels = HxGetTag<int*>(tags, "pixels");
50     int resWidth = HxGetTag<int>(tags, "resWidth");
51     int resHeight = HxGetTag<int>(tags, "resHeight");
52     HxGeoIntType gi = HxGetTag<HxGeoIntType>(tags, "gi");
53     int width = size.x();
54     int height = size.y();
55
```

```
56      RgbT rgbOp(tags);
57
58      if (((resWidth == -1) && (resHeight == -1)) ||
59                  ((resWidth == width) && (resHeight == height))) {
60          int nPix = width * height;
61          ImgDataPtrType p = ptr;
62          while (--nPix >= 0) {
63              *pixels++ = rgbOp.doIt(p.read());
64              p.incX();
65          }
66          return;
67      }
68
69      double sx = (double) width / resWidth;
70      double sy = (double) height / resHeight;
71      if (gi == NEAREST) {
72          for (int y=0 ; y<resHeight ; y++) {
73              for (int x=0 ; x<resWidth ; x++) {
74                  ImgDataPtrType p = ptr;
75                  p.incXYZ((int) (sx * x + 0.5), (int) (sy * y + 0.5), 0);
76                  *pixels++ = rgbOp.doIt(p.read());
77              }
78          }
79          return;
80      }
81
82      TYPENAME ImgSigT::ArithTypeDouble result;
83      for (int y=0 ; y<resHeight ; y++) {
84          for (int x=0 ; x<resWidth ; x++) {
85              result = HxFunc2dSample(ptr, result, sx * x, sy * y, 0, gi);
86              *pixels++ = rgbOp.doItDouble(result);
87          }
88      }
89  }
```

The documentation for this class was generated from the following files:

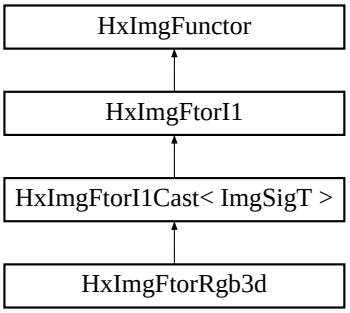
- **HxImgFtorRgb2d.h**
- HxImgFtorRgb2d.c

6.115 HxImgFtorRgb3d Class Template Reference

Instantiation of generic algorithm for display of 3d images.

`#include <HxImgFtorRgb3d.h>`

Inheritance diagram for HxImgFtorRgb3d::



Public Types

- typedef **HxImgFtorRgbKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorRgb3d** ()

  Constructor.

- virtual ~**HxImgFtorRgb3d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** ptr, **HxSizes** size, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.115.1 Detailed Description

template<class ImgSigT, class RgbT> class HxImgFtorRgb3d< ImgSigT, RgbT >

Instantiation of generic algorithm for display of 3d images.

6.115.2 Member Typedef Documentation

6.115.2.1 template<class ImgSigT, class RgbT> typedef HxImgFtorRgbKey HxImgFtorRgb3d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 395).

6.115.3 Constructor & Destructor Documentation

6.115.3.1 template<class ImgSigT, class RgbT> HxImgFtorRgb3d< ImgSigT, RgbT >::HxImgFtorRgb3d () [inline]

Constructor.

```
19     : HxImgFtorI1Cast<ImgSigT>(HxImgFtorRgbKey(HxClassName<ImgSigT>(),
20                                HxClassName<RgbT>()))
21 {
22 #ifdef CD_TRACE
23     HxEnvironment::instance()->outputStream()
24         << "HxImgFtorRgb3d::HxImgFtorRgb3d()" << STD_ENDL;
```

```
25 #endif
26     static HxRegKey* surKey
27         = HxRegistry::instance().insertKey("/imagefunctortable/rgb");
28
29     HxRegKey* k = surKey->insertKey(HxClassName<RgbT>());
30     k = k->insertKey("imagetype");
31     k->insertValue(
32         HxClassName<ImgSigT>(), HxRegData(HxClassName<ImgSigT>()));
33 }
```

6.115.3.2 template<class ImgSigT, class RgbT> HxImgFtorRgb3d< ImgSigT, RgbT >::~HxImgFtorRgb3d () [virtual]

Destructor.

```
37 {
38 #ifdef CD_TRACE
39     HxEnvironment::instance()->outputStream()
40         << "HxImgFtorRgb3d::~HxImgFtorRgb3d()" << STD_ENDL;
41 #endif
42 }
```

6.115.4 Member Function Documentation

6.115.4.1 template<class ImgSigT, class RgbT> void HxImgFtorRgb3d< ImgSigT, RgbT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

The actual operation.

Reimplemented from **HxImgFtorI1Cast** (p. 397).

```
48 {
49     int* pixels = HxGetTag<int*>(tags, "pixels");
50     int dimension = HxGetTag<int>(tags, "dimension");
51     int coordinate = HxGetTag<int>(tags, "coordinate");
52     int resWidth = HxGetTag<int>(tags, "resWidth");
53     int resHeight = HxGetTag<int>(tags, "resHeight");
54     HxGeoIntType gi = HxGetTag<HxGeoIntType>(tags, "gi");
55
56     RgbT rgbOp(tags);
57
58     int x, y, z;
59     HxSizes planeSize;
60     switch (dimension) {
61     case 1:
62         planeSize = HxSizes(size.y(), size.z(), 1);
63         break;
64     case 2:
65         planeSize = HxSizes(size.x(), size.z(), 1);
66         break;
67     case 3:
68         planeSize = HxSizes(size.x(), size.y(), 1);
69         break;
70     }
71
72     if (((resWidth == -1) && (resHeight == -1)) ||
73             ((resWidth == planeSize.x()) && (resHeight == planeSize.y()))) {
```

```
74          ImgDataPtrType p = ptr;
75          switch (dimension) {
76          case 1:
77              for (z=0 ; z<size.z() ; z++) {
78                  p = ptr;
79                  p.incXYZ(coordinate, 0, z);
80                  for (y=0 ; y<size.y() ; y++) {
81                      *pixels++ = rgbOp.doIt(p.read());
82                      p.incY();
83                  }
84              }
85              break;
86          case 2:
87              for (z=0 ; z<size.z() ; z++) {
88                  p = ptr;
89                  p.incXYZ(0, coordinate, z);
90                  for (x=0 ; x<size.x() ; x++) {
91                      *pixels++ = rgbOp.doIt(p.read());
92                      p.incX();
93                  }
94              }
95              break;
96          case 3:
97              ptr.incZ(coordinate);
98              int nPix = size.x() * size.y();
99              while (--nPix >= 0) {
100                  *pixels++ = rgbOp.doIt(ptr.read());
101                  ptr.incX();
102              }
103              break;
104          }
105          return;
106      }
107
108      double sx = (double) planeSize.x() / resWidth;
109      double sy = (double) planeSize.y() / resHeight;
110      TYPENAME ImgSigT::ArithTypeDouble result;
111      switch (dimension) {
112      case 1:
113          for (y=0 ; y<resHeight ; y++) {
114              for (x=0 ; x<resWidth ; x++) {
115                  result = HxFunc3dSample(ptr, result, coordinate, sx * x, sy * y, gi);
116                  *pixels++ = rgbOp.doItDouble(result);
117              }
118          }
119          break;
120      case 2:
121          for (y=0 ; y<resHeight ; y++) {
122              for (x=0 ; x<resWidth ; x++) {
123                  result = HxFunc3dSample(ptr, result, sx * x, coordinate, sy * y, gi);
124                  *pixels++ = rgbOp.doItDouble(result);
125              }
126          }
127          break;
128      case 3:
129          for (y=0 ; y<resHeight ; y++) {
130              for (x=0 ; x<resWidth ; x++) {
131                  result = HxFunc3dSample(ptr, result, sx * x, sy * y, coordinate, gi);
132                  *pixels++ = rgbOp.doItDouble(result);
133              }
134          }
135          break;
136      }
137  }
```

The documentation for this class was generated from the following files:

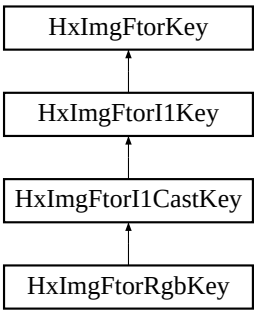
- **HxImgFtorRgb3d.h**
- HxImgFtorRgb3d.c

6.116 HxImgFtorRgbKey Class Reference

Key for HxImgFtorRgb.

`#include <HxImgFtorRgbKey.h>`

Inheritance diagram for HxImgFtorRgbKey::



Public Methods

- **HxImgFtorRgbKey** (**HxString** imgSig, **HxString** rgbName)

  Constructor.

6.116.1 Detailed Description

Key for HxImgFtorRgb.

6.116.2 Constructor & Destructor Documentation

6.116.2.1 HxImgFtorRgbKey::HxImgFtorRgbKey (HxString *imgSig*, HxString *rgbName*) [inline]

Constructor.

```
29     : HxImgFtorI1CastKey("HxImgFtorRgb", imgSig)
30 {
31     addArgument(rgbName);
32 }
```

The documentation for this class was generated from the following file:

- **HxImgFtorRgbKey.h**

6.117 HxImgFtorRuleBase Class Reference

Rule base.

`#include <HxImgFtorRuleBase.h>`

Public Types

- typedef HxIfRbPair **QueryResultType**

Public Methods

- void **setResultType** (**HxString** resultType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- void **setResultType** (**HxString** resultType, **HxString** ftorClass, **HxString** src1Type, **HxString** src2Type, **HxString** opName)
- QueryResultType **getResultType** (**HxImageSignature** defaultResult, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- QueryResultType **getResultType** (**HxImageSignature** defaultResult, **HxString** ftorClass, **HxString** src1Type, **HxString** src2Type, **HxString** opName)
- void **setArgumentType** (**HxString** argumentType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- QueryResultType **getArgumentType** (**HxImageSignature** defaultArgument, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- void **setKernelType** (**HxString** kernelType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- QueryResultType **getKernelType** (**HxImageSignature** defaultKernel, **HxString** ftorClass, **HxString** srcType, **HxString** opName)
- void **setIsModifying** (**HxString** ftorClass, **HxString** opName, bool f=true)
- bool **getIsModifying** (**HxString** ftorClass, **HxString** opName)
- **~HxImgFtorRuleBase** ()

Static Public Methods

- HxImgFtorRuleBase & **instance** ()

6.117.1 Detailed Description

Rule base.

The documentation for this class was generated from the following files:

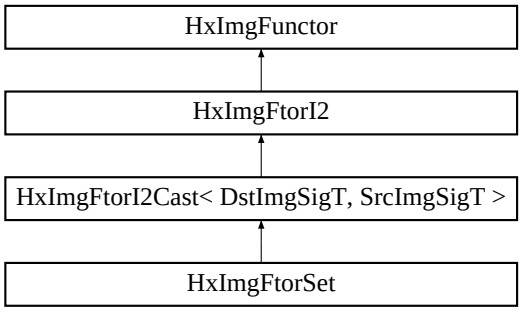
- **HxImgFtorRuleBase.h**
- HxImgFtorRuleBase.c

6.118 HxImgFtorSet Class Template Reference

Instantiation of generic algorithm for set operations on images.

`#include <HxImgFtorSet.h>`

Inheritance diagram for HxImgFtorSet::



Public Types

- typedef **HxImgFtorSetKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorSet** ()

  Constructor.

- virtual ~**HxImgFtorSet** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **HxSizes** srcSize, **HxSizes** dstSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.118.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT> class HxImgFtorSet< DstImgSigT, SrcImgSigT >

Instantiation of generic algorithm for set operations on images.

6.118.2 Member Typedef Documentation

6.118.2.1 template<class DstImgSigT, class SrcImgSigT> typedef HxImgFtorSetKey HxImgFtorSet::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 407).

6.118.3 Constructor & Destructor Documentation

6.118.3.1 template<class DstImgSigT, class SrcImgSigT> HxImgFtorSet< DstImgSigT, SrcImgSigT >::HxImgFtorSet () [inline]

Constructor.

```
21     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
22         HxImgFtorSetKey(HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>()))
23 {
24 }
```

6.118.3.2 template<class DstImgSigT, class SrcImgSigT> HxImgFtorSet< DstImgSigT, SrcImgSigT >::~HxImgFtorSet () [virtual]

Destructor.

```
28 {
29 }
```

6.118.4 Member Function Documentation

6.118.4.1 template<class DstImgSigT, class SrcImgSigT> void HxImgFtorSet< DstImgSigT, SrcImgSigT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

The actual operation.

Reimplemented from **HxImgFtorI2Cast** (p. 411).

```
68 {
69     HxPointInt srcBegin, dstBegin, srcEnd, dstEnd;
70 
71 
72     srcBegin = HxGetTag(tags, "srcBegin", HxPointInt(0, 0, 0));
73     srcEnd = HxGetTag(tags, "srcEnd", HxPointInt(srcSize - HxSizes(1, 1, 1)));
74     dstBegin = HxGetTag(tags, "dstBegin", HxPointInt(0, 0, 0));
75     dstEnd = dstSize - HxSizes(1, 1, 1);
76 
77 
78     if ((srcBegin.inf(HxPointInt(0, 0, 0)) != HxPointInt(0, 0, 0))
79             || (srcBegin.sup(srcEnd) != srcEnd))
80     {
81         HxEnvironment::instance()->errorStream()
```

```
82              << "Extended set: source begin out of range" << STD_ENDL;
83          return;
84      }
85
86      if ((dstBegin.inf(HxPointInt(0, 0, 0)) != HxPointInt(0, 0, 0))
87              || (dstBegin.sup(dstEnd) != dstEnd))
88      {
89          HxEnvironment::instance()->errorStream()
90              << "Extended set: destination begin out of range" << STD_ENDL;
91          return;
92      }
93
94      if ((srcEnd.inf(srcBegin) != srcBegin)
95              || (srcEnd.sup(srcSize - HxSizes(1, 1, 1))
96                  != (srcSize - HxSizes(1, 1, 1))))
97      {
98          HxEnvironment::instance()->errorStream()
99              << "Extended set: source end out of range" << STD_ENDL;
100         return;
101     }
102
103     HxSizes regionSize = HxSizes(srcEnd - srcBegin) + HxSizes(1, 1, 1);
104     regionSize = regionSize.inf(dstSize - HxSizes(dstBegin));
105
106     srcPtr.incXYZ(srcBegin.x(), srcBegin.y(), srcBegin.z());
107     dstPtr.incXYZ(dstBegin.x(), dstBegin.y(), dstBegin.z());
108
109     int y, z;
110
111     int lineSize = regionSize.x();
112
113     for (z = 0; z < regionSize.z(); z++)
114         for (y = 0; y < regionSize.y(); y++)
115         {
116             SrcDataPtrType sPtr(srcPtr);
117             DstDataPtrType dPtr(dstPtr);
118             sPtr.incXYZ(0, y, z);
119             dPtr.incXYZ(0, y, z);
120             HxFuncSet(dPtr, sPtr, lineSize);
121         }
122
123 }
```

The documentation for this class was generated from the following files:

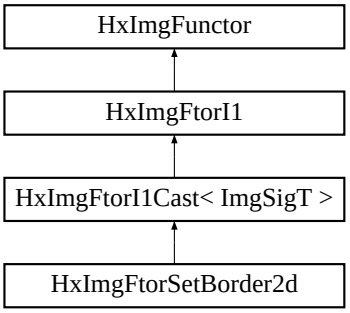
- **HxImgFtorSet.h**
- HxImgFtorSet.c

6.119 HxImgFtorSetBorder2d Class Template Reference

Instantiation of generic algorithm for border operations on 2D images.

`#include <HxImgFtorSetBorder2d.h>`

Inheritance diagram for HxImgFtorSetBorder2d::



Public Types

- typedef **HxImgFtorSetBorderKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorSetBorder2d** ()

  Constructor.

- virtual ~**HxImgFtorSetBorder2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** imgPtr, **HxSizes** imgSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls one of the functions below to do the actual work based on the "borderType" tag: **HxFuncBorderConstant2d** (p. 73), **HxFuncBorderMirror2d** (p. 70) *(default)*, **HxFuncBorderPropagate2d** (p. 75).

6.119.1 Detailed Description

template<class ImgSigT> class HxImgFtorSetBorder2d< ImgSigT >

Instantiation of generic algorithm for border operations on 2D images.

6.119.2 Member Typedef Documentation

6.119.2.1 template<class ImgSigT> typedef HxImgFtorSetBorderKey HxImgFtorSetBorder2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 395).

6.119.3 Constructor & Destructor Documentation

6.119.3.1 template<class ImgSigT> HxImgFtorSetBorder2d< ImgSigT >::HxImgFtorSetBorder2d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(
21         HxImgFtorSetBorderKey(HxClassName<ImgSigT>()))
22 {
23 }
```

6.119.3.2 template<class ImgSigT> HxImgFtorSetBorder2d< ImgSigT >::~HxImgFtorSetBorder2d () [virtual]

Destructor.

```
27 {
28 }
```

6.119.4 Member Function Documentation

6.119.4.1 template<class ImgSigT> void HxImgFtorSetBorder2d< ImgSigT >::doIt (ImgDataPtrType *imgPtr*, HxSizes *imgSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls one of the functions below to do the actual work based on the ”borderType” tag: **HxFuncBorderConstant2d** (p. 73), **HxFuncBorderMirror2d** (p. 70) (default), **HxFuncBorderPropagate2d** (p. 75).

Reimplemented from **HxImgFtorI1Cast** (p. 397).

```
36 {
37     HxBorderType borderType
38         = HxGetTag(tags, "borderType", HxBorderType(HXBORDERMIRROR));
39     HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0, 0, 0));
40
41     switch (borderType)
42     {
43     case HXBORDERCONSTANT   :
44         {
45             HxValue value = HxGetTag(tags, "borderValue", HxValue(0));
46             HxFuncBorderConstant2d(imgPtr, imgSize, borderSize, value);
47         }
48         break;
49     case HXBORDERMIRROR     :
50         HxFuncBorderMirror2d(imgPtr, imgSize, borderSize);
51         break;
52     case HXBORDERPROPAGATE  :
53         HxFuncBorderPropagate2d(imgPtr, imgSize, borderSize);
54         break;
55     }
56 }
```

The documentation for this class was generated from the following files:

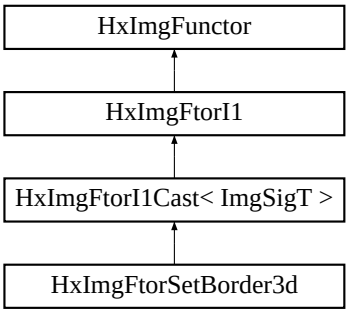
- **HxImgFtorSetBorder2d.h**
- HxImgFtorSetBorder2d.c

6.120 HxImgFtorSetBorder3d Class Template Reference

Instantiation of generic algorithm for border operations on 3D images.

`#include <HxImgFtorSetBorder3d.h>`

Inheritance diagram for HxImgFtorSetBorder3d::



Public Types

- typedef **HxImgFtorSetBorderKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorSetBorder3d** ()

  Constructor.

- virtual ~**HxImgFtorSetBorder3d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**ImgDataPtrType** imgPtr, **HxSizes** imgSize, **HxTagList** &tags, **HxImgFtor-Description** *=0)

  Calls one of the functions below to do the actual work based on the "borderType" tag: **HxFuncBorder-Constant3d** (p. 73), **HxFuncBorderMirror3d** (p. 71) *(default)*, **HxFuncBorderPropagate3d** (p. 75).

6.120.1 Detailed Description

template<class ImgSigT> class HxImgFtorSetBorder3d< ImgSigT >

Instantiation of generic algorithm for border operations on 3D images.

6.120.2 Member Typedef Documentation

6.120.2.1 template<class ImgSigT> typedef HxImgFtorSetBorderKey HxImgFtorSetBorder3d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 395).

6.120.3 Constructor & Destructor Documentation

6.120.3.1 template<class ImgSigT> HxImgFtorSetBorder3d< ImgSigT >::HxImgFtorSetBorder3d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(
21         HxImgFtorSetBorderKey(HxClassName<ImgSigT>()))
22 {
23 }
```

6.120.3.2 template<class ImgSigT> HxImgFtorSetBorder3d< ImgSigT >::~HxImgFtorSetBorder3d () [virtual]

Destructor.

```
27 {
28 }
```

6.120.4 Member Function Documentation

6.120.4.1 template<class ImgSigT> void HxImgFtorSetBorder3d< ImgSigT >::doIt (ImgDataPtrType *imgPtr*, HxSizes *imgSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls one of the functions below to do the actual work based on the ”borderType” tag: **HxFuncBorderConstant3d** (p. 73), **HxFuncBorderMirror3d** (p. 71) (default), **HxFuncBorderPropagate3d** (p. 75).

Reimplemented from **HxImgFtorI1Cast** (p. 397).

```
36 {
37     HxBorderType borderType
38         = HxGetTag(tags, "borderType", HxBorderType(HXBORDERMIRROR));
39     HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0, 0, 0));
40
41     switch (borderType)
42     {
43     case HXBORDERCONSTANT   :
44         {
45             HxValue value = HxGetTag(tags, "borderValue", HxValue(0));
46             HxFuncBorderConstant3d(imgPtr, imgSize, borderSize, value);
47         }
48         break;
49     case HXBORDERMIRROR     :
```

```
50         HxFuncBorderMirror3d(imgPtr, imgSize, borderSize);
51         break;
52     case HXBORDERPROPAGATE  :
53         HxFuncBorderPropagate3d(imgPtr, imgSize, borderSize);
54         break;
55     }
56 }
```

The documentation for this class was generated from the following files:

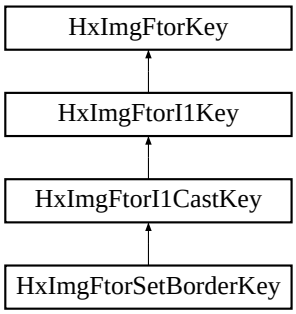
- **HxImgFtorSetBorder3d.h**
- HxImgFtorSetBorder3d.c

6.121 HxImgFtorSetBorderKey Class Reference

Key for HxImgFtorSetBorder.

`#include <HxImgFtorSetBorderKey.h>`

Inheritance diagram for HxImgFtorSetBorderKey::



Public Methods

- **HxImgFtorSetBorderKey** (**HxString** imgSig)
  Constructor.

6.121.1 Detailed Description

Key for HxImgFtorSetBorder.

6.121.2 Constructor & Destructor Documentation

6.121.2.1 HxImgFtorSetBorderKey::HxImgFtorSetBorderKey (HxString *imgSig*) [inline]

Constructor.

```
28     : HxImgFtorI1CastKey("HxImgFtorSetBorder", imgSig)
29 {
30 }
```

The documentation for this class was generated from the following file:

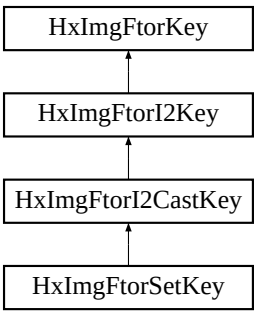
- **HxImgFtorSetBorderKey.h**

6.122 HxImgFtorSetKey Class Reference

Key for **HxImgFtorSet** (p. 472).

`#include <HxImgFtorSetKey.h>`

Inheritance diagram for HxImgFtorSetKey::



Public Methods

- **HxImgFtorSetKey** (**HxString** dstImgSig, **HxString** srcImgSig)

  Constructor.

6.122.1 Detailed Description

Key for **HxImgFtorSet** (p. 472).

6.122.2 Constructor & Destructor Documentation

6.122.2.1 HxImgFtorSetKey::HxImgFtorSetKey (HxString *dstImgSig*, HxString *srcImgSig*) `[inline]`

Constructor.

```
30     : HxImgFtorI2CastKey("HxImgFtorSet", dstImgSig, srcImgSig)
31 {
32 }
```

The documentation for this class was generated from the following file:

- **HxImgFtorSetKey.h**

6.123 HxImgFtorTable Class Reference

Table with (**HxImgFtorKey** (p. 447), **HxImgFunctor** (p. 490)∗) pairs.

`#include <HxImgFtorTable.h>`

Public Methods

- void **insert** (const **HxImgFtorKey** &key, **HxImgFunctor** ∗f)

  Insert a (key,functor) combination.

- **HxImgFunctor** ∗ **find** (const **HxImgFtorKey** &key)

  Find functor based on given key.

- void **addImgFtorObserver** (**HxImgFtorObserver** ∗)
- STD_OSTREAM & **put** (STD_OSTREAM &) const
- ∼**HxImgFtorTable** ()

Static Public Methods

- HxImgFtorTable & **instance** ()

  Access to this singleton class.

Protected Methods

- **HxImgFtorTable** ()
- **HxImgFtorTable** (const HxImgFtorTable &)

6.123.1 Detailed Description

Table with (**HxImgFtorKey** (p. 447), **HxImgFunctor** (p. 490)∗) pairs.

6.123.2 Member Function Documentation

6.123.2.1 HxImgFtorTable & HxImgFtorTable::instance () [static]

Access to this singleton class.

```
46 {
47     static HxImgFtorTable _instance;
48     return _instance;
49 }
```

6.123.2.2 void HxImgFtorTable::insert (const HxImgFtorKey & *key*, HxImgFunctor * *f*)

Insert a (key,functor) combination.

```
53 {
54     _map[key] = f;
55 
56     static int entryNum = 0;
57     static HxRegKey* entryKey
58         = HxRegistry::instance().insertKey("/imagefunctortable/entries");
59     HxRegKey* classKey
60         = entryKey->insertKey(key.getClassName());
61     HxString entryName("entry");
62     entryName += makeString(entryNum++);
63     classKey->insertValue(entryName, HxRegData(key.toString()));
64 
65     ObserverList::iterator p;
66     for (p = _observerList.begin(); p != _observerList.end(); p++)
67         (*p)->inserted(key, *f);
68 }
```

6.123.2.3 HxImgFunctor * HxImgFtorTable::find (const HxImgFtorKey & *key*)

Find functor based on given key.

```
72 {
73     Map::iterator i;
74 #ifdef IF_DEBUG
75     HxEnvironment::instance()->outputStream()
76         << "Was requested to find " << key << STD_ENDL;
77     HxEnvironment::instance()->flush();
78 #endif IF_DEBUG
79     i = _map.find(key);
80     if (i == _map.end())
81         return 0;
82 #ifdef IF_DEBUG
83     HxEnvironment::instance()->outputStream()
84         << "And indeed found" << STD_ENDL << " ==> ";
85     HxEnvironment::instance()->flush();
86     ((*i).second)->put(HxEnvironment::instance()->outputStream());
87     HxEnvironment::instance()->outputStream() << STD_ENDL;
88     HxEnvironment::instance()->flush();
89 #endif IF_DEBUG
90     return (*i).second;
91 }
```

The documentation for this class was generated from the following files:

- **HxImgFtorTable.h**
- HxImgFtorTable.c

6.124 HxImgFtorTableTem Class Template Reference

Template for typed access to the **HxImgFtorTable** (p. 481).

`#include <HxImgFtorTableTem.h>`

Public Methods

- FunctorType * **find** (const typename FunctorType::KeyType &key) const
  Find functor in the **HxImgFtorTable** (p. 481) *based on given key and cast the result to FunctorType.*

6.124.1 Detailed Description

template<class FunctorType> class HxImgFtorTableTem< FunctorType >

Template for typed access to the **HxImgFtorTable** (p. 481).

6.124.2 Member Function Documentation

6.124.2.1 template<class FunctorType> FunctorType* HxImgFtorTableTem< FunctorType >::find (const typename FunctorType::KeyType & *key*) const [inline]

Find functor in the **HxImgFtorTable** (p. 481) based on given key and cast the result to FunctorType.

```
28     {
29         return (FunctorType*)(HxImgFtorTable::instance().find(key));
30     }
```

The documentation for this class was generated from the following file:

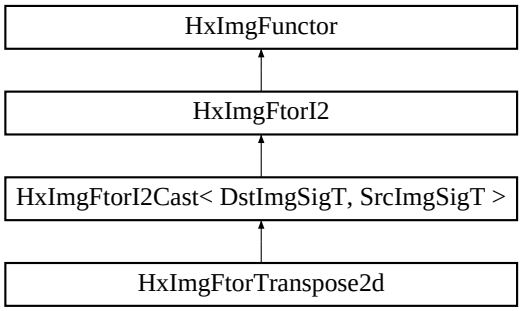
- **HxImgFtorTableTem.h**

6.125 HxImgFtorTranspose2d Class Template Reference

Instantiation of generic algorithm for transpose operations on 2D images.

`#include <HxImgFtorTranspose2d.h>`

Inheritance diagram for HxImgFtorTranspose2d::



Public Types

- typedef **HxImgFtorTranspose2dKey KeyType**
  The key type of this class.

Public Methods

- **HxImgFtorTranspose2d** ()

  Constructor.

- virtual ~**HxImgFtorTranspose2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  The actual operation.

6.125.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT> class HxImgFtorTranspose2d< DstImgSigT, SrcImgSigT >

Instantiation of generic algorithm for transpose operations on 2D images.

6.125.2 Member Typedef Documentation

6.125.2.1 template<class DstImgSigT, class SrcImgSigT> typedef HxImgFtorTranspose2dKey HxImgFtorTranspose2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 407).

6.125.3 Constructor & Destructor Documentation

6.125.3.1 template<class DstImgSigT, class SrcImgSigT> HxImgFtorTranspose2d< DstImgSigT, SrcImgSigT >::HxImgFtorTranspose2d () [inline]

Constructor.

```
18     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
19         HxImgFtorTranspose2dKey(
20             HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>()))
21 {
22 #ifdef CD_TRACE
23     HxEnvironment::instance()->outputStream()
24         << "HxImgFtorTranspose2d::HxImgFtorTranspose2d()" << STD_ENDL;
25 #endif
26 }
```

6.125.3.2 template<class DstImgSigT, class SrcImgSigT> HxImgFtorTranspose2d< DstImgSigT, SrcImgSigT >::~HxImgFtorTranspose2d () [virtual]

Destructor.

```
30 {
31 #ifdef CD_TRACE
32     HxEnvironment::instance()->outputStream()
33         << "HxImgFtorTranspose2d::~HxImgFtorTranspose2d()" << STD_ENDL;
34 #endif
35 }
```

6.125.4 Member Function Documentation

6.125.4.1 template<class DstImgSigT, class SrcImgSigT> void HxImgFtorTranspose2d< DstImgSigT, SrcImgSigT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

The actual operation.

Reimplemented from **HxImgFtorI2Cast** (p. 411).

```
53 {
54     // write size check!
55     int y = dstSize.x();
56     while (--y >= 0)
57     {
58         HxTransposeXLine(dstPtr, srcPtr, dstSize.y());
59         srcPtr.incY();
60         dstPtr.incX();
61     }
62 }
```

The documentation for this class was generated from the following files:

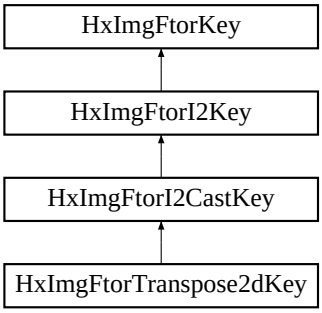
- **HxImgFtorTranspose2d.h**
- HxImgFtorTranspose2d.c

6.126 HxImgFtorTranspose2dKey Class Reference

Key for **HxImgFtorTranspose2d** (p. 483).

`#include <HxImgFtorTranspose2dKey.h>`

Inheritance diagram for HxImgFtorTranspose2dKey::



Public Methods

- **HxImgFtorTranspose2dKey** (**HxString** dstImgSig, **HxString** srcImgSig)

  Constructor.

6.126.1 Detailed Description

Key for **HxImgFtorTranspose2d** (p. 483).

6.126.2 Constructor & Destructor Documentation

6.126.2.1 HxImgFtorTranspose2dKey::HxImgFtorTranspose2dKey (HxString *dstImgSig*, HxString *srcImgSig*) [inline]

Constructor.

```
30         : HxImgFtorI2CastKey("HxImgFtorTranspose2d", dstImgSig, srcImgSig)
31 {
32 }
```

The documentation for this class was generated from the following file:

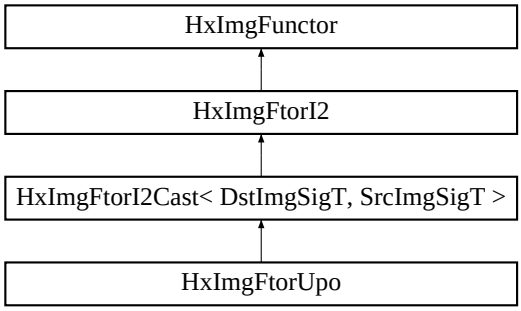
- **HxImgFtorTranspose2dKey.h**

6.127 HxImgFtorUpo Class Template Reference

Instantiation of generic algorithm for unary pixel operations on images.

`#include <HxImgFtorUpo.h>`

Inheritance diagram for HxImgFtorUpo::



Public Types

- typedef **HxImgFtorUpoKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorUpo** ()

  Constructor.

- virtual ~**HxImgFtorUpo** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**DstDataPtrType** dstPtr, **SrcDataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncUnaryPixOp** (p. 82) *to do the actual work.*

6.127.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class UpoT> class HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >

Instantiation of generic algorithm for unary pixel operations on images.

6.127.2 Member Typedef Documentation

6.127.2.1 template<class DstImgSigT, class SrcImgSigT, class UpoT> typedef HxImgFtorUpoKey HxImgFtorUpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 407).

6.127.3 Constructor & Destructor Documentation

6.127.3.1 template<class DstImgSigT, class SrcImgSigT, class UpoT> HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >::HxImgFtorUpo () [inline]

Constructor.

```
23     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
24         HxImgFtorUpoKey(HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
25             HxClassName<UpoT>()))
26 {
27 #ifdef CD_TRACE
28     HxEnvironment::instance()->outputStream()
29         << "HxImgFtorUpo::HxImgFtorUpo()" << STD_ENDL;
30 #endif
31     static HxRegKey* upoKey
32         = HxRegistry::instance().insertKey("/imagefunctortable/upo");
33
34     HxRegKey* k = upoKey->insertKey(HxClassName<UpoT>());
35     k = k->insertKey("resulttype");
36     k->insertValue(
37         HxClassName<SrcImgSigT>(), HxRegData(HxClassName<DstImgSigT>()));
38 }
```

6.127.3.2 template<class DstImgSigT, class SrcImgSigT, class UpoT> HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >::~HxImgFtorUpo () [virtual]

Destructor.

```
42 {
43 #ifdef CD_TRACE
44     HxEnvironment::instance()->outputStream()
45         << "HxImgFtorUpo::~HxImgFtorUpo()" << STD_ENDL;
46 #endif
47 }
```

6.127.4 Member Function Documentation

6.127.4.1 template<class DstImgSigT, class SrcImgSigT, class UpoT> void HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncUnaryPixOp** (p. 82) to do the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 411).

```
54 {
55     UpoT upo(tags);
56
57     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
58     HxFuncUnaryPixOp(
59         dstPtr, srcPtr, nPix, upo);
60 }
```

The documentation for this class was generated from the following files:

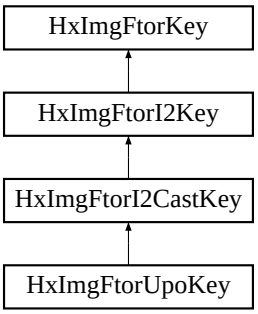
- **HxImgFtorUpo.h**
- HxImgFtorUpo.c

6.128 HxImgFtorUpoKey Class Reference

Key for **HxImgFtorUpo** (p. 486).

`#include <HxImgFtorUpoKey.h>`

Inheritance diagram for HxImgFtorUpoKey::



Public Methods

- **HxImgFtorUpoKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** upoName)

  Constructor.

6.128.1 Detailed Description

Key for **HxImgFtorUpo** (p. 486).

6.128.2 Constructor & Destructor Documentation

6.128.2.1 HxImgFtorUpoKey::HxImgFtorUpoKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *upoName*) [inline]

Constructor.

```
30     : HxImgFtorI2CastKey("HxImgFtorUpo", dstImgSig, srcImgSig)
31 {
32     addArgument(upoName);
33 }
```

The documentation for this class was generated from the following file:

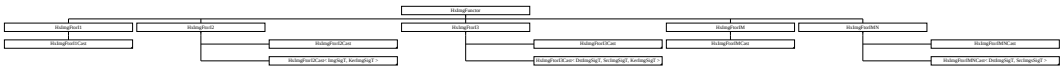
- **HxImgFtorUpoKey.h**

6.129 HxImgFunctor Class Reference

The base class of all image functors.

`#include <HxImgFunctor.h>`

Inheritance diagram for HxImgFunctor::



Public Methods

- **HxImgFunctor** (const **HxImgFtorKey** &)
- virtual ~**HxImgFunctor** ()
- virtual STD_OSTREAM & **put** (STD_OSTREAM &) const
- virtual **HxSizes minimumBorderSize** (**HxTagList** &tags) const

  some functor instantiations can determine the minimum border size needed to perform the operation correctly.

- virtual bool **probeOp** (**HxTagList** &) const

  some operator instantiations can be queried for pre-condition information.

Protected Methods

- **HxImgFtorDescription** * **getDescription** () const
- **HxSizes getBorderSize** (**HxTagList** &tags, **HxSizes** defSize=**HxSizes**(0, 0, 0)) const

  Support method that reads a border size from the tag list if set.

6.129.1 Detailed Description

The base class of all image functors.

6.129.2 Member Function Documentation

6.129.2.1 HxSizes HxImgFunctor::minimumBorderSize (HxTagList & *tags*) const [virtual]

some functor instantiations can determine the minimum border size needed to perform the operation correctly.

MNPO IMPORTANT DESIGN NOTES: this (obsolete) function should be implemented by means of the probeOp MNPO

Reimplemented in **HxImgFtorKernelNgb2d** (p. 445), and **HxImgFtorNgb2d** (p. 457).

```
30 {
31     return HxSizes(0,0,0);
32 }
```

6.129.2.2 bool HxImgFunctor::probeOp (HxTagList & *tags*) const [virtual]

some operator instantiations can be queried for pre-condition information.

Reimplemented in **HxImgFtorMNpo** (p. 451).

```
44 {
45     return true;
46 }
```

6.129.2.3 HxSizes HxImgFunctor::getBorderSize (HxTagList & *tags*, HxSizes *defSize* = HxSizes(0, 0, 0)) const [protected]

Support method that reads a border size from the tag list if set.

Returns a border size of (0, 0, 0) otherwise.

```
36 {
37     return HxGetTag(tags, "borderSize", defSize);
38 }
```

The documentation for this class was generated from the following files:

- **HxImgFunctor.h**
- HxImgFunctor.c

6.130 HxInstAddInfAss2d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 2d kernel.

Public Attributes

- **HxImgFtorGenConv2d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoInfAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.130.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddInfAss2d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 2d kernel.

6.130.2 Member Data Documentation

6.130.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoInfAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel2d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddInfAss2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddInfInst.c

6.131 HxInstAddInfAss2dK1d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv2dK1d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoInfAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.131.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddInfAss2dK1d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 1d kernel.

6.131.2 Member Data Documentation

6.131.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dK1d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoInfAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddInfAss2dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddInfInst.c

6.132 HxInstAddMaxAss2d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 2d kernel.

Public Attributes

- **HxImgFtorGenConv2d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMaxAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.132.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddMaxAss2d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 2d kernel.

6.132.2 Member Data Documentation

6.132.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMaxAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel2d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddMaxAss2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMaxInst.c

6.133 HxInstAddMaxAss2dK1d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv2dK1d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMaxAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.133.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddMaxAss2dK1d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 1d kernel.

6.133.2 Member Data Documentation

6.133.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dK1d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMaxAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddMaxAss2dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMaxInst.c

6.134 HxInstAddMinAss2d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 2d kernel.

Public Attributes

- **HxImgFtorGenConv2d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMinAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.134.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddMinAss2d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 2d kernel.

6.134.2 Member Data Documentation

6.134.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMinAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel2d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddMinAss2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMinInst.c

6.135 HxInstAddMinAss2dK1d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv2dK1d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMinAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.135.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddMinAss2dK1d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 1d kernel.

6.135.2 Member Data Documentation

6.135.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dK1d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMinAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddMinAss2dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMinInst.c

6.136 HxInstAddSupAss2d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 2d kernel.

Public Attributes

- **HxImgFtorGenConv2d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoSupAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.136.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddSupAss2d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 2d kernel.

6.136.2 Member Data Documentation

6.136.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoSupAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel2d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddSupAss2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddSupInst.c

6.137 HxInstAddSupAss2dK1d Class Template Reference

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv2dK1d**< ImgSigT, KerSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoSupAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.137.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstAddSupAss2dK1d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 1d kernel.

6.137.2 Member Data Documentation

6.137.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dK1d<ImgSigT, KerSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoSupAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstAddSupAss2dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddSupInst.c

6.138 HxInstantiatorAbs Class Template Reference

Instantiator for unary pixel operation with absolute value.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoAbs**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.138.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAbs< ImgSigT >

Instantiator for unary pixel operation with absolute value.

6.138.2 Member Data Documentation

6.138.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoAbs<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorAbs::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoAbsInst.c

6.139 HxInstantiatorAcos Class Template Reference

Instantiator for unary pixel operation with arc cosine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoAcos**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.139.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAcos< ImgSigT >

Instantiator for unary pixel operation with arc cosine.

6.139.2 Member Data Documentation

6.139.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoAcos<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorAcos::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoAcosInst.c

6.140 HxInstantiatorAdd Class Template Reference

Instantiator for binary pixel operation with addition.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoAdd**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.140.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAdd< ImgSigT >

Instantiator for binary pixel operation with addition.

6.140.2 Member Data Documentation

6.140.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoAdd<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoAddInst.c

6.141 HxInstantiatorAddReduce Class Template Reference

Instantiator for reduce operation with addition.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoAddAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.141.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAddReduce< ImgSigT >

Instantiator for reduce operation with addition.

6.141.2 Member Data Documentation

6.141.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorAddReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceAddAssignInst.c

6.142 HxInstantiatorAddV Class Template Reference

Instantiator for binary pixel operation with addition.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoAdd**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.142.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAddV< ImgSigT >

Instantiator for binary pixel operation with addition.

6.142.2 Member Data Documentation

6.142.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorAddV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoAddInst.c

6.143 HxInstantiatorAnd Class Template Reference

Instantiator for binary pixel operation with and.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoAnd**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.143.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAnd< ImgSigT >

Instantiator for binary pixel operation with and.

6.143.2 Member Data Documentation

6.143.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoAnd<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorAnd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoAndInst.c

6.144 HxInstantiatorAndV Class Template Reference

Instantiator for binary pixel operation with and.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoAnd**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.144.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAndV< ImgSigT >

Instantiator for binary pixel operation with and.

6.144.2 Member Data Documentation

6.144.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorAndV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoAndInst.c

6.145 HxInstantiatorArg Class Template Reference

Instantiator for unary pixel operation with argument.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoArg**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.145.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorArg< DstSigT, ImgSigT >

Instantiator for unary pixel operation with argument.

6.145.2 Member Data Documentation

6.145.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoArg<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorArg::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoArgInst.c

6.146 HxInstantiatorAsin Class Template Reference

Instantiator for unary pixel operation with arc sine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoAsin**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.146.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAsin< ImgSigT >

Instantiator for unary pixel operation with arc sine.

6.146.2 Member Data Documentation

6.146.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoAsin<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorAsin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoAsinInst.c

6.147 HxInstantiatorAtan Class Template Reference

Instantiator for unary pixel operation with arc tangent.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoAtan**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.147.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAtan< ImgSigT >

Instantiator for unary pixel operation with arc tangent.

6.147.2 Member Data Documentation

6.147.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoAtan<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorAtan::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoAtanInst.c

6.148 HxInstantiatorAtan2 Class Template Reference

Instantiator for unary pixel operation with arc tangent.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoAtan2**< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.148.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorAtan2< DstSigT, ImgSigT >

Instantiator for unary pixel operation with arc tangent.

6.148.2 Member Data Documentation

6.148.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoAtan2<typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorAtan2::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoAtan2Inst.c

6.149 HxInstantiatorCeil Class Template Reference

Instantiator for unary pixel operation with ceiling.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoCeil**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.149.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCeil< ImgSigT >

Instantiator for unary pixel operation with ceiling.

6.149.2 Member Data Documentation

6.149.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoCeil<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorCeil::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoCeilInst.c

6.150 HxInstantiatorColSpace Class Template Reference

Instantiator for unary pixel operation with color space conversion.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoColSpace**< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.150.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorColSpace< DstSigT, ImgSigT >

Instantiator for unary pixel operation with color space conversion.

6.150.2 Member Data Documentation

6.150.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoColSpace<typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorColSpace::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoColSpaceInst.c

6.151 HxInstantiatorComplement Class Template Reference

Instantiator for unary pixel operation with complement.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoComplement**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.151.1 Detailed Description

template<class ImgSigT> class HxInstantiatorComplement< ImgSigT >

Instantiator for unary pixel operation with complement.

6.151.2 Member Data Documentation

6.151.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoComplement<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorComplement::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoComplementInst.c

6.152 HxInstantiatorConjugate Class Template Reference

Instantiator for unary pixel operation with conjugate.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoConjugate**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.152.1 Detailed Description

template<class ImgSigT> class HxInstantiatorConjugate< ImgSigT >

Instantiator for unary pixel operation with conjugate.

6.152.2 Member Data Documentation

6.152.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoConjugate<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorConjugate::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoConjugateInst.c

6.153 HxInstantiatorCos Class Template Reference

Instantiator for unary pixel operation with cosine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoCos**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.153.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCos< ImgSigT >

Instantiator for unary pixel operation with cosine.

6.153.2 Member Data Documentation

6.153.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoCos<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorCos::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoCosInst.c

6.154 HxInstantiatorCosh Class Template Reference

Instantiator for unary pixel operation with hyperbolic cosine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoCosh**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.154.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCosh< ImgSigT >

Instantiator for unary pixel operation with hyperbolic cosine.

6.154.2 Member Data Documentation

6.154.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoCosh<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorCosh::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoCoshInst.c

6.155 HxInstantiatorCross Class Template Reference

Instantiator for binary pixel operation with cross product.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoCross**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.155.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCross< ImgSigT >

Instantiator for binary pixel operation with cross product.

6.155.2 Member Data Documentation

6.155.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoCross<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorCross::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoCrossInst.c

6.156 HxInstantiatorCrossV Class Template Reference

Instantiator for binary pixel operation with cross product.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoCross**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.156.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCrossV< ImgSigT >

Instantiator for binary pixel operation with cross product.

6.156.2 Member Data Documentation

6.156.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorCrossV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoCrossInst.c

6.157 HxInstantiatorDiv Class Template Reference

Instantiator for binary pixel operation with division.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoDiv**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.157.1 Detailed Description

template<class ImgSigT> class HxInstantiatorDiv< ImgSigT >

Instantiator for binary pixel operation with division.

6.157.2 Member Data Documentation

6.157.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoDiv<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorDiv::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoDivInst.c

6.158 HxInstantiatorDivV Class Template Reference

Instantiator for binary pixel operation with division.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoDiv**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.158.1 Detailed Description

template<class ImgSigT> class HxInstantiatorDivV< ImgSigT >

Instantiator for binary pixel operation with division.

6.158.2 Member Data Documentation

6.158.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorDivV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoDivInst.c

6.159 HxInstantiatorDot Class Template Reference

Instantiator for binary pixel operation with dot product.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoDot**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.159.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorDot< DstSigT, ImgSigT >

Instantiator for binary pixel operation with dot product.

6.159.2 Member Data Documentation

6.159.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoDot<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorDot::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoDotInst.c

6.160 HxInstantiatorDotV Class Template Reference

Instantiator for binary pixel operation with dot product.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoDot**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.160.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorDotV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with dot product.

6.160.2 Member Data Documentation

6.160.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorDotV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoDotInst.c

6.161 HxInstantiatorEqual Class Template Reference

Instantiator for binary pixel operation with equal.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.161.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with equal.

6.161.2 Member Data Documentation

6.161.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorEqual::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoEqualInst.c

6.162 HxInstantiatorEqualV Class Template Reference

Instantiator for binary pixel operation with equal.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.162.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with equal.

6.162.2 Member Data Documentation

6.162.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorEqualV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoEqualInst.c

6.163 HxInstantiatorExp Class Template Reference

Instantiator for unary pixel operation with exponent.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoExp**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.163.1 Detailed Description

template<class ImgSigT> class HxInstantiatorExp< ImgSigT >

Instantiator for unary pixel operation with exponent.

6.163.2 Member Data Documentation

6.163.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoExp<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorExp::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoExpInst.c

6.164 HxInstantiatorFloor Class Template Reference

Instantiator for unary pixel operation with floor.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoFloor**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

Instantiate image functor.

6.164.1 Detailed Description

template<class ImgSigT> class HxInstantiatorFloor< ImgSigT >

Instantiator for unary pixel operation with floor.

6.164.2 Member Data Documentation

6.164.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoFloor<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorFloor::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoFloorInst.c

6.165 HxInstantiatorGreaterEqual Class Template Reference

Instantiator for binary pixel operation with greater equal.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoGreaterEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.165.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater equal.

6.165.2 Member Data Documentation

6.165.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorGreaterEqual::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoGreaterEqualInst.c

6.166 HxInstantiatorGreaterEqualV Class Template Reference

Instantiator for binary pixel operation with greater equal.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoGreaterEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.166.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater equal.

6.166.2 Member Data Documentation

6.166.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorGreaterEqualV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoGreaterEqualInst.c

6.167 HxInstantiatorGreaterThan Class Template Reference

Instantiator for binary pixel operation with greater than.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoGreaterThan**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.167.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterThan< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater than.

6.167.2 Member Data Documentation

6.167.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorGreaterThan::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoGreaterThanInst.c

6.168 HxInstantiatorGreaterThanV Class Template Reference

Instantiator for binary pixel operation with greater than.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoGreaterThan**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.168.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterThanV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater than.

6.168.2 Member Data Documentation

6.168.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorGreaterThanV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoGreaterThanInst.c

6.169 HxInstantiatorInf Class Template Reference

Instantiator for binary pixel operation with infimum.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoInf**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.169.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorInf**< **ImgSigT** >

Instantiator for binary pixel operation with infimum.

6.169.2 Member Data Documentation

6.169.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoInf<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorInf::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoInfInst.c

6.170 HxInstantiatorInfReduce Class Template Reference

Instantiator for reduce operation with infimum.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoInfAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.170.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorInfReduce**< **ImgSigT** >

Instantiator for reduce operation with infimum.

6.170.2 Member Data Documentation

6.170.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorInfReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceInfAssignInst.c

6.171 HxInstantiatorInfV Class Template Reference

Instantiator for binary pixel operation with infimum.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoInf**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.171.1 Detailed Description

template<class ImgSigT> class HxInstantiatorInfV< ImgSigT >

Instantiator for binary pixel operation with infimum.

6.171.2 Member Data Documentation

6.171.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorInfV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoInfInst.c

6.172 HxInstantiatorLeftShift Class Template Reference

Instantiator for binary pixel operation with left shift.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoLeftShift**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.172.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLeftShift< ImgSigT >

Instantiator for binary pixel operation with left shift.

6.172.2 Member Data Documentation

6.172.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorLeftShift::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLeftShiftInst.c

6.173 HxInstantiatorLeftShiftV Class Template Reference

Instantiator for binary pixel operation with left shift.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoLeftShift**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.173.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLeftShiftV< ImgSigT >

Instantiator for binary pixel operation with left shift.

6.173.2 Member Data Documentation

6.173.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorLeftShiftV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLeftShiftInst.c

6.174 HxInstantiatorLessEqual Class Template Reference

Instantiator for binary pixel operation with less equal.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoLessEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.174.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less equal.

6.174.2 Member Data Documentation

6.174.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorLessEqual::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLessEqualInst.c

6.175 HxInstantiatorLessEqualV Class Template Reference

Instantiator for binary pixel operation with less equal.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoLessEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.175.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less equal.

6.175.2 Member Data Documentation

6.175.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorLessEqualV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLessEqualInst.c

6.176 HxInstantiatorLessThan Class Template Reference

Instantiator for binary pixel operation with less than.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoLessThan**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.176.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessThan< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less than.

6.176.2 Member Data Documentation

6.176.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoLessThan<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorLessThan::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLessThanInst.c

6.177 HxInstantiatorLessThanV Class Template Reference

Instantiator for binary pixel operation with less than.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoLessThan**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.177.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessThanV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less than.

6.177.2 Member Data Documentation

6.177.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorLessThanV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoLessThanInst.c

6.178 HxInstantiatorLog Class Template Reference

Instantiator for unary pixel operation with natural logarithm.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoLog**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.178.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLog< ImgSigT >

Instantiator for unary pixel operation with natural logarithm.

6.178.2 Member Data Documentation

6.178.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoLog<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorLog::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoLogInst.c

6.179 HxInstantiatorLog10 Class Template Reference

Instantiator for unary pixel operation with base 10 logarithm.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoLog10**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.179.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLog10< ImgSigT >

Instantiator for unary pixel operation with base 10 logarithm.

6.179.2 Member Data Documentation

6.179.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoLog10<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorLog10::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoLog10Inst.c

6.180 HxInstantiatorMax Class Template Reference

Instantiator for binary pixel operation with maximum.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoMax**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.180.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMax< ImgSigT >

Instantiator for binary pixel operation with maximum.

6.180.2 Member Data Documentation

6.180.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoMax<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorMax::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMaxInst.c

6.181 HxInstantiatorMaxReduce Class Template Reference

Instantiator for reduce operation with maximum.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoMaxAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.181.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMaxReduce< ImgSigT >

Instantiator for reduce operation with maximum.

6.181.2 Member Data Documentation

6.181.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMaxReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceMaxAssignInst.c

6.182 HxInstantiatorMaxV Class Template Reference

Instantiator for binary pixel operation with maximum.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoMax**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.182.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMaxV< ImgSigT >

Instantiator for binary pixel operation with maximum.

6.182.2 Member Data Documentation

6.182.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMaxV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMaxInst.c

6.183 HxInstantiatorMin Class Template Reference

Instantiator for binary pixel operation with minimum.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoMin**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.183.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMin< ImgSigT >

Instantiator for binary pixel operation with minimum.

6.183.2 Member Data Documentation

6.183.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoMin<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMinInst.c

6.184 HxInstantiatorMinReduce Class Template Reference

Instantiator for reduce operation with minimum.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoMinAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.184.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMinReduce< ImgSigT >

Instantiator for reduce operation with minimum.

6.184.2 Member Data Documentation

6.184.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMinReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceMinAssignInst.c

6.185 HxInstantiatorMinV Class Template Reference

Instantiator for binary pixel operation with minimum.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoMin**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.185.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMinV< ImgSigT >

Instantiator for binary pixel operation with minimum.

6.185.2 Member Data Documentation

6.185.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMinV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMinInst.c

6.186 HxInstantiatorMod Class Template Reference

Instantiator for binary pixel operation with modulo.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoMod**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.186.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMod< ImgSigT >

Instantiator for binary pixel operation with modulo.

6.186.2 Member Data Documentation

6.186.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoMod<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorMod::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoModInst.c

6.187 HxInstantiatorModV Class Template Reference

Instantiator for binary pixel operation with modulo.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoMod**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.187.1 Detailed Description

template<class ImgSigT> class HxInstantiatorModV< ImgSigT >

Instantiator for binary pixel operation with modulo.

6.187.2 Member Data Documentation

6.187.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorModV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoModInst.c

6.188 HxInstantiatorMul Class Template Reference

Instantiator for binary pixel operation with multiplication.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoMul**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.188.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMul**< **ImgSigT** >

Instantiator for binary pixel operation with multiplication.

6.188.2 Member Data Documentation

6.188.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoMul<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorMul::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMulInst.c

6.189 HxInstantiatorMulReduce Class Template Reference

Instantiator for reduce operation with multiplication.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoMulAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.189.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMulReduce**< **ImgSigT** >

Instantiator for reduce operation with multiplication.

6.189.2 Member Data Documentation

6.189.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMulReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceMulAssignInst.c

6.190 HxInstantiatorMulV Class Template Reference

Instantiator for binary pixel operation with multiplication.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoMul**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.190.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMulV< ImgSigT >

Instantiator for binary pixel operation with multiplication.

6.190.2 Member Data Documentation

6.190.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorMulV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoMulInst.c

6.191 HxInstantiatorNegate Class Template Reference

Instantiator for unary pixel operation with negation.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoNegate**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.191.1 Detailed Description

template<class ImgSigT> class HxInstantiatorNegate< ImgSigT >

Instantiator for unary pixel operation with negation.

6.191.2 Member Data Documentation

6.191.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoNegate<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorNegate::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoNegateInst.c

6.192 HxInstantiatorNorm1 Class Template Reference

Instantiator for unary pixel operation with L1 norm.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoNorm1**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

6.192.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm1< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L1 norm.

6.192.2 Member Data Documentation

6.192.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoNorm1<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorNorm1::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoNorm1Inst.c

6.193 HxInstantiatorNorm2 Class Template Reference

Instantiator for unary pixel operation with L2 norm.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoNorm2**< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.193.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm2< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L2 norm.

6.193.2 Member Data Documentation

6.193.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoNorm2<typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorNorm2::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoNorm2Inst.c

6.194 HxInstantiatorNorm2Sqr Class Template Reference

Instantiator for unary pixel operation with squared L2 norm.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoNorm2Sqr**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.194.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm2Sqr< DstSigT, ImgSigT >

Instantiator for unary pixel operation with squared L2 norm.

6.194.2 Member Data Documentation

6.194.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoNorm2Sqr<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorNorm2Sqr::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoNorm2SqrInst.c

6.195 HxInstantiatorNormInf Class Template Reference

Instantiator for unary pixel operation with L inf norm.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoNormInf**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.195.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNormInf< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L inf norm.

6.195.2 Member Data Documentation

6.195.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoNormInf<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorNormInf::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoNormInfInst.c

6.196 HxInstantiatorNotEqual Class Template Reference

Instantiator for binary pixel operation with not equal.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, **HxBpoNotEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.196.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNotEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with not equal.

6.196.2 Member Data Documentation

6.196.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorNotEqual::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoNotEqualInst.c

6.197 HxInstantiatorNotEqualV Class Template Reference

Instantiator for binary pixel operation with not equal.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, **HxBpoNotEqual**< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.197.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNotEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with not equal.

6.197.2 Member Data Documentation

6.197.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxBpoBind2Val<typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorNotEqualV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoNotEqualInst.c

6.198 HxInstantiatorOr Class Template Reference

Instantiator for binary pixel operation with or.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoOr**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.198.1 Detailed Description

template<class ImgSigT> class HxInstantiatorOr< ImgSigT >

Instantiator for binary pixel operation with or.

6.198.2 Member Data Documentation

6.198.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoOr<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorOr::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoOrInst.c

6.199 HxInstantiatorOrV Class Template Reference

Instantiator for binary pixel operation with or.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoOr**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.199.1 Detailed Description

template<class ImgSigT> class HxInstantiatorOrV< ImgSigT >

Instantiator for binary pixel operation with or.

6.199.2 Member Data Documentation

6.199.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorOrV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoOrInst.c

6.200 HxInstantiatorPow Class Template Reference

Instantiator for binary pixel operation with power.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoPow**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.200.1 Detailed Description

template<class ImgSigT> class HxInstantiatorPow< ImgSigT >

Instantiator for binary pixel operation with power.

6.200.2 Member Data Documentation

6.200.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoPow<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorPow::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoPowInst.c

6.201 HxInstantiatorPowV Class Template Reference

Instantiator for binary pixel operation with power.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoPow**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.201.1 Detailed Description

template<class ImgSigT> class HxInstantiatorPowV< ImgSigT >

Instantiator for binary pixel operation with power.

6.201.2 Member Data Documentation

6.201.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorPowV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoPowInst.c

6.202 HxInstantiatorProduct Class Template Reference

Instantiator for unary pixel operation with product.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoProduct**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.202.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorProduct< DstSigT, ImgSigT >

Instantiator for unary pixel operation with product.

6.202.2 Member Data Documentation

6.202.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoProduct<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorProduct::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoProductInst.c

6.203 HxInstantiatorRGB2Intensity Class Template Reference

Instantiator for unary pixel operation with **RGB2Intensity** (p. 862).

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **RGB2Intensity**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

6.203.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorRGB2Intensity< DstSigT, ImgSigT >

Instantiator for unary pixel operation with **RGB2Intensity** (p. 862).

6.203.2 Member Data Documentation

6.203.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, RGB2Intensity<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorRGB2Intensity::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRGB2Intensity.c

6.204 HxInstantiatorRightShift Class Template Reference

Instantiator for binary pixel operation with right shift.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoRightShift**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

Instantiate image functor.

6.204.1 Detailed Description

template<class ImgSigT> class HxInstantiatorRightShift< ImgSigT >

Instantiator for binary pixel operation with right shift.

6.204.2 Member Data Documentation

6.204.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorRightShift::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoRightShiftInst.c

6.205 HxInstantiatorRightShiftV Class Template Reference

Instantiator for binary pixel operation with right shift.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoRightShift**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.205.1 Detailed Description

template<class ImgSigT> class HxInstantiatorRightShiftV< ImgSigT >

Instantiator for binary pixel operation with right shift.

6.205.2 Member Data Documentation

6.205.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorRightShiftV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoRightShiftInst.c

6.206 HxInstantiatorRound Class Template Reference

Instantiator for unary pixel operation with round.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoRound**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.206.1 Detailed Description

template<class ImgSigT> class HxInstantiatorRound< ImgSigT >

Instantiator for unary pixel operation with round.

6.206.2 Member Data Documentation

6.206.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoRound<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorRound::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoRoundInst.c

6.207 HxInstantiatorSin Class Template Reference

Instantiator for unary pixel operation with sine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoSin**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.207.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSin< ImgSigT >

Instantiator for unary pixel operation with sine.

6.207.2 Member Data Documentation

6.207.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoSin<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorSin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoSinInst.c

6.208 HxInstantiatorSinh Class Template Reference

Instantiator for unary pixel operation with hyperbolic sine.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoSinh**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.208.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSinh< ImgSigT >

Instantiator for unary pixel operation with hyperbolic sine.

6.208.2 Member Data Documentation

6.208.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoSinh<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorSinh::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoSinhInst.c

6.209 HxInstantiatorSqrDst Struct Template Reference

Instantiator for binary pixel operation with squared distance.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoSqrDst**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.209.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorSqrDst< ImgSigT >

Instantiator for binary pixel operation with squared distance.

6.209.2 Member Data Documentation

6.209.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSqrDst::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxSquaredDistance.c

6.210 HxInstantiatorSqrt Class Template Reference

Instantiator for unary pixel operation with square root.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoSqrt**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.210.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSqrt< ImgSigT >

Instantiator for unary pixel operation with square root.

6.210.2 Member Data Documentation

6.210.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoSqrt<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorSqrt::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoSqrtInst.c

6.211 HxInstantiatorSub Class Template Reference

Instantiator for binary pixel operation with subtraction.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoSub**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.211.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSub< ImgSigT >

Instantiator for binary pixel operation with subtraction.

6.211.2 Member Data Documentation

6.211.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoSub<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSub::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoSubInst.c

6.212 HxInstantiatorSubV Class Template Reference

Instantiator for binary pixel operation with subtraction.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoSub**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.212.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSubV< ImgSigT >

Instantiator for binary pixel operation with subtraction.

6.212.2 Member Data Documentation

6.212.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorSubV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoSubInst.c

6.213 HxInstantiatorSum Class Template Reference

Instantiator for unary pixel operation with sum.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoSum**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.213.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorSum< DstSigT, ImgSigT >

Instantiator for unary pixel operation with sum.

6.213.2 Member Data Documentation

6.213.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoSum<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSum::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoSumInst.c

6.214 HxInstantiatorSup Class Template Reference

Instantiator for binary pixel operation with supremum.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoSup**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.214.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorSup**< **ImgSigT** >

Instantiator for binary pixel operation with supremum.

6.214.2 Member Data Documentation

6.214.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoSup<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSup::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoSupInst.c

6.215 HxInstantiatorSupReduce Class Template Reference

Instantiator for reduce operation with supremum.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxReduceAdaptor< **HxBpoSupAssign**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.215.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorSupReduce**< **ImgSigT** >

Instantiator for reduce operation with supremum.

6.215.2 Member Data Documentation

6.215.2.1 template<class ImgSigT> HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorSupReduce::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxReduceSupAssignInst.c

6.216 HxInstantiatorSupV Class Template Reference

Instantiator for binary pixel operation with supremum.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoSup**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.216.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSupV< ImgSigT >

Instantiator for binary pixel operation with supremum.

6.216.2 Member Data Documentation

6.216.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorSupV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoSupInst.c

6.217 HxInstantiatorTan Class Template Reference

Instantiator for unary pixel operation with tangent.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoTan**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.217.1 Detailed Description

template<class ImgSigT> class HxInstantiatorTan< ImgSigT >

Instantiator for unary pixel operation with tangent.

6.217.2 Member Data Documentation

6.217.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoTan<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorTan::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoTanInst.c

6.218 HxInstantiatorTanh Class Template Reference

Instantiator for unary pixel operation with hyperbolic tangent.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoTanh**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.218.1 Detailed Description

template<class ImgSigT> class HxInstantiatorTanh< ImgSigT >

Instantiator for unary pixel operation with hyperbolic tangent.

6.218.2 Member Data Documentation

6.218.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoTanh<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxInstantiatorTanh::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoTanhInst.c

6.219 HxInstantiatorTriStateThreshold Struct Template Reference

Instantiator for unary pixel operation with tri state threshold.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, **HxUpoTriStateThreshold**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.219.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorTriStateThreshold< ImgSigT >

Instantiator for unary pixel operation with tri state threshold.

6.219.2 Member Data Documentation

6.219.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorTriStateThreshold::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxTriStateThreshold.c

6.220 HxInstantiatorUpoMax Class Template Reference

Instantiator for unary pixel operation with unary maximum.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoMax**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.220.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorUpoMax< DstSigT, ImgSigT >

Instantiator for unary pixel operation with unary maximum.

6.220.2 Member Data Documentation

6.220.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoMax<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorUpoMax::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoMaxInst.c

6.221 HxInstantiatorUpoMin Class Template Reference

Instantiator for unary pixel operation with unary minimum.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **HxUpoMin**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.221.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorUpoMin< DstSigT, ImgSigT >

Instantiator for unary pixel operation with unary minimum.

6.221.2 Member Data Documentation

6.221.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoMin<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorUpoMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoMinInst.c

6.222 HxInstantiatorXor Class Template Reference

Instantiator for binary pixel operation with exclusive or.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoXor**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.222.1 Detailed Description

template<class ImgSigT> class HxInstantiatorXor< ImgSigT >

Instantiator for binary pixel operation with exclusive or.

6.222.2 Member Data Documentation

6.222.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoXor<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorXor::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoXorInst.c

6.223 HxInstantiatorXorV Class Template Reference

Instantiator for binary pixel operation with exclusive or.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, **HxBpoXor**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > **f**

  Instantiate image functor.

6.223.1 Detailed Description

template<class ImgSigT> class HxInstantiatorXorV< ImgSigT >

Instantiator for binary pixel operation with exclusive or.

6.223.2 Member Data Documentation

6.223.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpo-Bind2Val<typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > > HxInstantiatorXorV::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoXorInst.c

6.224 HxInstMulAddAss2d Class Template Reference

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 2d kernel.

Public Attributes

- **HxImgFtorGenConv2d**< ImgSigT, KerSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel2d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.224.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstMulAddAss2d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 2d kernel.

6.224.2 Member Data Documentation

6.224.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2d<ImgSigT, KerSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel2d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstMulAddAss2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

6.225 HxInstMulAddAss2dK1d Class Template Reference

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv2dK1d**< ImgSigT, KerSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.225.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstMulAddAss2dK1d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 1d kernel.

6.225.2 Member Data Documentation

6.225.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dK1d<ImgSigT, KerSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstMulAddAss2dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

6.226 HxInstMulAddAss3d Class Template Reference

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 3d kernel.

Public Attributes

- **HxImgFtorGenConv3d**< ImgSigT, KerSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel3d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.226.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstMulAddAss3d< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 3d kernel.

6.226.2 Member Data Documentation

6.226.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv3d<ImgSigT, KerSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel3d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstMulAddAss3d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

6.227 HxInstMulAddAss3dK1d Class Template Reference

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 1d kernel.

Public Attributes

- **HxImgFtorGenConv3dK1d**< DstSigT, SrcSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename SrcSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

6.227.1 Detailed Description

template<class DstSigT, class SrcSigT, class KerSigT> class HxInstMulAddAss3dK1d< DstSigT, SrcSigT, KerSigT >

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 1d kernel.

6.227.2 Member Data Documentation

6.227.2.1 template<class DstSigT, class SrcSigT, class KerSigT> HxImgFtorGenConv3dK1d<DstSigT, SrcSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename SrcSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType>, HxKernel1d<typename KerSigT::DataPtrType, typename KerSigT::ArithType> > HxInstMulAddAss3dK1d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

6.228 HxLocalInterpol Class Reference

Class definition of a functor that interpolates data points to compute knots and control points for a cubic BSpline.

`#include <HxLocalInterpol.h>`

Public Methods

- **HxLocalInterpol** ()

  Default ctor: consistent but useless object.

- **HxLocalInterpol** (int degree, const HxPointSetR2 &inputData, int closed=0)

  Create and initialize functor with given data points and curve type.

- ~**HxLocalInterpol** ()

  Destructor.

- HxPointSetR2 **allP** () const

  Get all generated control points.

- int **numP** () const

  The number of control points.

- vector< double > **allKnots** () const

  Get generated knots vector.

- int **numKnots** () const

  The number of knots.

- STD_OSTREAM & **dump** (ostream &) const

6.228.1 Detailed Description

Class definition of a functor that interpolates data points to compute knots and control points for a cubic BSpline.

Based on section 9.3.4 of ”The NURBS book”, ”PIEGL, L. and TILLER, W.”, Springer, 1997.

This could be used as a **HxBSplineCurve** (p. 213) constructor, but that would require a more general version.

6.228.2 Constructor & Destructor Documentation

6.228.2.1 HxLocalInterpol::HxLocalInterpol ()

Default ctor: consistent but useless object.

```
19 {
20     makeDefault();
21 }
```

6.228.2.2 HxLocalInterpol::HxLocalInterpol (int *degree*, const HxPointSetR2 & *p*, int *closed* = 0)

Create and initialize functor with given data points and curve type.

```
25 {
26     if ( degree != 3 ) {
27         message("(constructor) not cubic - setting degree=3");
28         degree =3;
29     }
30
31     if ( p.size() <= 0 ) {
32         message("(constructor) no data points - making default");
33         makeDefault();
34         return;
35     }
36
37     _degree = degree;
38     _data = p;
39     _closed = closed;
40
41     _n = _data.size() -1;
42     if ( _closed )
43         _initClosed();
44     else
45         _initOpen();
46 }
```

6.228.2.3 HxLocalInterpol::~HxLocalInterpol ()

Destructor.

```
49 {
50 }
```

6.228.3 Member Function Documentation

6.228.3.1 HxPointSetR2 HxLocalInterpol::allP () const [inline]

Get all generated control points.

```
99 {
100     return _points;
101 }
```

6.228.3.2 int HxLocalInterpol::numP () const [inline]

The number of control points.

```
105 {
106     return _points.size();
107 }
```

6.228.3.3 vector< double > HxLocalInterpol::allKnots () const [inline]

Get generated knots vector.

```
111 {
112     return _knots;
113 }
```

6.228.3.4 int HxLocalInterpol::numKnots () const [inline]

The number of knots.

```
117 {
118     return _knots.size();
119 }
```

The documentation for this class was generated from the following files:

- **HxLocalInterpol.h**
- HxLocalInterpol.c

6.229 HxMatrix Class Reference

Class definition for matrices.

`#include <HxMatrix.h>`

Constructors

- **HxMatrix** ()

  Empty matrix.

- **HxMatrix** (int nrow, int ncol)

  Empty matrix with given number of rows and columns.

- **HxMatrix** (int nrow, int ncol, double a)

  Matrix with constant value.

- **HxMatrix** (int nrow, int ncol, double *data)

  Matrix with given data.

- **HxMatrix** (const HxMatrix &m)

  Copy constructor.

- **HxMatrix** (const **HxVector** &v)

  Copy from vector constructor.

- **HxMatrix** (const **HxVector** &v1, const **HxVector** &v2)

  Copy from 2 vectors constructor.

- **HxMatrix** (const **HxVector** &v1, const **HxVector** &v2, const **HxVector** &v3)

  Copy from 3 vectors constructor.

- **HxMatrix** (const **HxVector** &v1, const **HxVector** &v2, const **HxVector** &v3, const **HxVector** &v4)

  Copy from 4 vectors constructor.

- **HxMatrix** (const **HxVector** &v1, const **HxVector** &v2, const **HxVector** &v3, const **HxVector** &v4, const **HxVector** &v5)

  Copy from 5 vectors constructor.

- **HxMatrix** (const **HxVector** &v1, const **HxVector** &v2, const **HxVector** &v3, const **HxVector** &v4, const **HxVector** &v5, const **HxVector** &v6)

  Copy from 6 vectors constructor.

Inquiry

- int **nRow** () const

  Number of rows.

- int **nCol** () const

  Number of columns.

- int **nElem** () const

  Number of elements.

- int **valid** () const

  Indicates whether the matrix is valid.

Operators

- HxMatrix & **operator=** (double a)

  Assign constant value.

- HxMatrix & **operator=** (const HxMatrix &m)

  Normal assigment.

- double * **operator[ ]** (int i) const

  Subscripting, start with 0.

- HxMatrix **operator-** () const

  Unary minus.

- HxMatrix **operator** * (const HxMatrix &a, double b)

  Multiplication.

- HxMatrix **operator** * (double a, const HxMatrix &b)

Multiplication.

- HxMatrix **operator** ∗ (const HxMatrix &a, const HxMatrix &b)

  Multiplication.

- **HxVector operator** ∗ (const **HxVector** &a, const HxMatrix &b)

  Multiplication.

- **HxVector operator** ∗ (const HxMatrix &a, const **HxVector** &b)

  Multiplication.

- HxMatrix **operator/** (const HxMatrix &a, double b)

  Division.

- HxMatrix **operator/** (double a, const HxMatrix &b)

  Division.

- HxMatrix **operator+** (const HxMatrix &a, const HxMatrix &b)

  Addition.

- HxMatrix **operator+** (const HxMatrix &a, double b)

  Addition.

- HxMatrix **operator+** (double a, const HxMatrix &b)

  Addition.

- HxMatrix **operator-** (const HxMatrix &a, const HxMatrix &b)

  Subtraction.

- HxMatrix **operator-** (const HxMatrix &a, double b)

  Subtraction.

- HxMatrix **operator-** (double a, const HxMatrix &b)

  Subtraction.

- int **operator==** (const HxMatrix &a, const HxMatrix &b)

  Equal.

- int **operator!=** (const HxMatrix &a, const HxMatrix &b)

  Not equal.

- **HxVec3Double operator** ∗ (const **HxVec3Double** &a, const HxMatrix &b)

  Multiplication.

- **HxVec3Double operator** ∗ (const HxMatrix &a, const **HxVec3Double** &b)

  Multiplication.

Operations

- HxMatrix **i** () const
  Inverse.
- HxMatrix **t** () const
  Transpose.
- HxMatrix **svd** (**HxVector** &W, HxMatrix &V) const
  Singular Value Decomposition.
- HxMatrix **add** (const HxMatrix &b) const
  Addition.
- HxMatrix **add** (const double val) const
  Addition.
- HxMatrix **sub** (const HxMatrix &b) const
  Subtraction.
- HxMatrix **sub** (const double val) const
  Subtraction.
- HxMatrix **mul** (const HxMatrix &b) const
  Multiplication.
- HxMatrix **mul** (const **HxVector** &v) const
  Multiplication.
- HxMatrix **mul** (const double val) const
  Multiplication.
- HxMatrix **div** (const double val) const
  Division.
- HxMatrix **sin** () const
  Apply sin to each element.
- HxMatrix **cos** () const
  Apply cos to each element.
- HxMatrix **tan** () const
  Apply tan to each element.
- HxMatrix **sinh** () const
  Apply sinh to each element.
- HxMatrix **cosh** () const
  Apply cosh to each element.

- HxMatrix **tanh** () const
  Apply tanh to each element.
- HxMatrix **exp** () const
  Apply exp to each element.
- HxMatrix **log** () const
  Apply log to each element.
- HxMatrix **sqrt** () const
  Apply sqrt to each element.
- HxMatrix **abs** () const
  Apply abs to each element.
- HxMatrix **sgn** () const
  Apply sgn to each element.
- HxMatrix **map** (double(∗f)(double)) const
  Map f to each element of this.

Matrix generation

Generate coordinate transformation matrices for postfix vector multiplication

- HxMatrix **translate2d** (double x, double y)
  Translation in 2D.
- HxMatrix **scale2d** (double sx, double sy)
  Scaling in 2D.
- HxMatrix **rotate2d** (double alpha)
  Rotation in 2D (alpha in rad).
- HxMatrix **rotate2dDeg** (double alpha)
  Rotation in 2D (alpha in deg).
- HxMatrix **reflect2d** (int doX, int doY)
  Reflection in 2D (if (doX != 0) reflect X), etc.
- HxMatrix **shear2d** (double sx, double sy)
  Shearing in 2D.
- HxMatrix **translate3d** (double x, double y, double z)
  Translation in 3D.
- HxMatrix **scale3d** (double sx, double sy, double sz)
  Scaling in 3D.

- HxMatrix **rotateX3d** (double alpha)

  Rotation around X-axis in 3D (alpha in rad).

- HxMatrix **rotateX3dDeg** (double alpha)

  Rotation around X-axis in 3D (alpha in deg).

- HxMatrix **rotateY3d** (double alpha)

  Rotation around Y-axis in 3D (alpha in rad).

- HxMatrix **rotateY3dDeg** (double alpha)

  Rotation around Y-axis in 3D (alpha in deg).

- HxMatrix **rotateZ3d** (double alpha)

  Rotation around Z-axis in 3D (alpha in rad).

- HxMatrix **rotateZ3dDeg** (double alpha)

  Rotation around Z-axis in 3D (alpha in deg).

- HxMatrix **reflect3d** (int doX, int doY, int doZ)

  Reflection in 3D (if (doX != 0) reflect X), etc.

- HxMatrix **projection** (double f)

  Projection matrix.

- HxMatrix **camera** (double f)

  Camera transformation.

- HxMatrix **lift2dTo3dXY** ()

  Lift 2D plane to 3D XY-plane.

Public Methods

- **~HxMatrix** ()
- std::ostream & **put** (std::ostream &os) const

Friends

- class **HxVector**

6.229.1 Detailed Description

Class definition for matrices.

The dimensions are of arbitrary size.

6.229.2 Constructor & Destructor Documentation

6.229.2.1 HxMatrix::HxMatrix () [inline]

Empty matrix.

```
323 {
324     _nr = 0;
325     _nc = 0;
326     _data = 0;
327 }
```

6.229.2.2 HxMatrix::HxMatrix (int *nRow*, int *nCol*) [inline]

Empty matrix with given number of rows and columns.

```
331 {
332     _nr = nRow;
333     _nc = nCol;
334     _data = new double[_nr * _nc];
335 }
```

6.229.2.3 HxMatrix::HxMatrix (int *nRow*, int *nCol*, double *a*) [inline]

Matrix with constant value.

```
339 {
340     _nr = nRow;
341     _nc = nCol;
342     _data = new double[_nr * _nc];
343 
344     double* t = _data;
345     int i = nElem();
346     while (--i >= 0)
347         *t++ = a;
348 }
```

6.229.2.4 HxMatrix::HxMatrix (int *nRow*, int *nCol*, double * *data*) [inline]

Matrix with given data.

```
352 {
353     _nr = nRow;
354     _nc = nCol;
355     _data = data;
356 }
```

6.229.2.5 HxMatrix::HxMatrix (const HxMatrix & *m*) [inline]

Copy constructor.

```
359 {
360     _nr = m.nRow();
361     _nc = m.nCol();
362     _data = new double[_nr * _nc];
363 
364     double* t = m._data;
365     double* u = _data;
366     int i = m.nElem();
367     while (--i >= 0)
368         *u++ = *t++;
369 }
```

6.229.2.6 HxMatrix::HxMatrix (const HxVector & *v*)

Copy from vector constructor.

```
35 {
36     _nc = v.nElem();
37     _nr = 1;
38     _data = new double[_nr * _nc];
39 
40     double* t = v._data;
41     double* u = _data;
42     int i = v.nElem();
43     while (--i >= 0)
44         *u++ = *t++;
45 }
```

6.229.2.7 HxMatrix::HxMatrix (const HxVector & *v1*, const HxVector & *v2*)

Copy from 2 vectors constructor.

```
48 {
49     if (v1.nElem() != v2.nElem()) {
50         error("differently sized input vectors for matrix construction.");
51         _nr = 0; _nc = 0; _data = 0;
52         return;
53     }
54     _nc = v1.nElem();
55     _nr = 2;
56     _data = new double[_nr * _nc];
57 
58     double* t = v1._data;
59     double* u = _data;
60     int i = v1.nElem();
61     while (--i >= 0)
62         *u++ = *t++;
63 
64     i = v2.nElem();
65     t = v2._data;
66     while (--i >= 0)
67         *u++ = *t++;
68 }
```

6.229.2.8 HxMatrix::HxMatrix (const HxVector & *v1*, const HxVector & *v2*, const HxVector & *v3*)

Copy from 3 vectors constructor.

```
71 {
72     if ((v1.nElem() != v2.nElem()) || (v1.nElem() != v3.nElem())) {
73         error("differently sized input vectors for matrix construction.");
74         _nr = 0; _nc = 0; _data = 0;
75         return;
76     }
77     _nc = v1.nElem();
78     _nr = 3;
79     _data = new double[_nr * _nc];
80 
81     double* t = v1._data;
82     double* u = _data;
83     int i = v1.nElem();
84     while (--i >= 0)
85         *u++ = *t++;
86 
87     i = v2.nElem();
88     t = v2._data;
89     while (--i >= 0)
90         *u++ = *t++;
91 
92     i = v3.nElem();
93     t = v3._data;
94     while (--i >= 0)
95         *u++ = *t++;
96 }
```

6.229.2.9 HxMatrix::HxMatrix (const HxVector & *v1*, const HxVector & *v2*, const HxVector & *v3*, const HxVector & *v4*)

Copy from 4 vectors constructor.

```
100 {
101     if ((v1.nElem() != v2.nElem()) || (v1.nElem() != v3.nElem()) ||
102         (v1.nElem() != v4.nElem())) {
103         error("differently sized input vectors for matrix construction.");
104         _nr = 0; _nc = 0; _data = 0;
105         return;
106     }
107     _nc = v1.nElem();
108     _nr = 4;
109     _data = new double[_nr * _nc];
110 
111     double* t = v1._data;
112     double* u = _data;
113     int i = v1.nElem();
114     while (--i >= 0)
115         *u++ = *t++;
116 
117     i = v2.nElem();
118     t = v2._data;
119     while (--i >= 0)
120         *u++ = *t++;
121 
122     i = v3.nElem();
123     t = v3._data;
124     while (--i >= 0)
125         *u++ = *t++;
126 
127     i = v4.nElem();
128     t = v4._data;
129     while (--i >= 0)
```

```
130         *u++ = *t++;
131 }
```

6.229.2.10 HxMatrix::HxMatrix (const HxVector & *v1*, const HxVector & *v2*, const HxVector & *v3*, const HxVector & *v4*, const HxVector & *v5*)

Copy from 5 vectors constructor.

```
135 {
136     if ((v1.nElem() != v2.nElem()) || (v1.nElem() != v3.nElem()) ||
137         (v1.nElem() != v4.nElem()) || (v1.nElem() != v5.nElem())) {
138         error("differently sized input vectors for matrix construction.");
139         _nr = 0; _nc = 0; _data = 0;
140         return;
141     }
142     _nc = v1.nElem();
143     _nr = 5;
144     _data = new double[_nr * _nc];
145 
146     double* t = v1._data;
147     double* u = _data;
148     int i = v1.nElem();
149     while (--i >= 0)
150         *u++ = *t++;
151 
152     i = v2.nElem();
153     t = v2._data;
154     while (--i >= 0)
155         *u++ = *t++;
156 
157     i = v3.nElem();
158     t = v3._data;
159     while (--i >= 0)
160         *u++ = *t++;
161 
162     i = v4.nElem();
163     t = v4._data;
164     while (--i >= 0)
165         *u++ = *t++;
166 
167     i = v5.nElem();
168     t = v5._data;
169     while (--i >= 0)
170         *u++ = *t++;
171 }
```

6.229.2.11 HxMatrix::HxMatrix (const HxVector & *v1*, const HxVector & *v2*, const HxVector & *v3*, const HxVector & *v4*, const HxVector & *v5*, const HxVector & *v6*)

Copy from 6 vectors constructor.

```
175 {
176     if ((v1.nElem() != v2.nElem()) || (v1.nElem() != v3.nElem()) ||
177         (v1.nElem() != v4.nElem()) || (v1.nElem() != v5.nElem()) ||
178         (v1.nElem() != v6.nElem())) {
179         error("differently sized input vectors for matrix construction.");
180         _nr = 0; _nc = 0; _data = 0;
181         return;
```

```
182     }
183     _nc = v1.nElem();
184     _nr = 6;
185     _data = new double[_nr * _nc];
186
187     double* t = v1._data;
188     double* u = _data;
189     int i = v1.nElem();
190     while (--i >= 0)
191         *u++ = *t++;
192
193     i = v2.nElem();
194     t = v2._data;
195     while (--i >= 0)
196         *u++ = *t++;
197
198     i = v3.nElem();
199     t = v3._data;
200     while (--i >= 0)
201         *u++ = *t++;
202
203     i = v4.nElem();
204     t = v4._data;
205     while (--i >= 0)
206         *u++ = *t++;
207
208     i = v5.nElem();
209     t = v5._data;
210     while (--i >= 0)
211         *u++ = *t++;
212
213     i = v6.nElem();
214     t = v6._data;
215     while (--i >= 0)
216         *u++ = *t++;
217 }
```

6.229.3 Member Function Documentation

6.229.3.1 HxMatrix HxMatrix::translate2d (double *x*, double *y*) [static]

Translation in 2D.

```
221 {
222     HxMatrix m(3,3);
223     m[0][0] = 1; m[0][1] = 0; m[0][2] = x;
224     m[1][0] = 0; m[1][1] = 1; m[1][2] = y;
225     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1;
226     /* prefix:
227     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0;
228     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0;
229     m[2][0] = x; m[2][1] = y; m[2][2] = 1;
230     */
231     return m;
232 }
```

6.229.3.2 HxMatrix HxMatrix::scale2d (double *sx*, double *sy*) [static]

Scaling in 2D.

```
236 {
237     HxMatrix m(3,3);
238     m[0][0] = sx; m[0][1] =  0; m[0][2] = 0;
239     m[1][0] =  0; m[1][1] = sy; m[1][2] = 0;
240     m[2][0] =  0; m[2][1] =  0; m[2][2] = 1;
241     return m;
242 }
```

6.229.3.3 HxMatrix HxMatrix::rotate2d (double *alpha*) [static]

Rotation in 2D (alpha in rad).

```
246 {
247     HxMatrix m(3,3);
248     m[0][0] = ::cos(alpha); m[0][1] = -::sin(alpha); m[0][2] = 0;
249     m[1][0] = ::sin(alpha); m[1][1] = ::cos(alpha);  m[1][2] = 0;
250     m[2][0] = 0;            m[2][1] = 0;             m[2][2] = 1;
251     /* prefix:
252     m[0][0] =  ::cos(alpha); m[0][1] = ::sin(alpha); m[0][2] = 0;
253     m[1][0] = -::sin(alpha); m[1][1] = ::cos(alpha); m[1][2] = 0;
254     m[2][0] = 0;             m[2][1] = 0;            m[2][2] = 1;
255     */
256     return m;
257 }
```

6.229.3.4 HxMatrix HxMatrix::rotate2dDeg (double *alpha*) [static]

Rotation in 2D (alpha in deg).

```
261 {
262     return rotate2d(M_PI*alpha/180.0);
263 }
```

6.229.3.5 HxMatrix HxMatrix::reflect2d (int *doX*, int *doY*) [static]

Reflection in 2D (if (doX != 0) reflect X), etc.

```
267 {
268     double rx = (doX) ? -1 : 1;
269     double ry = (doY) ? -1 : 1;
270     HxMatrix m(3,3);
271     m[0][0] = rx; m[0][1] =  0; m[0][2] = 0;
272     m[1][0] =  0; m[1][1] = ry; m[1][2] = 0;
273     m[2][0] =  0; m[2][1] =  0; m[2][2] = 1;
274     return m;
275 }
```

6.229.3.6 HxMatrix HxMatrix::shear2d (double *sx*, double *sy*) [static]

Shearing in 2D.

```
279 {
280     HxMatrix m(3,3);
281     m[0][0] =  1; m[0][1] = sx; m[0][2] = 0;
282     m[1][0] = sy; m[1][1] =  1; m[1][2] = 0;
283     m[2][0] =  0; m[2][1] =  0; m[2][2] = 1;
284     /* prefix:
285     m[0][0] =  1; m[0][1] = sy; m[0][2] = 0;
286     m[1][0] = sx; m[1][1] =  1; m[1][2] = 0;
287     m[2][0] =  0; m[2][1] =  0; m[2][2] = 1;
288     */
289     return m;
290 }
```

6.229.3.7 HxMatrix HxMatrix::translate3d (double *x*, double *y*, double *z*) [static]

Translation in 3D.

```
294 {
295     HxMatrix m(4,4);
296     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0; m[0][3] = x;
297     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0; m[1][3] = y;
298     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1; m[2][3] = z;
299     m[3][0] = 0; m[3][1] = 0; m[3][2] = 0; m[3][3] = 1;
300     /* prefix:
301     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0; m[0][3] = 0;
302     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0; m[1][3] = 0;
303     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1; m[2][3] = 0;
304     m[3][0] = x; m[3][1] = y; m[3][2] = z; m[3][3] = 1;
305     */
306     return m;
307 }
```

6.229.3.8 HxMatrix HxMatrix::scale3d (double *sx*, double *sy*, double *sz*) [static]

Scaling in 3D.

```
311 {
312     HxMatrix m(4,4);
313     m[0][0] = sx; m[0][1] =  0; m[0][2] =  0; m[0][3] = 0;
314     m[1][0] =  0; m[1][1] = sy; m[1][2] =  0; m[1][3] = 0;
315     m[2][0] =  0; m[2][1] =  0; m[2][2] = sz; m[2][3] = 0;
316     m[3][0] =  0; m[3][1] =  0; m[3][2] =  0; m[3][3] = 1;
317     return m;
318 }
```

6.229.3.9 HxMatrix HxMatrix::rotateX3d (double *alpha*) [static]

Rotation around X-axis in 3D (alpha in rad).

```
322 {
323     HxMatrix m(4,4);
324     m[0][0] = 1; m[0][1] = 0;           m[0][2] = 0;            m[0][3] = 0;
325     m[1][0] = 0; m[1][1] = ::cos(alpha); m[1][2] = -::sin(alpha); m[1][3] = 0;
326     m[2][0] = 0; m[2][1] = ::sin(alpha); m[2][2] = ::cos(alpha);  m[2][3] = 0;
327     m[3][0] = 0; m[3][1] = 0;           m[3][2] = 0;            m[3][3] = 1;
```

```
328     /* prefix
329     m[0][0] = 1; m[0][1] = 0;             m[0][2] = 0;            m[0][3] = 0;
330     m[1][0] = 0; m[1][1] = ::cos(alpha);  m[1][2] = ::sin(alpha); m[1][3] = 0;
331     m[2][0] = 0; m[2][1] = -::sin(alpha); m[2][2] = ::cos(alpha); m[2][3] = 0;
332     m[3][0] = 0; m[3][1] = 0;             m[3][2] = 0;            m[3][3] = 1;
333     */
334     return m;
335 }
```

6.229.3.10 HxMatrix HxMatrix::rotateX3dDeg (double *alpha*) [static]

Rotation around X-axis in 3D (alpha in deg).

```
339 {
340     return rotateX3d(M_PI*alpha/180.0);
341 }
```

6.229.3.11 HxMatrix HxMatrix::rotateY3d (double *alpha*) [static]

Rotation around Y-axis in 3D (alpha in rad).

```
345 {
346     HxMatrix m(4,4);
347     alpha = M_PI*alpha/180.0;
348     m[0][0] = ::cos(alpha);  m[0][1] = 0; m[0][2] = ::sin(alpha); m[0][3] = 0;
349     m[1][0] = 0;             m[1][1] = 1; m[1][2] = 0;            m[1][3] = 0;
350     m[2][0] = -::sin(alpha); m[2][1] = 0; m[2][2] = ::cos(alpha); m[2][3] = 0;
351     m[3][0] = 0;             m[3][1] = 0; m[3][2] = 0;            m[3][3] = 1;
352     /* prefix:
353     m[0][0] = ::cos(alpha); m[0][1] = 0; m[0][2] = -::sin(alpha); m[0][3] = 0;
354     m[1][0] = 0;            m[1][1] = 1; m[1][2] = 0;             m[1][3] = 0;
355     m[2][0] = ::sin(alpha); m[2][1] = 0; m[2][2] = ::cos(alpha);  m[2][3] = 0;
356     m[3][0] = 0;            m[3][1] = 0; m[3][2] = 0;             m[3][3] = 1;
357     */
358     return m;
359 }
```

6.229.3.12 HxMatrix HxMatrix::rotateY3dDeg (double *alpha*) [static]

Rotation around Y-axis in 3D (alpha in deg).

```
363 {
364     return rotateY3d(M_PI*alpha/180.0);
365 }
```

6.229.3.13 HxMatrix HxMatrix::rotateZ3d (double *alpha*) [static]

Rotation around Z-axis in 3D (alpha in rad).

```
369 {
370     HxMatrix m(4,4);
371     m[0][0] = ::cos(alpha); m[0][1] = -::sin(alpha); m[0][2] = 0; m[0][3] = 0;
```

```
372     m[1][0] = ::sin(alpha); m[1][1] = ::cos(alpha);  m[1][2] = 0; m[1][3] = 0;
373     m[2][0] = 0;            m[2][1] = 0;             m[2][2] = 1; m[2][3] = 0;
374     m[3][0] = 0;            m[3][1] = 0;             m[3][2] = 0; m[3][3] = 1;
375     /* prefix:
376     m[0][0] = ::cos(alpha);  m[0][1] = ::sin(alpha); m[0][2] = 0; m[0][3] = 0;
377     m[1][0] = -::sin(alpha); m[1][1] = ::cos(alpha); m[1][2] = 0; m[1][3] = 0;
378     m[2][0] = 0;             m[2][1] = 0;            m[2][2] = 1; m[2][3] = 0;
379     m[3][0] = 0;             m[3][1] = 0;            m[3][2] = 0; m[3][3] = 1;
380     */
381     return m;
382 }
```

6.229.3.14 HxMatrix HxMatrix::rotateZ3dDeg (double *alpha*) [static]

Rotation around Z-axis in 3D (alpha in deg).

```
386 {
387     return rotateZ3d(M_PI*alpha/180.0);
388 }
```

6.229.3.15 HxMatrix HxMatrix::reflect3d (int *doX*, int *doY*, int *doZ*) [static]

Reflection in 3D (if (doX != 0) reflect X), etc.

```
392 {
393     double rx = (doX) ? -1 : 1;
394     double ry = (doY) ? -1 : 1;
395     double rz = (doZ) ? -1 : 1;
396     HxMatrix m(4,4);
397     m[0][0] = rx; m[0][1] =  0; m[0][2] =  0; m[0][3] = 0;
398     m[1][0] =  0; m[1][1] = ry; m[1][2] =  0; m[1][3] = 0;
399     m[2][0] =  0; m[2][1] =  0; m[2][2] = rz; m[2][3] = 0;
400     m[3][0] =  0; m[3][1] =  0; m[3][2] =  0; m[3][3] = 1;
401     return m;
402 }
```

6.229.3.16 HxMatrix HxMatrix::projection (double *f*) [static]

Projection matrix.

```
406 {
407     HxMatrix m(4,4);
408     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0; m[0][3] = 0;
409     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0; m[1][3] = 0;
410     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1; m[2][3] = 1/f;
411     m[3][0] = 0; m[3][1] = 0; m[3][2] = 0; m[3][3] = 1;
412     /* prefix:
413     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0;   m[0][3] = 0;
414     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0;   m[1][3] = 0;
415     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1;   m[2][3] = 0;
416     m[3][0] = 0; m[3][1] = 0; m[3][2] = 1/f; m[3][3] = 1;
417     */
418     return m;
419 }
```

6.229.3.17 HxMatrix HxMatrix::camera (double *f*) [static]

Camera transformation.

```
423 {
424     HxMatrix m(3,4);
425     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0;   m[0][3] = 0;
426     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0;   m[1][3] = 0;
427     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1/f; m[2][3] = 1;
428     /* prefix:
429     HxMatrix m(4,3);
430     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0;
431     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0;
432     m[2][0] = 0; m[2][1] = 0; m[2][2] = 1/f;
433     m[3][0] = 0; m[3][1] = 0; m[3][2] = 1;
434     */
435     return m;
436 }
```

6.229.3.18 HxMatrix HxMatrix::lift2dTo3dXY () [static]

Lift 2D plane to 3D XY-plane.

```
440 {
441     HxMatrix m(4,3);
442     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0;
443     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0;
444     m[2][0] = 0; m[2][1] = 0; m[2][2] = 0;
445     m[3][0] = 0; m[3][1] = 0; m[3][2] = 1;
446     /* prefix:
447     HxMatrix m(3,4);
448     m[0][0] = 1; m[0][1] = 0; m[0][2] = 0; m[0][3] = 0;
449     m[1][0] = 0; m[1][1] = 1; m[1][2] = 0; m[1][3] = 0;
450     m[2][0] = 0; m[2][1] = 0; m[2][2] = 0; m[2][3] = 1;
451     */
452     return m;
453 }
```

6.229.3.19 int HxMatrix::nRow () const [inline]

Number of rows.

```
378 {
379     return _nr;
380 }
```

6.229.3.20 int HxMatrix::nCol () const [inline]

Number of columns.

```
384 {
385     return _nc;
386 }
```

6.229.3.21 int HxMatrix::nElem () const [inline]

Number of elements.

```
390 {
391     return _nr * _nc;
392 }
```

6.229.3.22 int HxMatrix::valid () const [inline]

Indicates whether the matrix is valid.

```
396 {
397     return ((_nc != 0) && (_nr != 0));
398 }
```

6.229.3.23 HxMatrix & HxMatrix::operator= (double *a*) [inline]

Assign constant value.

```
402 {
403     int i = nElem();
404     double *t = _data;
405     while (--i >= 0)
406         *t++ = a;
407     return *this;
408 }
```

6.229.3.24 HxMatrix & HxMatrix::operator= (const HxMatrix & *m*) [inline]

Normal assigment.

```
412 {
413     if (this != &m) {
414         delete [] _data;
415         _nr = m.nRow();
416         _nc = m.nCol();
417         _data = new double [_nr * _nc];
418         double *t = _data;
419         double *u = m._data;
420         int i = m.nElem();
421         while (--i >= 0)
422             *t++ = *u++;
423     }
424     return *this;
425 }
```

6.229.3.25 double * HxMatrix::operator[ ] (int *i*) const [inline]

Subscripting, start with 0.

```
429 {
430     return &_data[i*_nc];
431 }
```

6.229.3.26 HxMatrix HxMatrix::operator- () const [inline]

Unary minus.

```
435 {
436     HxMatrix m(*this);
437     double* t = m._data;
438     double* u = _data;
439     int i = nElem();
440     while (--i >= 0)
441         *t++ = -(*u++);
442     return m;
443 }
```

6.229.3.27 HxMatrix HxMatrix::i () const

Inverse.

```
591 {
592     if (nRow() != nCol()) {
593         error("Inverse: matrix is not square!.");
594         return HxMatrix(0, 0);
595     }
596 
597     int size = nRow();
598     HxMatrix m(size,size), tmp(*this);
599     short* idx = new short [size];
600     double d;
601 
602     if (!ludcmp(tmp._data, size, idx, &d)) {
603         error( "Inverse: singular matrix can't be inverted." );
604         delete [] idx;
605         return HxMatrix(0,0);
606     }
607 
608     double* res = new double [size];
609 
610     for (int j = 0; j < size; j++) {
611 
612     // ADB 6 Feb 2001 - Fix for new 'for' scoping.
613     int i;
614 
615         for (i = 0; i < size; i++)
616             res[i] = 0.0;
617         res[j] = 1.0;
618         lubksb(tmp._data, size, idx, res);
619         for (i = 0; i < size; i++)
620             m[i][j] = res[i];
621     }
622 
623      delete [] res;
624      delete [] idx;
625 
626      return m;
627 }
```

6.229.3.28 HxMatrix HxMatrix::t () const [inline]

Transpose.

```
549 {
550     HxMatrix m(nCol(), nRow());
551     int i, j;
552     for (i=0 ; i<nRow() ; i++)
553         for (j=0 ; j<nCol() ; j++)
554             m[j][i] = (*this)[i][j];
555     return m;
556 }
```

6.229.3.29 HxMatrix HxMatrix::svd (HxVector & *W*, HxMatrix & *V*) const

Singular Value Decomposition.

m = m.svd(W,V) ∗ W.diag() ∗ V.t()

```
632 {
633     int row = nRow();
634     int col = nCol();
635 
636     if (col > row) {
637         error( "Svd: Matrix must be augmented with extra rows of zeros." );
638         W = HxVector(0);
639         V = HxMatrix(0,0);
640         return HxMatrix(0,0);
641     }
642 
643     HxMatrix U(*this);
644     W = HxVector(col);
645     V = HxMatrix(col,col);
646 
647     svdcmp(U._data, W._data, V._data, col, row);
648 
649     /* sort on eigenvalue */
650     for (int i = 0; i < col; i++) {
651         int idx = i;
652         double val = W[idx];
653 
654         for (int j = i+1; j < col; j++)
655             if( W[j] > val ) {
656                 idx = j;
657                 val = W[idx];
658             }
659 
660         if (idx != i) {
661             val = W[idx]; W[idx] = W[i]; W[i] = val;
662 
663 
664         // ADB 6 Feb 2001 - Fix for new 'for' scoping.
665         int j;
666             for (j = 0; j < col; j++) {
667                 val = V[j][idx];
668                 V[j][idx] = V[j][i];
669                 V[j][i] = val;
670             }
671             for (j = 0; j < row; j++) {
672                 val = U[j][idx];
673                 U[j][idx] = U[j][i];
674                 U[j][i] = val;
675             }
676         }
677     }
678 
```

```
679     return U;
680 }
```

6.229.3.30 HxMatrix HxMatrix::add (const HxMatrix & *b*) const

Addition.

Equivalent to : a+b

```
684 {
685     return *this+b;
686 }
```

6.229.3.31 HxMatrix HxMatrix::add (const double *val*) const

Addition.

Equivalent to : a+val

```
690 {
691     return *this+val;
692 }
```

6.229.3.32 HxMatrix HxMatrix::sub (const HxMatrix & *b*) const

Subtraction.

Equivalent to : a-b

```
696 {
697     return *this-b;
698 }
```

6.229.3.33 HxMatrix HxMatrix::sub (const double *val*) const

Subtraction.

Equivalent to : a-val

```
702 {
703     return *this-val;
704 }
```

6.229.3.34 HxMatrix HxMatrix::mul (const HxMatrix & *b*) const

Multiplication.

Equivalent to : a∗b

```
708 {
709     return *this*b;
710 }
```

6.229.3.35 HxMatrix HxMatrix::mul (const HxVector & *v*) const

Multiplication.

Equivalent to : a*v

```
720 {
721     return *this*v;
722 }
```

6.229.3.36 HxMatrix HxMatrix::mul (const double *val*) const

Multiplication.

Equivalent to : a*val

```
714 {
715     return *this*val;
716 }
```

6.229.3.37 HxMatrix HxMatrix::div (const double *val*) const

Division.

Equivalent to : a/val

```
726 {
727     return *this/val;
728 }
```

6.229.3.38 HxMatrix HxMatrix::sin () const

Apply sin to each element.

```
732 {
733     return map(::sin);
734 }
```

6.229.3.39 HxMatrix HxMatrix::cos () const

Apply cos to each element.

```
738 {
739     return map(::cos);
740 }
```

6.229.3.40 HxMatrix HxMatrix::tan () const

Apply tan to each element.

```
744 {
745     return map(::tan);
746 }
```

6.229.3.41 HxMatrix HxMatrix::sinh () const

Apply sinh to each element.

```
750 {
751     return map(::sinh);
752 }
```

6.229.3.42 HxMatrix HxMatrix::cosh () const

Apply cosh to each element.

```
756 {
757     return map(::cosh);
758 }
```

6.229.3.43 HxMatrix HxMatrix::tanh () const

Apply tanh to each element.

```
762 {
763     return map(::tanh);
764 }
```

6.229.3.44 HxMatrix HxMatrix::exp () const

Apply exp to each element.

```
768 {
769     return map(::exp);
770 }
```

6.229.3.45 HxMatrix HxMatrix::log () const

Apply log to each element.

```
774 {
775     return map(::log);
776 }
```

6.229.3.46 HxMatrix HxMatrix::sqrt () const

Apply sqrt to each element.

```
780 {
781     return map(::sqrt);
782 }
```

6.229.3.47 HxMatrix HxMatrix::abs () const

Apply abs to each element.

```
786 {
787     return map(::fabs);
788 }
```

6.229.3.48 HxMatrix HxMatrix::sgn () const

Apply sgn to each element.

```
794 {
795     return map(::sgn);
796 }
```

6.229.3.49 HxMatrix HxMatrix::map (double(∗ *f*)(double)) const [inline]

Map f to each element of this.

```
566 {
567     HxMatrix m(*this);
568     double* t = m._data;
569     double* u = _data;
570     int i = nElem();
571     while (--i >= 0)
572             *t++ = f(*u++);
573     return m;
574 }
```

6.229.4 Friends And Related Function Documentation

6.229.4.1 HxMatrix operator ∗ (const HxMatrix & *a*, double *b*) [friend]

Multiplication.

```
447 {
448     HxMatrix m(a);
449     double* t = m._data;
450     double* u = a._data;
451     int i = a.nElem();
452     while (--i >= 0)
453         *t++ = *u++ * b;
454     return m;
455 }
```

6.229.4.2 HxMatrix operator ∗ (double *a*, const HxMatrix & *b*) [friend]

Multiplication.

```
459 {
460     HxMatrix m(b);
461     double* t = m._data;
462     double* u = b._data;
463     int i = b.nElem();
464     while (--i >= 0)
465     *t++ = a * *u++;
466     return m;
467 }
```

6.229.4.3 HxMatrix operator ∗ (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Multiplication.

```
504 {
505     if (a.nCol() != b.nRow()) {
506         error("nonconformant HxMatrix * HxMatrix operands.");
507         return HxMatrix(0,0);
508     }
509     HxMatrix m(a.nRow(), b.nCol());
510     double sum;
511     int i, j, k;
512     for (i=0 ; i<a.nRow() ; i++) {
513         for (j=0 ; j<b.nCol() ; j++) {
514             sum = 0;
515             for (k=0 ; k<a.nCol() ; k++)
516                 sum += a[i][k] * b[k][j];
517             m[i][j] = sum;
518         }
519     }
520     return m;
521 }
```

6.229.4.4 HxVector operator ∗ (const HxVector & *a*, const HxMatrix & *b*) [friend]

Multiplication.

```
525 {
526     if (a.nElem() != b.nRow()) {
527         error("nonconformant HxVector * HxMatrix operands.");
528         return HxVector(0);
529     }
530     HxVector v(b.nCol());
531     double sum;
532     int i, j;
533     for (i=0 ; i<b.nCol() ; i++) {
534         sum = 0;
535         for (j=0 ; j<b.nRow() ; j++)
536             sum += a[j] * b[j][i];
537         v[i] = sum;
538     }
539     return v;
540 }
```

6.229.4.5 HxVector operator ∗ (const HxMatrix & *a*, const HxVector & *b*) [friend]

Multiplication.

```
544 {
545     if (b.nElem() != a.nCol()) {
546         error("nonconformant HxMatrix * HxVector operands.");
547         return HxVector(0);
548     }
549     HxVector v(a.nRow());
550     double sum;
551     int i, j;
552     for (i=0 ; i<a.nRow() ; i++) {
553         sum = 0;
554         for (j=0 ; j<a.nCol() ; j++)
555             sum += a[i][j] * b[j];
556         v[i] = sum;
557     }
558     return v;
559 }
```

6.229.4.6 HxMatrix operator/ (const HxMatrix & *a*, double *b*) [friend]

Division.

```
471 {
472     HxMatrix m(a);
473     double* t = m._data;
474     double* u = a._data;
475     int i = a.nElem();
476     while (--i >= 0)
477         *t++ = *u++ / b;
478     return m;
479 }
```

6.229.4.7 HxMatrix operator/ (double *a*, const HxMatrix & *b*) [friend]

Division.

```
483 {
484     HxMatrix m(b);
485     double* t = m._data;
486     double* u = b._data;
487     int i = b.nElem();
488     while (--i >= 0)
489         *t++ = a / *u++;
490     return m;
491 }
```

6.229.4.8 HxMatrix operator+ (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Addition.

```
472 {
473     if ((a.nCol() != b.nCol()) || (a.nRow() != b.nRow()) ) {
474         error("nonconformant HxMatrix + HxMatrix operands.");
475         return HxMatrix(0,0);
476     }
477     HxMatrix m(a.nRow(), b.nCol());
```

```
478     int i, j;
479     for (i=0 ; i<a.nRow() ; i++) {
480         for (j=0 ; j<a.nCol() ; j++)
481             m[i][j] = a[i][j] + b[i][j];
482     }
483     return m;
484 }
```

6.229.4.9 HxMatrix operator+ (const HxMatrix & *a*, double *b*) [friend]

Addition.

```
495 {
496     HxMatrix m(a);
497     double* t = m._data;
498     double* u = a._data;
499     int i = a.nElem();
500     while (--i >= 0)
501         *t++ = *u++ + b;
502     return m;
503 }
```

6.229.4.10 HxMatrix operator+ (double *a*, const HxMatrix & *b*) [friend]

Addition.

```
507 {
508     HxMatrix m(b);
509     double* t = m._data;
510     double* u = b._data;
511     int i = b.nElem();
512     while (--i >= 0)
513         *t++ = a + *u++;
514     return m;
515 }
```

6.229.4.11 HxMatrix operator- (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Subtraction.

```
488 {
489     if ((a.nCol() != b.nCol()) || (a.nRow() != b.nRow()) ) {
490         error("nonconformant HxMatrix - HxMatrix operands.");
491         return HxMatrix(0,0);
492     }
493     HxMatrix m(a.nRow(), b.nCol());
494     int i, j;
495     for (i=0 ; i<a.nRow() ; i++) {
496         for (j=0 ; j<a.nCol() ; j++)
497             m[i][j] = a[i][j] - b[i][j];
498     }
499     return m;
500 }
```

6.229.4.12 HxMatrix operator- (const HxMatrix & *a*, double *b*) [friend]

Subtraction.

```
519 {
520     HxMatrix m(a);
521     double* t = m._data;
522     double* u = a._data;
523     int i = a.nElem();
524     while (--i >= 0)
525         *t++ = *u++ - b;
526     return m;
527 }
```

6.229.4.13 HxMatrix operator- (double *a*, const HxMatrix & *b*) [friend]

Subtraction.

```
531 {
532     HxMatrix m(b);
533     double* t = m._data;
534     double* u = b._data;
535     int i = b.nElem();
536     while (--i >= 0)
537         *t++ = a - *u++;
538     return m;
539 }
```

6.229.4.14 int operator== (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Equal.

```
457 {
458     if ((a.nCol() != b.nCol()) || (a.nRow() != b.nRow()))
459         return 0;
460     double *t = a._data;
461     double *u = b._data;
462     int i = a.nElem();
463     while (--i >= 0) {
464         if (fabs(*t++ - *u++) > HxMatrix_EPS)
465             return 0;
466     }
467     return 1;
468 }
```

6.229.4.15 int operator!= (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Not equal.

```
543 {
544     return !(a == b);
545 }
```

6.229.4.16 HxVec3Double operator ∗ (const HxVec3Double & *a*, const HxMatrix & *b*) [friend]

Multiplication.

Matrix must have matching dimensions.

```
563 {
564     if ((b.nRow() != 3) || (b.nCol() != 3)) {
565         error("nonconformant HxVec3Double * HxMatrix operands.");
566         return HxVec3Double();
567     }
568     double v[3];
569     for (int i=0 ; i<b.nCol() ; i++) {
570         v[i] = a.x()*b[i][0] + a.y()*b[i][1] + a.z()*b[i][2];
571     }
572     return HxVec3Double(v[0], v[1], v[2]);
573 }
```

6.229.4.17 HxVec3Double operator ∗ (const HxMatrix & *a*, const HxVec3Double & *b*) [friend]

Multiplication.

Matrix must have matching dimensions.

```
577 {
578     if ((a.nRow() != 3) || (a.nCol() != 3)) {
579         error("nonconformant HxMatrix * HxVec3Double operands.");
580         return HxVec3Double();
581     }
582     double v[3];
583     for (int i=0 ; i<a.nRow() ; i++) {
584         v[i] = a[i][0]*b.x() + a[i][1]*b.y() + a[i][2]*b.z();
585     }
586     return HxVec3Double(v[0], v[1], v[2]);
587 }
```

The documentation for this class was generated from the following files:

- **HxMatrix.h**
- HxMatrix.c

6.230 HxMfBpo Class Reference

Class definition of method frame for binary pixel operations.

`#include <HxMfBpo.h>`

Public Methods

- **HxMfBpo** (**HxImageData** ∗src1, **HxImageData** ∗src2, **HxString** bpoName)

  Constructor.

- ∼**HxMfBpo** ()

  Destructor.

- **HxImageData** ∗ **source1** () const

  The first argument image of the frame.

- **HxImageData** ∗ **source2** () const

  The second argument image of the frame.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

- bool **preOpIsOk** () const

  Indicates whether initialization was OK.

6.230.1 Detailed Description

Class definition of method frame for binary pixel operations.

6.230.2 Constructor & Destructor Documentation

6.230.2.1 HxMfBpo::HxMfBpo (HxImageData ∗ *src1*, HxImageData ∗ *src2*, HxString *bpoName*)

Constructor.

```
18     : _src1(src1), _src2(src2), _tmpSrc1(0), _tmpSrc2(0), _preOpIsOk(true)
19 {
20     if (!src1 || !src2)
21     {
22         _preOpIsOk = false;
23         return;
24     }
25 
26     HxImageSignature src1Sig(src1->signature());
27     HxImageSignature src2Sig(src2->signature());
28     HxImageSignature broadestSig(src1Sig.broadest(src2Sig));
29     HxImageSignature resultSig;
30 
31     HxImgFtorRuleBase::QueryResultType qRes;
32 
33     qRes = HxImgFtorRuleBase::instance().getResultType(
34                 src1Sig, "bpo",
35                 src1Sig.toString(), src2Sig.toString(), bpoName);
36 
37     if (qRes)
38     {
39         /*
40          *  Only very specific combinations of argument types allowed.
41          *  Argument types stay what they are.
42          */
43         resultSig = qRes;
44     }
45     else
46     {
47         // Type of result and second argument depends on type of first
48         // argument.
49 
```

```
50          qRes = HxImgFtorRuleBase::instance().getArgumentType(
51                      src1Sig, "bpo", src1Sig.toString(), bpoName);
52
53          src1Sig = (qRes && src2Sig.isEqual(qRes)) ? src1Sig : broadestSig;
54
55          // If type of second argument does not fit the required type
56          // another attempt is made with the broadest signature.
57          /*  24/01/2001 IMPORTANT DESIGN NOTE:
58           *  One can argue that if a second argument type can be found
59           *  when querying with the original first argument type as key,
60           *  and the given second argument type does not match the query result,
61           *  the second argument should be converted to the query result, but
62           *  only when the query result is broader than the second argument
63           *  type.
64           */
65          // src1Sig = (qRes && src2Sig.broadest(qRes).isEqual(qRes))
66          //              ? src1Sig : broadestSig;
67
68          qRes = HxImgFtorRuleBase::instance().getArgumentType(
69                      src2Sig, "bpo", src1Sig.toString(), bpoName);
70
71          if (!qRes)
72          {
73              HxEnvironment::instance()->errorStream()
74                  << "Cannot apply binary pixel operation " << bpoName << " to "
75                  << "images with type " << src1Sig.toString() << " and "
76                  << src2Sig.toString() << STD_ENDL;
77              HxEnvironment::instance()->flush();
78              _preOpIsOk = false;
79          }
80          else
81          {
82              src2Sig = qRes;
83          }
84
85          resultSig = HxImgFtorRuleBase::instance().getResultType(
86                      src1Sig, "bpo", src1Sig.toString(), bpoName);
87      }
88
89      HxSizes sizes = src1->sizes().sup(src2->sizes());
90
91      if (!src1Sig.isEqual(src1->signature())) {
92          _tmpSrc1 = HxImgDataFactory::instance().makeImage(src1Sig, sizes);
93          _tmpSrc1->set(src1);
94          _src1 = _tmpSrc1;
95      }
96
97      if (!src2Sig.isEqual(src2->signature())) {
98          _tmpSrc2 = HxImgDataFactory::instance().makeImage(src2Sig, sizes);
99          _tmpSrc2->set(src2);
100         _src2 = _tmpSrc2;
101     }
102
103     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
104
105 }
```

6.230.2.2 HxMfBpo::~HxMfBpo ()

Destructor.

```
108 {
```

```
109     if (_tmpSrc1)
110         delete _tmpSrc1;
111     if (_tmpSrc2)
112         delete _tmpSrc2;
113 }
```

6.230.3 Member Function Documentation

6.230.3.1 HxImageData * HxMfBpo::source1 () const

The first argument image of the frame.

```
117 {
118     return _src1;
119 }
```

6.230.3.2 HxImageData * HxMfBpo::source2 () const

The second argument image of the frame.

```
123 {
124     return _src2;
125 }
```

6.230.3.3 HxImageData * HxMfBpo::result () const

The result image of the frame.

```
129 {
130     return _result;
131 }
```

6.230.3.4 bool HxMfBpo::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
54 {
55     return _preOpIsOk;
56 }
```

The documentation for this class was generated from the following files:

- **HxMfBpo.h**
- HxMfBpo.c

6.231 HxMfBroadestSig Class Reference

Class definition of method frame for broadest signature.

`#include <HxMfBroadestSig.h>`

Public Methods

- **HxMfBroadestSig** (**HxImageData** *objImg, **HxImageData** *argImg, int pixDim=0, int makeResult=1)

  Constructor.

- **~HxMfBroadestSig** ()

  Destructor.

- **HxImageData** * **object** () const

  The object image of the frame.

- **HxImageData** * **argument** () const

  The argument image of the frame.

- **HxImageData** * **result** () const

  The result image of the frame.

6.231.1 Detailed Description

Class definition of method frame for broadest signature.

This method frame is used to allow for heterogeneous image data types.

6.231.2 Constructor & Destructor Documentation

6.231.2.1 HxMfBroadestSig::HxMfBroadestSig (HxImageData * *objImg*, HxImageData * *argImg*, int *pixDim* = 0, int *makeResult* = 1)

Constructor.

Object() will point to a copy of objImg with a signature equal to the broadest of objImg and argImg and sizes equal to the maximum of objImg and argImg. However, if (pixDim != 0) it will have the specified pixel dimensionality. In case the signature of argImg is not equal to the signature of **object**() (p. 588) a temporary image with that signature will be created for **argument**() (p. 588) with the same pixel values as argImg. if (makeResult == 1) the result will actually be allocated (and used by **object**() (p. 588)). if (makeResult == 0) there is no result image and a temporary image (with the broadest signature) may have to be allocated for **object**() (p. 588).

```
23                                      :
24          _object(objImg), _argument(argImg), _tmpArg(0), _tmpObj(0)
25 {
26     if (!_object || !_argument)
27         return;
28
29     HxImageSignature objSig(_object->signature());
30     HxImageSignature argSig(_argument->signature());
31     HxImageSignature resultSig(objSig.broadest(argSig));
32     if (pixDim != 0)
33         resultSig.setPixelDimensionality(pixDim);
34     HxSizes sizes = std::max(_object->sizes(), _argument->sizes());
35
36     if (!argSig.isEqual(resultSig)) {
```

```
37         _tmpArg = HxImgDataFactory::instance().makeImage(resultSig, sizes);
38         _tmpArg->set(_argument);
39         _argument = _tmpArg;
40     }
41
42     if (makeResult) {
43         _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
44         if (_result)
45             _result->set(_object);
46         _object = _result;
47     } else { // since we did not make the result (and the new object) we
48               // may have to "adjust" the object sig to match the broadest
49         if (!objSig.isEqual(resultSig)) {
50             _tmpObj = HxImgDataFactory::instance().makeImage(resultSig, sizes);
51             _tmpObj->set(_object);
52             _object = _tmpObj;
53         }
54     }
55 }
```

6.231.2.2 HxMfBroadestSig::~HxMfBroadestSig ()

Destructor.

```
58 {
59     if (_tmpArg)
60         delete _tmpArg;
61     if (_tmpObj)
62         delete _tmpObj;
63 }
```

6.231.3 Member Function Documentation

6.231.3.1 HxImageData * HxMfBroadestSig::object () const

The object image of the frame.

```
67 {
68     return _object;
69 }
```

6.231.3.2 HxImageData * HxMfBroadestSig::argument () const

The argument image of the frame.

```
73 {
74     return _argument;
75 }
```

6.231.3.3 HxImageData * HxMfBroadestSig::result () const

The result image of the frame.

```
79 {
80     return _result;
81 }
```

The documentation for this class was generated from the following files:

- **HxMfBroadestSig.h**
- HxMfBroadestSig.c

6.232 HxMfGenConv Class Reference

Class definition of method frame for generalized convolution operations.

`#include <HxMfGenConv.h>`

Public Methods

- **HxMfGenConv** (**HxImageData** ∗source, **HxImageData** ∗kernel, **HxImageRep::ResultPrecision** resPrec, bool is1dConv=false, **HxImageRep::ResultPrecision** interPrec=HxImageRep::DEFAULT_PREC)

  Constructor.

- **HxMfGenConv** (**HxImageData** ∗source, **HxImageData** ∗kernel, **HxImageData** ∗kernel2, **HxImageRep::ResultPrecision** resPrec, bool is1dConv=false, **HxImageRep::ResultPrecision** interPrec=HxImageRep::DEFAULT_PREC)

  Constructor.

- ∼**HxMfGenConv** ()

  Destructor.

- **HxImageData** ∗ **source** () const

  The source image of the frame.

- **HxImageData** ∗ **kernel** () const

  The kernel image of the frame.

- **HxImageData** ∗ **kernel2** () const

  The second kernel image of the frame.

- **HxImageData** ∗ **object** () const

  The image to perform the convolution on.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

- void **setObjectAsSource** ()

  Use the object as source.

- bool **preOpIsOk** () const

  Indicates whether initialization was OK.

6.232.1 Detailed Description

Class definition of method frame for generalized convolution operations.

The method frame tries to satisfy the general rules set out for generalized convolutions. If this is not possible, preOpOk() will return false.

6.232.2 Constructor & Destructor Documentation

6.232.2.1 HxMfGenConv::HxMfGenConv (HxImageData * *source*, HxImageData * *kernel*, HxImageRep::ResultPrecision *resPrec*, bool *is1dConv* = false, HxImageRep::ResultPrecision *interPrec* = HxImageRep::DEFAULT_PREC)

Constructor.

The kernel should have the same dimensionality as the image, otherwise preOpOk() will return false. If necessary the type of the kernel will be converted according to the following rules:

1) The kernel type signature must be equal or broader than the source type signature. 2a) If the kernel has an integral pixel type its pixel precision will be that of int. 2b) If the kernel has a real pixel type its pixel precision will be that of double.

The type of the result image will be set according to the specified precision:

SOURCE_PREC The result type is the same as the source type. ARITH_PREC The result type is the same as the type of the kernel after the kernel has been converted. SMALL_PREC The result type will be the same as above but with a smaller pixel precision. If the kernel has an integral pixel type the result pixel precision is that of short. If the kernel has a real pixel type the result pixel precision is that of float.

If the intermediate precision is set to ARITH_PREC a temporary object image with the same type as the kernel image will be allocated. Whenever the method frame is queried for the reslt image the object image will be copied to the result image.

Note that the actual convolution will be performed in the precision of the kernel type after conversion.

```
22     :   _preOpIsOk(true), _resultIsCopied(false), _source(source),
23         _kernel(kernel), _tmpKernel(0), _kernel2(0), _tmpKernel2(0)
24 {
25     initMethodFrame(resPrec, interPrec, is1dConv);
26 }
```

6.232.2.2 HxMfGenConv::HxMfGenConv (HxImageData * *source*, HxImageData * *kernel*, HxImageData * *kernel2*, HxImageRep::ResultPrecision *resPrec*, bool *is1dConv* = false, HxImageRep::ResultPrecision *interPrec* = HxImageRep::DEFAULT_PREC)

Constructor.

```
32     :   _preOpIsOk(true), _resultIsCopied(false), _source(source),
33         _kernel(kernel), _tmpKernel(0), _kernel2(kernel2), _tmpKernel2(0)
34 {
35     initMethodFrame(resPrec, interPrec, is1dConv);
36 }
```

6.232.2.3 HxMfGenConv::~HxMfGenConv ()

Destructor.

```
39 {
40     if (_tmpKernel)
41         delete _tmpKernel;
42     if (_tmpKernel2)
43         delete _tmpKernel2;
44     if (_object != _result)
45         delete _object;
46 }
```

6.232.3 Member Function Documentation

6.232.3.1 HxImageData * HxMfGenConv::source () const

The source image of the frame.

```
50 {
51     return _source;
52 }
```

6.232.3.2 HxImageData * HxMfGenConv::kernel () const

The kernel image of the frame.

```
63 {
64     return _kernel;
65 }
```

6.232.3.3 HxImageData * HxMfGenConv::kernel2 () const

The second kernel image of the frame.

```
69 {
70     return _kernel2;
71 }
```

6.232.3.4 HxImageData * HxMfGenConv::object () const

The image to perform the convolution on.

```
56 {
57     ((HxMfGenConv*)this)->_resultIsCopied = false;
58     return _object;
59 }
```

6.232.3.5 HxImageData * HxMfGenConv::result () const

The result image of the frame.

```
75 {
76     if ((_result != _object) && !_resultIsCopied)
77     {
78         _result->set(_object);
79         ((HxMfGenConv*)this)->_resultIsCopied = true;
80     }
81     return _result;
82 }
```

6.232.3.6 void HxMfGenConv::setObjectAsSource ()

Use the object as source.

```
86 {
87     _source = _object;
88 }
```

6.232.3.7 bool HxMfGenConv::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
126 {
127     return _preOpIsOk;
128 }
```

The documentation for this class was generated from the following files:

- **HxMfGenConv.h**
- HxMfGenConv.c

6.233 HxMfIdentity Class Reference

Class definition of method frame for identity.

`#include <HxMfIdentity.h>`

Public Methods

- **HxMfIdentity** (**HxImageData** *objImg, int pixDim=0)

  Constructor.

- **~HxMfIdentity** ()

  Destructor.

- **HxImageData** * **object** () const

  The object image of the frame.

- **HxImageData** * **result** () const

  The result image of the frame.

6.233.1 Detailed Description

Class definition of method frame for identity.

The method frame just copies the image data to ensure the value paradigm.

6.233.2 Constructor & Destructor Documentation

6.233.2.1 HxMfIdentity::HxMfIdentity (HxImageData * *objImg*, int *pixDim* = 0)

Constructor.

Object() and **result**() (p. 593) will point to a copy of objImg (with the same signature, sizes and pixel values) to take care of the value paradigm. However, if (pixDim != 0) it will have the specified pixel dimensionality.

```
16                                                          : _object(objImg)
17 {
18     if (!_object)
19         return;
20
21     HxImageSignature objSig(_object->signature());
22     if (pixDim != 0)
23         objSig.setPixelDimensionality(pixDim);
24     HxSizes sizes = _object->sizes();
25
26     _result = HxImgDataFactory::instance().makeImage(objSig, sizes);
27     if (_result)
28         _result->set(_object);
29     _object = _result;
30 }
```

6.233.2.2 HxMfIdentity::~HxMfIdentity ()

Destructor.

```
33 {
34 }
```

6.233.3 Member Function Documentation

6.233.3.1 HxImageData * HxMfIdentity::object () const

The object image of the frame.

```
38 {
39     return _object;
40 }
```

6.233.3.2 HxImageData * HxMfIdentity::result () const

The result image of the frame.

```
44 {
45     return _result;
46 }
```

The documentation for this class was generated from the following files:

- **HxMfIdentity.h**
- HxMfIdentity.c

6.234 HxMfKernelNgb Class Reference

Class definition of a method frame for neighbourhood operations using a kernel.

`#include <HxMfKernelNgb.h>`

Public Methods

- **HxMfKernelNgb** (**HxImageData** ∗source, **HxImageData** ∗kernel, **HxString** ngbName, **HxTagList** &tags)
  Constructor.
- ∼**HxMfKernelNgb** ()
  Destructor.
- **HxImageData** ∗ **source** () const
  The source image of the frame.
- **HxImageData** ∗ **kernel** () const
  The kernel image of the frame.
- **HxImageData** ∗ **result** () const
  The result image of the frame.
- bool **preOpIsOk** () const
  Indicates whether initialization was OK.

6.234.1 Detailed Description

Class definition of a method frame for neighbourhood operations using a kernel.

6.234.2 Constructor & Destructor Documentation

6.234.2.1 HxMfKernelNgb::HxMfKernelNgb (HxImageData ∗ *srcImg*, HxImageData ∗ *kernel*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

A result image will be allocated with the same size as the source image. The registry will be queried for the result type. The registry will also be queried for the kernel type. If necessary the kernel image will be converted to this type.

```
21     : _source(srcImg), _kernel(kernel), _result(0),
22         _tmpKernel(0), _preOpIsOk(true)
23 {
24     if (!_source || !_kernel)
25     {
26         _preOpIsOk = false;
27         return;
28     }
29
30     HxImageSignature srcSig(_source->signature());
31
32     HxImageSignature resultSig
33         = HxImgFtorRuleBase::instance().getResultType(
34             srcSig, "kernelNgb", srcSig.toString(), ngbName);
35
36     HxImageSignature kernelSig
37         = HxImgFtorRuleBase::instance().getKernelType(
38             _kernel->signature(), "kernelNgb", srcSig.toString(), ngbName);
39
40     _result = HxImgDataFactory::instance().makeImage(resultSig, srcImg->sizes());
41
42     if (kernelSig != _kernel->signature())
43     {
44         _tmpKernel = HxImgDataFactory::instance().makeImage(
45             kernelSig, _kernel->sizes());
46         _tmpKernel->setPartImage(_kernel);
47         _tmpKernel->weight(_kernel->weight().x());
48         _kernel = _tmpKernel;
49     }
50 }
```

6.234.2.2 HxMfKernelNgb::~HxMfKernelNgb ()

Destructor.

```
53 {
54     if (_tmpKernel)
55         delete _tmpKernel;
56 }
```

6.234.3 Member Function Documentation

6.234.3.1 HxImageData * HxMfKernelNgb::source () const

The source image of the frame.

```
60 {
61     return _source;
62 }
```

6.234.3.2 HxImageData * HxMfKernelNgb::kernel () const

The kernel image of the frame.

```
66 {
67     return _kernel;
68 }
```

6.234.3.3 HxImageData * HxMfKernelNgb::result () const

The result image of the frame.

```
72 {
73     return _result;
74 }
```

6.234.3.4 bool HxMfKernelNgb::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
60 {
61     return _preOpIsOk;
62 }
```

The documentation for this class was generated from the following files:

- **HxMfKernelNgb.h**
- HxMfKernelNgb.c

6.235 HxMfMNpo Class Reference

Class definition of method frame for unary pixel operations.

`#include <HxMfMNpo.h>`

Public Methods

- **HxMfMNpo** (**HxImageData** **srcs, int srcCnt, **HxString** mpoName)
  Constructor.
- ~**HxMfMNpo** ()
  Destructor.
- **HxImageData** ** **results** () const
  The result images of the frame.
- **HxImageData** * **results** (int n) const
- int **resultCnt** () const
  The number of source images of the frame.
- **HxImageData** ** **sources** () const
  The source images of the frame.
- int **sourceCnt** () const
  The number of source images of the frame.
- bool **preOpIsOk** () const
  Indicates whether initialization was OK.

6.235.1 Detailed Description

Class definition of method frame for unary pixel operations.

6.235.2 Constructor & Destructor Documentation

6.235.2.1 HxMfMNpo::HxMfMNpo (HxImageData ** *srcs*, int *srcCnt*, HxString *mpoName*)

Constructor.

```
19     : _preOpIsOk(true)
20 {
21     if (srcCnt <= 0) {
22         _preOpIsOk = false;
23         return;
24     }
25 
26     _srcCnt = srcCnt;
27     _src = new HxImageData * [srcCnt];
28     _tmp = new HxImageData * [srcCnt];
29 
30     int i;
31     for (i = 0; i < srcCnt; i++) {
32         if (!srcs[i]) {
33             _preOpIsOk = false;
34             return;
35         }
36         _src[i] = srcs[i];
37         _tmp[i] = 0;
38     }
39 
40     /* find broadest signature and minimum sizes */
41     HxImageSignature broadestSig(srcs[0]->signature());
42     HxSizes sizes = srcs[0]->sizes();
43     for (i = 1; i < srcCnt; i++) {
44         HxImageSignature srcnSig(srcs[i]->signature());
45         broadestSig = broadestSig.broadest(srcnSig);
46         sizes = sizes.sup(srcs[i]->sizes());
47     }
48 
49     static HxRegKey* mpoKey
50         = HxRegistry::instance().findKey("/imagefunctortable/mpo");
51     HxRegKey* k = mpoKey ? mpoKey->findKey(mpoName) : 0;
52     k = k ? k->findKey("resulttype") : 0;
53     const HxRegValue* v = k ? k->findValue(broadestSig.toString()) : 0;
54     HxImageSignature resultSig
55         = v ? HxImageSignature::NameToSignature(v->getData().toString()) :
56             broadestSig;
57 
58     for (i = 0; i < srcCnt; i++) {
59         if (!broadestSig.isEqual(srcs[i]->signature())) {
60             _tmp[i] = HxImgDataFactory::instance().makeImage(broadestSig, sizes);
61             _tmp[i]->set(srcs[i]);
62             _src[i] = _tmp[i];
63         }
64     }
65 
67     HxTagList tags;
68     HxAddTag(tags, "sourceCnt", _srcCnt);
69     if (!HxImageData::probeMNpo(resultSig, broadestSig, mpoName, tags)) {
70         _preOpIsOk = false;
71         return;
```

```
72     }
73     _resCnt = HxGetTag(tags, "resultCnt", 1);
74
75     _result = new HxImageData * [_resCnt];
76     for (i = 0; i < _resCnt; i++)
77         _result[i] = HxImgDataFactory::instance().makeImage(resultSig, sizes);
78 }
```

6.235.2.2 HxMfMNpo::~HxMfMNpo ()

Destructor.

```
81 {
82     for (int i = 0; i < _srcCnt; i++) {
83         if (_tmp[i])
84             delete _tmp[i];
85     }
86     delete [] _result;
87     delete [] _tmp;
88     delete [] _src;
89 }
```

6.235.3 Member Function Documentation

6.235.3.1 HxImageData ** HxMfMNpo::results () const

The result images of the frame.

```
105 {
106     return _result;
107 }
```

6.235.3.2 int HxMfMNpo::resultCnt () const

The number of source images of the frame.

```
111 {
112     return _resCnt;
113 }
```

6.235.3.3 HxImageData ** HxMfMNpo::sources () const

The source images of the frame.

```
93 {
94     return _src;
95 }
```

6.235.3.4 int HxMfMNpo::sourceCnt () const

The number of source images of the frame.

```
99 {
100     return _srcCnt;
101 }
```

6.235.3.5 bool HxMfMNpo::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
57 {
58     return _preOpIsOk;
59 }
```

The documentation for this class was generated from the following files:

- **HxMfMNpo.h**
- HxMfMNpo.c

6.236 HxMfMpo Class Reference

Class definition of method frame for unary pixel operations.

`#include <HxMfMpo.h>`

Public Methods

- **HxMfMpo** (**HxImageData** **srcs, int nSrcs, **HxString** mpoName)
  Constructor.
- **~HxMfMpo** ()
  Destructor.
- **HxImageData** ** **sources** () const
  The source images of the frame.
- int **nSources** () const
  The number of source images of the frame.
- **HxImageData** * **result** () const
  The result image of the frame.

6.236.1 Detailed Description

Class definition of method frame for unary pixel operations.

6.236.2 Constructor & Destructor Documentation

6.236.2.1 HxMfMpo::HxMfMpo (HxImageData ** *srcs*, int *nSrcs*, HxString *mpoName*)

Constructor.

```
19 {
20     if (!nSrcs)
21         return;
22 
23     _nSrcs = nSrcs;
24     _src = new HxImageData * [nSrcs];
25     _tmp = new HxImageData * [nSrcs];
26 
27     int i;
28     for (i = 0; i < nSrcs; i++) {
29         if (!srcs[i])
30             return;
31         _src[i] = srcs[i];
32         _tmp[i] = 0;
33     }
34 
35     /* find broadest signature and minimum sizes */
36     HxImageSignature broadestSig(srcs[0]->signature());
37     HxSizes sizes = srcs[0]->sizes();
38     for (i = 1; i < nSrcs; i++) {
39         HxImageSignature srcnSig(srcs[i]->signature());
40         broadestSig = broadestSig.broadest(srcnSig);
41         sizes = sizes.sup(srcs[i]->sizes());
42     }
43 
44     static HxRegKey* mpoKey
45         = HxRegistry::instance().findKey("/imagefunctortable/mpo");
46     HxRegKey* k = mpoKey ? mpoKey->findKey(mpoName) : 0;
47     k = k ? k->findKey("resulttype") : 0;
48     const HxRegValue* v = k ? k->findValue(broadestSig.toString()) : 0;
49     HxImageSignature resultSig
50         = v ? HxImageSignature::NameToSignature(v->getData().toString()) :
51             broadestSig;
52 
53     for (i = 0; i < nSrcs; i++) {
54         if (!broadestSig.isEqual(srcs[i]->signature())) {
55             _tmp[i] = HxImgDataFactory::instance().makeImage(broadestSig, sizes);
56             _tmp[i]->set(srcs[i]);
57             _src[i] = _tmp[i];
58         }
59     }
60 
61     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
62 }
```

6.236.2.2 HxMfMpo::~HxMfMpo ()

Destructor.

```
65 {
66     for (int i = 0; i < _nSrcs; i++) {
67         if (_tmp[i])
68             delete _tmp[i];
69     }
```

```
70     delete [] _tmp;
71     delete [] _src;
72 }
```

6.236.3 Member Function Documentation

6.236.3.1 HxImageData ** HxMfMpo::sources () const

The source images of the frame.

```
76 {
77     return _src;
78 }
```

6.236.3.2 int HxMfMpo::nSources () const

The number of source images of the frame.

```
82 {
83     return _nSrcs;
84 }
```

6.236.3.3 HxImageData * HxMfMpo::result () const

The result image of the frame.

```
88 {
89     return _result;
90 }
```

The documentation for this class was generated from the following files:

- **HxMfMpo.h**
- HxMfMpo.c

6.237 HxMfNgb Class Reference

Class definition of method frame for neighbourhood operations.

`#include <HxMfNgb.h>`

Public Methods

- **HxMfNgb** (**HxImageData** *srcImg, **HxString** ngbName, **HxTagList** &tags)

  Constructor.

- ~**HxMfNgb** ()

  Destructor.

- **HxImageData** ∗ **source** () const

  The argument image of the frame.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

- bool **preOpIsOk** () const

  Indicates whether initialization was OK.

6.237.1 Detailed Description

Class definition of method frame for neighbourhood operations.

6.237.2 Constructor & Destructor Documentation

6.237.2.1 HxMfNgb::HxMfNgb (HxImageData ∗ *srcImg*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

A result image will be allocated with the same size as the source image. The image functor rule base will be queried for the result type.

```
20     : _source(srcImg), _result(0), _preOpIsOk(true)
21 {
22     if (!_source)
23     {
24         _preOpIsOk = false;
25         return;
26     }
27 
28     HxImageSignature srcSig(_source->signature());
29 
30     HxImageSignature resultSig
31         = HxImgFtorRuleBase::instance().getResultType(
32                 srcSig, "ngb", srcSig.toString(), ngbName);
33 
34     HxSizes sizes = _source->sizes();
35 
36     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
37 }
```

6.237.2.2 HxMfNgb::∼HxMfNgb ()

Destructor.

```
40 {
41 }
```

6.237.3 Member Function Documentation

6.237.3.1 HxImageData * HxMfNgb::source () const

The argument image of the frame.

```
45 {
46     return _source;
47 }
```

6.237.3.2 HxImageData * HxMfNgb::result () const

The result image of the frame.

```
51 {
52     return _result;
53 }
```

6.237.3.3 bool HxMfNgb::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
50 {
51     return _preOpIsOk;
52 }
```

The documentation for this class was generated from the following files:

- **HxMfNgb.h**
- HxMfNgb.c

6.238 HxMfReqIntermediatePixType Class Reference

Class definition of method frame for required intermediate pixel type.

`#include <HxMfReqIntermediatePixType.h>`

Public Methods

- **HxMfReqIntermediatePixType** (int pixelDimensionality, **HxValueType** pixelType, int pixelPrecision, **HxImageData** *obj, **HxImageData** *ker, **HxImageData** *ker2=0)

  Constructor.

- **~HxMfReqIntermediatePixType** ()

  Destructor.

- **HxImageData** * **object** () const

  The object image of the frame.

- **HxImageData** ∗ **kernel** () const

  The (first) kernel image of the frame.

- **HxImageData** ∗ **kernel2** () const

  The second kernel image of the frame.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

6.238.1 Detailed Description

Class definition of method frame for required intermediate pixel type.

The method frame is used when a function consists of several steps, and the intermediate results require a specific pixel type other than the pixel type of the operand image.

6.238.2 Constructor & Destructor Documentation

6.238.2.1 HxMfReqIntermediatePixType::HxMfReqIntermediatePixType (int *pixelDimensionality*, HxValueType *pixelType*, int *pixelPrecision*, HxImageData ∗ *obj*, HxImageData ∗ *ker*, HxImageData ∗ *ker2* = 0)

Constructor.

The required signature for the intermediate image has the dimensionality of ”obj”, and the specified ”pixelDimensionality”, ”pixelType”,and ”pixelPrecision”. Object will point to an image with this signature that is set to the pixel values of ”obj”. If the signature of ”obj” is the same as the required signature **result**() (p. 606) will point to the same as **object**() (p. 605). Otherwise, an empty image will be created and the destructor will set the pixel values of result to the pixel values of **object**() (p. 605). Both ”ker” and ”ker2” are required to have a signature with the image dimensionality and pixel dimensionality of the intermediate type, and max of the pixel types of the intermediate image and the kernel, and the maximum pixel precision. If either ”ker” or ”ker2” have a different signature a temporary kernel will be created and set to their pixel values.

```
19                                          : _object(obj), _kernel(ker),
20     _kernel2(ker2), _result(0), _tmp(0), _tmp2(0)
21 {
22     if (!_object || !_kernel)
23         return;
24
25     // Make new object
26     HxSizes sizes = obj->sizes();
27     HxImageSignature objSig(obj->signature());
28     HxImageSignature intermediateSig(objSig.imageDimensionality(),
29                         pixelDimensionality, pixelType, pixelPrecision);
30     _object = HxImgDataFactory::instance().makeImage(intermediateSig, sizes);
31     if (!_object)
32         return;
33     _object->set(obj);
34
35     // set up result
36     if (intermediateSig.isEqual(objSig)) {
37         _result = _object;
38     } else {
39         _result = HxImgDataFactory::instance().makeImage(objSig, sizes);
40     }
```

```
41
42      // adjust kernel if necessary
43      HxImageData* ptr = setArg(_kernel, intermediateSig);
44      if (ptr != 0) {
45          _tmp = ptr;
46          _kernel = ptr;
47      }
48
49      // adjust kernel2 if present and necessary
50      if (!_kernel2)
51          return;
52      HxImageData* ptr2 = setArg(_kernel2, intermediateSig);
53      if (ptr2 != 0) {
54          _tmp2 = ptr2;
55          _kernel2 = ptr2;
56      }
57  }
```

6.238.2.2 HxMfReqIntermediatePixType::~HxMfReqIntermediatePixType ()

Destructor.

```
60  {
61      if (_result != _object) {
62          _result->set(_object);
63          delete _object;
64      }
65      if (_tmp)
66          delete _tmp;
67      if (_tmp2)
68          delete _tmp2;
69  }
```

6.238.3 Member Function Documentation

6.238.3.1 HxImageData * HxMfReqIntermediatePixType::object () const

The object image of the frame.

```
73  {
74      return _object;
75  }
```

6.238.3.2 HxImageData * HxMfReqIntermediatePixType::kernel () const

The (first) kernel image of the frame.

```
79  {
80      return _kernel;
81  }
```

6.238.3.3 HxImageData * HxMfReqIntermediatePixType::kernel2 () const

The second kernel image of the frame.

```
85 {
86     return _kernel2;
87 }
```

6.238.3.4 HxImageData * HxMfReqIntermediatePixType::result () const

The result image of the frame.

```
91 {
92     return _result;
93 }
```

The documentation for this class was generated from the following files:

- **HxMfReqIntermediatePixType.h**
- HxMfReqIntermediatePixType.c

6.239 HxMfReqKernelPixType Class Reference

Class definition of method frame for required kernel pixel type.

`#include <HxMfReqKernelPixType.h>`

Public Methods

- **HxMfReqKernelPixType** (**HxImageData** *obj, **HxImageData** *ker, int pixDim=0)
  Constructor.
- **~HxMfReqKernelPixType** ()
  Destructor.
- **HxImageData * object** () const
  The object image of the frame.
- **HxImageData * kernel** () const
  The kernel image of the frame.
- **HxImageData * result** () const
  The result image of the frame.

6.239.1 Detailed Description

Class definition of method frame for required kernel pixel type.

The method frame is used to ensure that the kernel has an appropriate signature.

6.239.2 Constructor & Destructor Documentation

6.239.2.1 HxMfReqKernelPixType::HxMfReqKernelPixType (HxImageData * *obj*, HxImageData * *ker*, int *pixDim* = 0)

Constructor.

Object() and **result**() (p. 608) will point to a copy of obj with the same signature, sizes, and pixel values. However, if (pixDim != 0) it will have the specified pixel dimensionality. The required signature for the kernel has the image dimensionality and pixel dimensionality of "obj", the max of the pixel types of "obj" and "ker", and the maximum pixel precision. If this signature is not equal to the signature of ker a temporary kernel will be created for **kernel**() (p. 608) and set to the value of "ker".

```
17                     : _object(obj), _kernel(ker), _tmp(0)
18 {
19     if (!_object || !_kernel)
20         return;
21
22     HxImageSignature objSig(_object->signature());
23     if (pixDim != 0)
24         objSig.setPixelDimensionality(pixDim);
25
26     // Adjust kernel if necessary
27     HxImageData* ptr = setArg(_kernel, objSig);
28     if (ptr != 0) {
29         _tmp = ptr;
30         _kernel = ptr;
31     }
32
33     // HxMfIdentity for object
34     _result = HxImgDataFactory::instance().makeImage(objSig, obj->sizes());
35     if (_result)
36         _result->set(_object);
37     _object = _result;
38 }
```

6.239.2.2 HxMfReqKernelPixType::~HxMfReqKernelPixType ()

Destructor.

```
41 {
42     if (_tmp)
43         delete _tmp;
44 }
```

6.239.3 Member Function Documentation

6.239.3.1 HxImageData * HxMfReqKernelPixType::object () const

The object image of the frame.

```
48 {
49     return _object;
50 }
```

6.239.3.2 HxImageData * HxMfReqKernelPixType::kernel () const

The kernel image of the frame.

```
54 {
55     return _kernel;
56 }
```

6.239.3.3 HxImageData * HxMfReqKernelPixType::result () const

The result image of the frame.

```
60 {
61     return _result;
62 }
```

The documentation for this class was generated from the following files:

- **HxMfReqKernelPixType.h**
- HxMfReqKernelPixType.c

6.240 HxMfReqMaskPixType Class Reference

Class definition of method frame for required mask pixel type.

`#include <HxMfReqMaskPixType.h>`

Public Methods

- **HxMfReqMaskPixType** (**HxImageData** *mask, **HxImageData** *obj, int pixDim=0)

  Constructor.

- **HxMfReqMaskPixType** (**HxImageData** *mask, int pixDim, **HxValueType** pixType, int pixPrec)

  Constructor.

- ~**HxMfReqMaskPixType** ()

  Destructor.

- **HxImageData** * **mask** () const

  The mask image of the frame.

6.240.1 Detailed Description

Class definition of method frame for required mask pixel type.

The method frame is used to ensure that the mask has an appropriate signature.

6.240.2 Constructor & Destructor Documentation

6.240.2.1 HxMfReqMaskPixType::HxMfReqMaskPixType (HxImageData * *mask*, HxImageData * *obj*, int *pixDim* = 0)

Constructor.

if (pixDim != 0) the required signature of the mask has the specified pixel dimensionality. Otherwise, it will have the pixel dimensionality of obj. The required signature for the mask has the image dimensionality and pixel dimensionality of "obj", the max of the pixel types of "obj" and "ker", and the maximum pixel precision. If this signature is not equal to the signature of mask a temporary mask will be created for **mask**() (p. 610) and set to the value of "mask".

```
18                      : _mask(mask), _tmp(0)
19 {
20     if (!obj || !_mask)
21         return;
22
23     HxImageSignature objSig(obj->signature());
24     if (pixDim != 0)
25         objSig.setPixelDimensionality(pixDim);
26
27     // Adjust mask if necessary
28     HxImageData* ptr = setArg(_mask, objSig);
29     if (ptr != 0) {
30         _tmp = ptr;
31         _mask = ptr;
32     }
33 }
```

6.240.2.2 HxMfReqMaskPixType::HxMfReqMaskPixType (HxImageData * *mask*, int *pixDim*, HxValueType *pixType*, int *pixPrec*)

Constructor.

The required signature has the dimensionality of "mask" and the specified "pixDim", "pixType, and "pixPrec". If this signature is not equal to the signature of mask a temporary mask will be created for **mask**() (p. 610) and set to the value of "mask".

```
36                                          : _mask(mask), _tmp(0)
37 {
38     if (!_mask)
39         return;
40
41     HxImageSignature maskSig(_mask->signature());
42     HxImageSignature newMaskSig(maskSig.imageDimensionality(), pixDim, pixType,
43                                 pixPrec);
44     if (!newMaskSig.isEqual(maskSig)) {
45         _tmp = HxImgDataFactory::instance().makeImage(newMaskSig, mask->sizes());
46         _tmp->set(_mask);
47         _tmp->weight(_mask->weight().x());
48         _mask = _tmp;
49     }
50 }
```

6.240.2.3 HxMfReqMaskPixType::~HxMfReqMaskPixType ()

Destructor.

```
53 {
54     if (_tmp)
55         delete _tmp;
56 }
```

6.240.3 Member Function Documentation

6.240.3.1 HxImageData * HxMfReqMaskPixType::mask () const

The mask image of the frame.

```
60 {
61     return _mask;
62 }
```

The documentation for this class was generated from the following files:

- **HxMfReqMaskPixType.h**
- HxMfReqMaskPixType.c

6.241 HxMfReqResultPixType Class Reference

Class definition of method frame for required result pixel type.

`#include <HxMfReqResultPixType.h>`

Public Methods

- **HxMfReqResultPixType** (int pixelDimensionality, **HxValueType** pixelType, int pixelPrecision, **HxImageData** *objImg)

  Constructor.

- **~HxMfReqResultPixType** ()

  Destructor.

- **HxImageData * object** () const

  The object image of the frame.

- **HxImageData * result** () const

  The result image of the frame.

6.241.1 Detailed Description

Class definition of method frame for required result pixel type.

The method frame is used when the resulting iamge requires a specific pixel type other than the pixel type of the operand image.

6.241.2 Constructor & Destructor Documentation

6.241.2.1 HxMfReqResultPixType::HxMfReqResultPixType (int *pixelDimensionality*, HxValueType *pixelType*, int *pixelPrecision*, HxImageData * *objImg*)

Constructor.

Object() will point to a copy of objImg with the same signature, sizes, and pixel values. If the signature of objImg is the same as the signature required for the result image (specified by the given parameters) **result**() (p. 612) will point to the same as **object**() (p. 611). Otherwise, an empty image will be created and the destructor will set the pixel values of result to the pixel values of **object**() (p. 611).

```
19         : _object(0), _result(0)
20 {
21     if (!objImg)
22         return;
23 
24     HxImageSignature objectSig(objImg->signature());
25     HxSizes sizes = objImg->sizes();
26 
27     _object = HxImgDataFactory::instance().makeImage(objectSig, sizes);
28     if (!_object)
29         return;
30     _object->set(objImg);
31 
32     HxImageSignature resultSig(objectSig.imageDimensionality(),
33                         pixelDimensionality, pixelType, pixelPrecision);
34 
35     if (resultSig.isEqual(objectSig)) {
36         _result = _object;
37     } else {
38         _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
39     }
40 }
```

6.241.2.2 HxMfReqResultPixType::~HxMfReqResultPixType ()

Destructor.

```
43 {
44     /* Dirty: the pixel values of _result are set after(!) the pointer has
45      * been returned in result() so result() is only valid after destruction
46      * of the method frame.
47      */
48     if (_result != _object) {
49         _result->set(_object);
50         delete _object;
51     }
52 }
```

6.241.3 Member Function Documentation

6.241.3.1 HxImageData * HxMfReqResultPixType::object () const

The object image of the frame.

```
56 {
```

```
57     return _object;
58 }
```

6.241.3.2 HxImageData * HxMfReqResultPixType::result () const

The result image of the frame.

```
62 {
63     return _result;
64 }
```

The documentation for this class was generated from the following files:

- **HxMfReqResultPixType.h**
- HxMfReqResultPixType.c

6.242 HxMfResize Class Reference

Class definition of method frame for resizing of image.

`#include <HxMfResize.h>`

Public Methods

- **HxMfResize** (**HxImageData** *objImg, **HxSizes** newSize, **HxImageData** *argImg=0)

  Constructor.

- **~HxMfResize** ()

  Destructor.

- **HxImageData** * **object** () const

  The object image of the frame.

- **HxImageData** * **argument** () const

  The argument image of the frame.

- **HxImageData** * **result** () const

  The result image of the frame.

6.242.1 Detailed Description

Class definition of method frame for resizing of image.

The method frame is used when the resulting image has a different size than the operand image.

6.242.2 Constructor & Destructor Documentation

6.242.2.1 HxMfResize::HxMfResize (HxImageData * *objImg*, HxSizes *newSize*, HxImageData * *argImg* = 0)

Constructor.

Object() and **result**() (p. 614) will point to an empty image with the signature of objImg and the specified sizes. In case the signature of argImg is not equal to the signature of **object**() (p. 613) a temporary image with that signature will be created for **argument**() (p. 613) with the same pixel values as argImg.

```
17     : _object(objImg), _argument(argImg), _tmpArg(0)
18 {
19     if (!_object)
20         return;
21 
22     HxImageSignature objSig(_object->signature());
23 
24     if (_argument) {
25         HxImageSignature argSig(_argument->signature());
26         if (!argSig.isEqual(objSig)) {
27             _tmpArg = HxImgDataFactory::instance().makeImage(objSig, newSize);
28             _tmpArg->set(_argument);
29             _argument = _tmpArg;
30         }
31     }
32 
33     _result = HxImgDataFactory::instance().makeImage(objSig, newSize);
34 //    if (_result)
35 //        _result->set(_object);
36     _object = _result;
37 }
```

6.242.2.2 HxMfResize::~HxMfResize ()

Destructor.

```
40 {
41     if (_tmpArg)
42         delete _tmpArg;
43 }
```

6.242.3 Member Function Documentation

6.242.3.1 HxImageData * HxMfResize::object () const

The object image of the frame.

```
47 {
48     return _object;
49 }
```

6.242.3.2 HxImageData * HxMfResize::argument () const

The argument image of the frame.

```
53 {
54     return _argument;
55 }
```

6.242.3.3 HxImageData * HxMfResize::result () const

The result image of the frame.

```
59 {
60     return _result;
61 }
```

The documentation for this class was generated from the following files:

- **HxMfResize.h**
- HxMfResize.c

6.243 HxMfTranspose Class Reference

Class definition of method frame for unary pixel operations.

`#include <HxMfTranspose.h>`

Public Methods

- **HxMfTranspose** (**HxImageData** *objImg)

  Constructor.

- ~**HxMfTranspose** ()

  Destructor.

- **HxImageData** * **object** () const

  The object image of the frame.

- **HxImageData** * **result** () const

  The result image of the frame.

6.243.1 Detailed Description

Class definition of method frame for unary pixel operations.

6.243.2 Constructor & Destructor Documentation

6.243.2.1 HxMfTranspose::HxMfTranspose (HxImageData * *objImg*)

Constructor.

```
18     : _object(objImg)
19 {
20     if (!_object)
21         return;
22
23     HxImageSignature objSig(_object->signature());
24
25     HxSizes objSizes = _object->sizes();
26     HxSizes resultSizes = objSizes;
27
28     switch (objSig.imageDimensionality())
29     {
30     case 1  :
31         break;
32     case 2  :
33         resultSizes = HxSizes(objSizes.y(), objSizes.x(), 1);
34         break;
35     case 3  :
36         resultSizes = HxSizes(objSizes.y(), objSizes.z(), objSizes.x());
37         break;
38     }
39     _result = HxImgDataFactory::instance().makeImage(objSig, resultSizes);
40 }
```

6.243.2.2 HxMfTranspose::~HxMfTranspose ()

Destructor.

```
43 {
44 }
```

6.243.3 Member Function Documentation

6.243.3.1 HxImageData * HxMfTranspose::object () const

The object image of the frame.

```
48 {
49     return _object;
50 }
```

6.243.3.2 HxImageData * HxMfTranspose::result () const

The result image of the frame.

```
54 {
55     return _result;
56 }
```

The documentation for this class was generated from the following files:

- **HxMfTranspose.h**
- HxMfTranspose.c

6.244 HxMfUpo Class Reference

Class definition of method frame for unary pixel operations.

`#include <HxMfUpo.h>`

Public Methods

- **HxMfUpo** (**HxImageData** ∗objImg, **HxString** upoName)

  Constructor.

- ∼**HxMfUpo** ()

  Destructor.

- **HxImageData** ∗ **object** () const

  The object image of the frame.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

6.244.1 Detailed Description

Class definition of method frame for unary pixel operations.

6.244.2 Constructor & Destructor Documentation

6.244.2.1 HxMfUpo::HxMfUpo (HxImageData ∗ *objImg*, HxString *upoName*)

Constructor.

```
20     : _object(objImg)
21 {
22     if (!_object)
23         return;
24 
25     HxImageSignature objSig(_object->signature());
26 
27     static HxRegKey* upoKey
28         = HxRegistry::instance().findKey("/imagefunctortable/upo");
29     HxRegKey* k = upoKey ? upoKey->findKey(upoName) : 0;
30     k = k ? k->findKey("resulttype") : 0;
31     const HxRegValue* v = k ? k->findValue(objSig.toString()) : 0;
32     HxImageSignature resultSig
33         = v ? HxImageSignature::NameToSignature(v->getData().toString()) : objSig;
34 
35     HxSizes sizes = _object->sizes();
36 
37     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
38 }
```

6.244.2.2 HxMfUpo::~HxMfUpo ()

Destructor.

```
41 {
42 }
```

6.244.3 Member Function Documentation

6.244.3.1 HxImageData * HxMfUpo::object () const

The object image of the frame.

```
46 {
47     return _object;
48 }
```

6.244.3.2 HxImageData * HxMfUpo::result () const

The result image of the frame.

```
52 {
53     return _result;
54 }
```

The documentation for this class was generated from the following files:

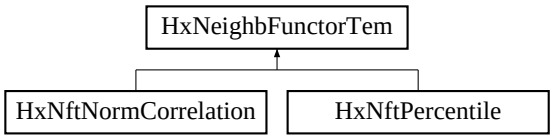
- **HxMfUpo.h**
- HxMfUpo.c

6.245 HxNeighbFunctorTem Class Template Reference

Class definition for neighbourhood functor.

`#include <HxNeighbFunctorTem.h>`

Inheritance diagram for HxNeighbFunctorTem::



Public Methods

- **HxNeighbFunctorTem** (**HxString** name)

  Constructor.

- virtual ~**HxNeighbFunctorTem** ()
- virtual void **init** ()=0

  Initialize the functor.

- virtual void **next** (ArgType pixV, ArgType maskV, **HxPoint** p)=0

  Process the next element.

- virtual ResType **result** ()=0

  Produce result.

- virtual int **hasPhase2** ()

  Does functor have a second phase?.

- virtual void **init2** ()

  Initialize phase 2.

- virtual void **next2** (ArgType pixV, ArgType maskV, **HxPoint** p)

  Process the next element in phase 2.

- virtual ResType **result2** ()

  Produce result phase 2.

6.245.1 Detailed Description

template<class ArgType, class ResType> class HxNeighbFunctorTem< ArgType, ResType >

Class definition for neighbourhood functor.

6.245.2 Constructor & Destructor Documentation

6.245.2.1 template<class ArgType, class ResType> HxNeighbFunctorTem< ArgType, ResType >::HxNeighbFunctorTem (HxString *name*)

Constructor.

```
20 {
21     HxNeighbFunctorTable<ArgType, ResType>::instance()->insert(name, this);
22     _name = name;
23 }
```

6.245.3 Member Function Documentation

6.245.3.1 template<class ArgType, class ResType> virtual void HxNeighbFunctorTem< ArgType, ResType >::init () [pure virtual]

Initialize the functor.

6.245.3.2 template<class ArgType, class ResType> virtual void HxNeighbFunctorTem< ArgType, ResType >::next (ArgType *pixV*, ArgType *maskV*, HxPoint *p*) [pure virtual]

Process the next element.

6.245.3.3 template<class ArgType, class ResType> virtual ResType HxNeighbFunctorTem< ArgType, ResType >::result () [pure virtual]

Produce result.

6.245.3.4 template<class ArgType, class ResType> int HxNeighbFunctorTem< ArgType, ResType >::hasPhase2 () [virtual]

Does functor have a second phase?.

Default: 0.

```
39 {
40     return 0;
41 }
```

6.245.3.5 template<class ArgType, class ResType> void HxNeighbFunctorTem< ArgType, ResType >::init2 () [virtual]

Initialize phase 2.

Default: ignore.

```
46 {
47 }
```

6.245.3.6 template<class ArgType, class ResType> void HxNeighbFunctorTem< ArgType, ResType >::next2 (ArgType *pixV*, ArgType *maskV*, HxPoint *p*) [virtual]

Process the next element in phase 2.

Default: ignore.

```
52 {
53 }
```

6.245.3.7 template<class ArgType, class ResType> ResType HxNeighbFunctorTem< ArgType, ResType >::result2 () [virtual]

Produce result phase 2.

Default: HxScalarInt(0).

```
58 {
59     return HxScalarInt(0);
60 }
```

The documentation for this class was generated from the following files:

- **HxNeighbFunctorTem.h**
- HxNeighbFunctorTem.c

6.246 HxNgbIsMaxGradDir2d Class Template Reference

Non maximum suppression in the direction of the gradient.

`#include <HxNgbIsMaxGradDir2d.h>`

Public Types

- typedef HxNgbCnumTag **IteratorCategory**
- typedef HxNgb1PhaseTag **PhaseCategory**
- typedef HxNgbTransInVarTag **TransVarianceCategory**
- typedef HxCnum **CnumType**

Public Methods

- **HxNgbIsMaxGradDir2d** (**HxTagList** &tags)
- **~HxNgbIsMaxGradDir2d** ()
- **HxSizes size** ()
- CnumType **begin** ()
- CnumType **end** ()
- void **init** (const ArithT &value)
- void **next** (int x, int y, const ArithT &value)
- ResultT **result** () const

Static Public Methods

- **HxString className** ()

6.246.1 Detailed Description

template<class ResultT, class ArithT> class HxNgbIsMaxGradDir2d< ResultT, ArithT >

Non maximum suppression in the direction of the gradient.

If the current pixel is smaller than the two pixels in the gradient and opposite direction it is set to 0. ArithT is required to be a 2 dimensional vector.

The documentation for this class was generated from the following files:

- **HxNgbIsMaxGradDir2d.h**
- HxNgbIsMaxGradDir2d.c

6.247 HxNgbNc2dInst Class Template Reference

Instantiator for kernel based neighbourhood operation with normalized correlation.

Public Attributes

- **HxImgFtorKernelNgb2d**< DstSigT, SrcSigT, KerSigT, **HxNgbNormCorrelation**< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

6.247.1 Detailed Description

template<class DstSigT, class SrcSigT, class KerSigT> class HxNgbNc2dInst< DstSigT, SrcSigT, KerSigT >

Instantiator for kernel based neighbourhood operation with normalized correlation.

6.247.2 Member Data Documentation

6.247.2.1 template<class DstSigT, class SrcSigT, class KerSigT> HxImgFtorKernelNgb2d<DstSigT, SrcSigT, KerSigT, HxNgbNormCorrelation<typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble> > HxNgbNc2dInst::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbNormCorrelationInst.c

6.248 HxNgbNonMaxSuppression2d Class Template Reference

Neighbourhood functor for non maximum suppression in the gradient direction.

`#include <HxNgbNonMaxSuppression2d.h>`

Public Types

- typedef HxNgbCnumTag **IteratorCategory**

  Coordinate enumerator version.

- typedef HxNgb1PhaseTag **PhaseCategory**

  1 phase.

- typedef HxNgbTransInVarTag **TransVarianceCategory**

  Translation invariant.

- typedef HxCnum **CnumType**

Public Methods

- **HxNgbNonMaxSuppression2d** (**HxTagList** &tags)
- **~HxNgbNonMaxSuppression2d** ()
- **HxSizes size** ()
- CnumType **begin** ()
- CnumType & **end** ()
- void **init** (const ArithT &value)
- void **next** (int x, int y, const ArithT &value)
- const ArithT & **result** () const

Static Public Methods

- **HxString className** ()

  The name : "nonMaxSuppression".

6.248.1 Detailed Description

template<class ArithT> class HxNgbNonMaxSuppression2d< ArithT >

Neighbourhood functor for non maximum suppression in the gradient direction.

If the current pixel is smaller than the two pixels in the gradient and opposite direction it is set to 0. ArithT is required to be a 2 dimensional vector.

6.248.2 Member Typedef Documentation

6.248.2.1 template<class ArithT> typedef HxNgbCnumTag HxNgbNonMaxSuppression2d::IteratorCategory

Coordinate enumerator version.

6.248.2.2 template<class ArithT> typedef HxNgb1PhaseTag HxNgbNonMaxSuppression2d::PhaseCategory

1 phase.

6.248.2.3 template<class ArithT> typedef HxNgbTransInVarTag HxNgbNonMaxSuppression2d::TransVarianceCategory

Translation invariant.

6.248.3 Member Function Documentation

6.248.3.1 template<class ArithT> HxString HxNgbNonMaxSuppression2d< ArithT >::className () [inline, static]

The name : "nonMaxSuppression".

```
109 {
110     static HxString s("nonMaxSuppression");
111     return s;
112 }
```

The documentation for this class was generated from the following files:

- **HxNgbNonMaxSuppression2d.h**
- HxNgbNonMaxSuppression2d.c

6.249 HxNgbNonMaxSuppression2dInst Class Template Reference

Instantiator for neighbourhood operation with non maximum suppression.

Public Attributes

- **HxImgFtorNgb2d**< ImgSigT, ImgSigT, **HxNgbNonMaxSuppression2d**< typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.249.1 Detailed Description

template<class ImgSigT> class HxNgbNonMaxSuppression2dInst< ImgSigT >

Instantiator for neighbourhood operation with non maximum suppression.

6.249.2 Member Data Documentation

6.249.2.1 template<class ImgSigT> HxImgFtorNgb2d<ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d<typename ImgSigT::ArithType> > HxNgbNonMaxSuppression2dInst::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbNonMaxSuppression2dInst.c

6.250 HxNgbNormCorrelation Class Template Reference

Neighbourhood functor for normalized (cross) correlation filter.

`#include <HxNgbNormCorrelation.h>`

Public Types

- typedef HxNgbLoopTag **IteratorCategory**

Loop version.

- typedef HxNgb2PhaseTag **PhaseCategory**

  2 phases.

- typedef HxNgbTransInVarTag **TransVarianceCategory**

  Translation invariant.

Public Methods

- **HxNgbNormCorrelation** (**HxTagList** &tags)
- ~**HxNgbNormCorrelation** ()
- **HxSizes size** ()
- void **init** ()
- void **next** (int x, int y, ArithT pixV, ArithT maskV)
- void **init2** ()
- void **next2** (int x, int y, ArithT pixV, ArithT maskV)
- ResultT **result** ()

Static Public Methods

- **HxString className** ()

  The name : "normalizedCorrelation".

6.250.1 Detailed Description

template<class ArithT, class ResultT> class HxNgbNormCorrelation< ArithT, ResultT >

Neighbourhood functor for normalized (cross) correlation filter.

Formula:

\begin{verbatim} Sum[f(x,y) - fBar(x,y)][w(x,y) - wBar] x,y ———————————————————————————————— Sqrt[Sum[f(x,y) - fBar(x,y)]^2 * Sum[w(x,y) - wBar]^2] x,y x,y \end{verbatim}

6.250.2 Member Typedef Documentation

6.250.2.1 template<class ArithT, class ResultT> typedef HxNgbLoopTag HxNgbNormCorrelation::IteratorCategory

Loop version.

6.250.2.2 template<class ArithT, class ResultT> typedef HxNgb2PhaseTag HxNgbNormCorrelation::PhaseCategory

2 phases.

6.250.2.3 template<class ArithT, class ResultT> typedef HxNgbTransInVarTag HxNgbNormCorrelation::TransVarianceCategory

Translation invariant.

6.250.3 Member Function Documentation

6.250.3.1 template<class ArithT, class ResultT> HxString HxNgbNormCorrelation< ArithT, ResultT >::className () [static]

The name : "normalizedCorrelation".

```
30 {
31     static HxString name("normalizedCorrelation");
32     return name;
33 }
```

The documentation for this class was generated from the following files:

- **HxNgbNormCorrelation.h**
- HxNgbNormCorrelation.c

6.251 HxNgbPercentile2d Class Template Reference

Neighbourhood functor for percentile filter.

`#include <HxNgbPercentile2d.h>`

Public Types

- typedef HxNgbLoopTag **IteratorCategory**
  Loop version.
- typedef HxNgb1PhaseTag **PhaseCategory**
  1 phase.
- typedef HxNgbTransInVarTag **TransVarianceCategory**
  Translation invariant.

Public Methods

- **HxNgbPercentile2d** (**HxTagList** &tags)
  Constructor.
- ~**HxNgbPercentile2d** ()
- **HxSizes size** ()
- void **init** ()
- void **next** (int x, int y, ArithT value)
- ArithT **result** () const

Static Public Methods

- **HxString className** ()

  The name : "percentile".

6.251.1 Detailed Description

template<class ArithT> class HxNgbPercentile2d< ArithT >

Neighbourhood functor for percentile filter.

6.251.2 Member Typedef Documentation

6.251.2.1 template<class ArithT> typedef HxNgbLoopTag HxNgbPercentile2d::IteratorCategory

Loop version.

6.251.2.2 template<class ArithT> typedef HxNgb1PhaseTag HxNgbPercentile2d::PhaseCategory

1 phase.

6.251.2.3 template<class ArithT> typedef HxNgbTransInVarTag HxNgbPercentile2d::TransVarianceCategory

Translation invariant.

6.251.3 Constructor & Destructor Documentation

6.251.3.1 template<class ArithT> HxNgbPercentile2d< ArithT >::HxNgbPercentile2d (HxTagList & *tags*)

Constructor.

Taglist should contain: int "size", double "percentile"

```
20     : _values(0)
21 {
22     _size = HxGetTag(tags, "size", 3);
23     double p = HxGetTag(tags, "percentile", 0.5);
24     _pctIdx = p * (_size * _size);
25 
26     _values = new ArithT[_size * _size];
27 
28 }
```

6.251.4 Member Function Documentation

6.251.4.1 template<class ArithT> HxString HxNgbPercentile2d< ArithT >::className () [inline, static]

The name : ”percentile”.

```
70 {
71     static HxString s("percentile");
72     return s;
73 }
```

The documentation for this class was generated from the following files:

- **HxNgbPercentile2d.h**
- HxNgbPercentile2d.c

6.252 HxNgbPercentile2dInst Class Template Reference

Instantiator for neighbourhood operation with percentile.

Public Attributes

- **HxImgFtorNgb2d**< ImgSigT, ImgSigT, **HxNgbPercentile2d**< typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

6.252.1 Detailed Description

template<class ImgSigT> class HxNgbPercentile2dInst< ImgSigT >

Instantiator for neighbourhood operation with percentile.

6.252.2 Member Data Documentation

6.252.2.1 template<class ImgSigT> HxImgFtorNgb2d<ImgSigT, ImgSigT, HxNgbPercentile2d<typename ImgSigT::ArithType> > HxNgbPercentile2dInst::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbPercentile2dInst.c

6.253 HxNJet Class Reference

Class definition for NJet.

`#include <HxNJet.h>`

Public Methods

- **HxNJet** ()

  Constructor.

- **HxNJet** (**HxImageRep** im, int N, double scale, double precision=3)

  Construct an NJet from the given image at the given scale in given precision using Gaussians.

- **HxNJet** (**HxString** fileName)

  Read an NJet from file.

- **HxNJet** (const HxNJet &rhs)

  Copy constructor.

- virtual ~**HxNJet** ()

  Destructor.

- bool **toFile** (**HxString** fileName) const

  Write an NJet to file.

- HxNJet & **operator=** (const HxNJet &rhs)

  Assignment operator.

- int **ident** () const

  The identity of the NJet.

- int **order** () const

  The order N.

- double **scale** () const

  The scale.

- int **nrComponents** () const

  The number of components.

- int **isColor** () const

  Indicator whether its a color or grey value NJet.

- **HxImageRep xy** (int x, int y) const

  Get the specified 2D component.

- **HxImageRep xyz** (int x, int y, int z) const

  Get the specified 3D component.

- **HxImageRep xyl** (int x, int y, int l) const

  Get the specified 2D color component.

- **HxImageRep xyzl** (int x, int y, int z, int l) const

  Get the specified 3D color component.

- **HxImageRep getLidx** (int i) const

  Get the specified L component.

- **HxImageRep getJidx** (int i) const

  Get the specified J component.

- **HxImageRep getMidx** (int i) const

  Get the specified M component.

- **HxImageList getLList** () const

  Get all the L components.

- **HxImageList getJList** () const

  Get all the J components.

- **HxImageList getMList** () const

  Get all the M components.

- **HxImageList getList** () const

  Get all the components.

- **HxImageRep getLw** () const

  Get the L gradient magnitude.

- **HxImageRep getJw** () const

  Get the J gradient magnitude.

- **HxImageRep getMw** () const

  Get the M gradient magnitude.

- void **rotate** (double phi)
- void **rotateDeg** (double phi)
- void **rotate** (**HxImageRep** phi)
- void **resample** (double fac)
- void **truncate** (int order)
- void **normalize** ()
- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Put some information on the given stream.

- int **ord2idx** (int i, int j) const

  Translate from ord to idx.

- int **ord2idx** (int i, int j, int k) const

  Translate from ord to idx 3D (not supported yet).

6.253.1 Detailed Description

Class definition for NJet.

Generates all components up to and including the N-th order in the following sequence: For 2D images:

- order 0 : L,
- order 1 : Lx, Ly,
- order 2 : Lxx, Lxy, Lyy,
- etc.

Color images are stored in the opponent color respresentation, the L component representing luminance (the Gaussian smoothed spectral response), the J component the first order spectral derivative (yellow-blue), and the M component the second order spectral derivative (red-green). Grey images are only stored in the L component, J and M being zero.

Components L, J, and M can be specified in one of the following ways:

- xy : the indices represent the x and y derivative order. For example, xy(0,1) is Ly.
- xyz : the indices represent the x, y, and z derivative order. For example, xyz(1,1,2) is Lxyzz.
- xyl : the indices represent the x, y, and spectral (lambda) derivative order. For example, xyl(1,1,2) is Mxy.
- xyzl: the indices represent the x, y, z, and spectral (lambda) derivative order. For example, xyzl(1,1,1,1) is Jxyz.
- idx : the index is the index in the sequence in which all components are generated (and stored in the internal data structure). For example, getLidx(3) is Ly in case of 2D images. Low level function, preferably do not use.

TODO: IMPLEMENTATION FOR 3D IMAGES AND ORDERS HIGHER THAN 2.

6.253.2 Constructor & Destructor Documentation

6.253.2.1 HxNJet::HxNJet ()

Constructor.

```
15                : _pointee(0)
16 {
17 }
```

6.253.2.2 HxNJet::HxNJet (HxImageRep *im*, int *N*, double *scale*, double *precision* = 3)

Construct an NJet from the given image at the given scale in given precision using Gaussians.

```
20     : _order(N), _scale(scale)
21 {
22     _pointee = HxNJetDataFactory::instance().makeGauss(im, N, scale,
23                                                        precision);
24 }
```

6.253.2.3 HxNJet::HxNJet (HxString *fileName*)

Read an NJet from file.

```
27 {
28     HxTagList tags;
29
30     _pointee = HxNJetDataFactory::instance().fromFile(fileName, tags);
31     _order = HxGetTag(tags, "NJet Order", 4);
32     _scale = HxGetTag(tags, "NJet Scale", 3.0);
33 }
```

6.253.2.4 HxNJet::HxNJet (const HxNJet & *rhs*)

Copy constructor.

```
35                                  : _order(rhs._order), _scale(rhs._scale),
36                                      _pointee(rhs.pointee())
37 {
38 }
```

6.253.2.5 HxNJet::~HxNJet () [virtual]

Destructor.

```
41 {
42 }
```

6.253.3 Member Function Documentation

6.253.3.1 bool HxNJet::toFile (HxString *fileName*) const

Write an NJet to file.

```
46 {
47     HxTagList tags;
48
49     return HxNJetDataFactory::instance().toFile(*this, fileName, tags);
50 }
```

6.253.3.2 HxNJet & HxNJet::operator= (const HxNJet & *rhs*)

Assignment operator.

```
54 {
55     _order = rhs._order;
56     _scale = rhs._scale;
57     _pointee = rhs._pointee;
58     return *this;
59 }
```

6.253.3.3 int HxNJet::ident () const

The identity of the NJet.

```
63 {
64     return pointee() ? pointee()->ident() : 0;
65 }
```

6.253.3.4 int HxNJet::order () const

The order N.

```
69 {
70     return _order;
71 }
```

6.253.3.5 double HxNJet::scale () const

The scale.

```
75 {
76     return _scale;
77 }
```

6.253.3.6 int HxNJet::nrComponents () const

The number of components.

```
81 {
82     return pointee() ? pointee()->nrComponents() : 0;
83 }
```

6.253.3.7 int HxNJet::isColor () const

Indicator whether its a color or grey value NJet.

```
87 {
88     return pointee() ? pointee()->isColor() : 0;
89 }
```

6.253.3.8 HxImageRep HxNJet::xy (int *x*, int *y*) const

Get the specified 2D component.

```
114 {
115     return getLidx(ord2idx(x, y));
116 }
```

6.253.3.9 HxImageRep HxNJet::xyz (int *x*, int *y*, int *z*) const

Get the specified 3D component.

```
120 {
121     return getLidx(ord2idx(x, y, z));
122 }
```

6.253.3.10 HxImageRep HxNJet::xyl (int *x*, int *y*, int *l*) const

Get the specified 2D color component.

```
126 {
127     switch (l) {
128     case 0:
129         return getLidx(ord2idx(x, y));
130     case 1:
131         return getJidx(ord2idx(x, y));
132     case 2:
133         return getMidx(ord2idx(x, y));
134     }
135     return HxImageRep();
136 }
```

6.253.3.11 HxImageRep HxNJet::xyzl (int *x*, int *y*, int *z*, int *l*) const

Get the specified 3D color component.

```
140 {
141     switch (l) {
142     case 0:
143         return getLidx(ord2idx(x, y, z));
144     case 1:
145         return getJidx(ord2idx(x, y, z));
146     case 2:
147         return getMidx(ord2idx(x, y, z));
148     }
149     return HxImageRep();
150 }
```

6.253.3.12 HxImageRep HxNJet::getLidx (int *i*) const

Get the specified L component.

```
93 {
94     HxImageRep im;
95     return pointee() ? pointee()->getL(i) : im;
96 }
```

6.253.3.13 HxImageRep HxNJet::getJidx (int *i*) const

Get the specified J component.

```
100 {
101     HxImageRep im;
102     return pointee() ? pointee()->getJ(i) : im;
103 }
```

6.253.3.14 HxImageRep HxNJet::getMidx (int *i*) const

Get the specified M component.

```
107 {
108     HxImageRep im;
109     return pointee() ? pointee()->getM(i) : im;
110 }
```

6.253.3.15 HxImageList HxNJet::getLList () const

Get all the L components.

```
154 {
155     HxImageList l;
156 
157     for (int i=0; i < nrComponents(); i++)
158         l += getLidx(i);
159 
160     return l;
161 }
```

6.253.3.16 HxImageList HxNJet::getJList () const

Get all the J components.

```
165 {
166     HxImageList l;
167 
168     if (isColor()) {
169         for (int i=0; i < nrComponents(); i++)
170             l += getJidx(i);
171     }
172 
173     return l;
174 }
```

6.253.3.17 HxImageList HxNJet::getMList () const

Get all the M components.

```
178 {
179     HxImageList l;
180 
181     if (isColor()) {
182         for (int i=0; i < nrComponents(); i++)
183             l += getMidx(i);
184     }
185 
186     return l;
187 }
```

6.253.3.18 HxImageList HxNJet::getList () const

Get all the components.

```
191 {
192     HxImageList l;
193 
194     HxImageList ll = getLList();
195     HxImageList jl = getJList();
196     HxImageList ml = getMList();
197 
198     l = ll + jl;
199     l = l + ml;
200 
201     return l;
202 }
```

6.253.3.19 HxImageRep HxNJet::getLw () const

Get the L gradient magnitude.

```
206 {
207     HxImageRep im;
208     return pointee() ? pointee()->getLw() : im;
209 }
```

6.253.3.20 HxImageRep HxNJet::getJw () const

Get the J gradient magnitude.

```
213 {
214     HxImageRep im;
215     return pointee() ? pointee()->getJw() : im;
216 }
```

6.253.3.21 HxImageRep HxNJet::getMw () const

Get the M gradient magnitude.

```
220 {
221     HxImageRep im;
222     return pointee() ? pointee()->getMw() : im;
223 }
```

6.253.3.22 STD_OSTREAM & HxNJet::put (STD_OSTREAM & *os*) const

Put some information on the given stream.

```
265 {
266     if (isColor())
267         os << "Color ";
268     os << "NJet " << ident() << ", N = " << order()
269        << ", scale = " << scale() << STD_ENDL;
270     return os;
271 }
```

6.253.3.23 int HxNJet::ord2idx (int *i*, int *j*) const [inline]

Translate from ord to idx.

```
161                             {   int n = i+j;
162                                 return n*(n+1)/2+j;}
```

6.253.3.24 int HxNJet::ord2idx (int *i*, int *j*, int *k*) const [inline]

Translate from ord to idx 3D (not supported yet).

```
166                             {   int n = i+j;
167                                 return n*(n+1)/2+j;}
```

The documentation for this class was generated from the following files:

- **HxNJet.h**
- HxNJet.c

6.254 HxPixOp1PhaseTag Struct Reference

1 phase pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

6.254.1 Detailed Description

1 phase pixel operation.

6.254.2 Member Function Documentation

6.254.2.1 HxString HxPixOp1PhaseTag::toString () [inline]

Convert tag to string.

```
24 { return "1 phase"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.255 HxPixOp2PhaseTag Struct Reference

2 phase pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

6.255.1 Detailed Description

2 phase pixel operation.

6.255.2 Member Function Documentation

6.255.2.1 HxString HxPixOp2PhaseTag::toString () [inline]

Convert tag to string.

```
30 { return "2 phase"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.256 HxPixOpInTag Struct Reference

In pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

6.256.1 Detailed Description

In pixel operation.

6.256.2 Member Function Documentation

6.256.2.1 HxString HxPixOpInTag::toString () [inline]

Convert tag to string.

```
58 { return "In"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.257 HxPixOpNPhaseTag Struct Reference

N phase pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

6.257.1 Detailed Description

N phase pixel operation.

6.257.2 Member Function Documentation

6.257.2.1 HxString HxPixOpNPhaseTag::toString () [inline]

Convert tag to string.

```
36 { return "N phase"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.258 HxPixOpOutTag Struct Reference

Out pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

6.258.1 Detailed Description

Out pixel operation.

6.258.2 Member Function Documentation

6.258.2.1 HxString HxPixOpOutTag::toString () `[inline]`

Convert tag to string.

```
64 { return "Out"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.259 HxPixOpTransInVarTag Struct Reference

Translation invariant pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

6.259.1 Detailed Description

Translation invariant pixel operation.

6.259.2 Member Function Documentation

6.259.2.1 HxString HxPixOpTransInVarTag::toString () [inline]

Convert tag to string.

```
50 { return "Translation invariant"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.260 HxPixOpTransVarTag Struct Reference

Translation variant pixel operation.

`#include <HxPixOpCategory.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

6.260.1 Detailed Description

Translation variant pixel operation.

6.260.2 Member Function Documentation

6.260.2.1 HxString HxPixOpTransVarTag::toString () [inline]

Convert tag to string.

```
44 { return "Translation variant"; }
```

The documentation for this struct was generated from the following file:

- **HxPixOpCategory.h**

6.261 HxPointList Class Reference

Class definition for list of HxPoint's.

`#include <HxPointList.h>`

Public Types

- typedef std::back_insert_iterator< HxPointList > **back_insert_iterator**

  back inserter.

Public Methods

- HxPointList & **operator<<** (const **HxPoint** &)

  Add point to the list.

- void **eraseAll** ()

  Remove all points from the list.

6.261.1 Detailed Description

Class definition for list of HxPoint's.

Specialization of list from STL.

6.261.2 Member Typedef Documentation

6.261.2.1 typedef std::back_insert_iterator<HxPointList> HxPointList::back_insert_iterator

back inserter.

6.261.3 Member Function Documentation

6.261.3.1 HxPointList & HxPointList::operator<< (const HxPoint & *s*) [inline]

Add point to the list.

```
48 {
49     push_back(s);
50     return *this;
51 }
```

6.261.3.2 void HxPointList::eraseAll () [inline]

Remove all points from the list.

```
55 {
56     erase(begin(), end());
57 }
```

The documentation for this class was generated from the following file:

- **HxPointList.h**

6.262 HxPointR2 Class Reference

Class definition for points in R2 (real-value coordinates).

`#include <HxPointR2.h>`

Public Methods

- **HxPointR2** ()
  Constructor.
- **HxPointR2** (double d1, double d2)
  Constructor.
- **HxPointR2** (const **HxVec2Double** &v)
  Copy constructor.
- double **x** () const
  Get the x coordinate of the point.
- double **y** () const
  Get the y coordinate of the point.
- HxPointR2 **add** (const **HxVectorR2** &arg) const
  Add the given vector to this point.
- HxPointR2 **sub** (const **HxVectorR2** &arg) const
  Subtract the given vector from this point.
- STD_OSTREAM & **put** (STD_OSTREAM &) const
  Put the arrow on the given stream.
- STD_OSTREAM & **dump** (HxPointR2 &) const
- **HxString toString** () const

Friends

- class **HxVectorR2**

6.262.1 Detailed Description

Class definition for points in R2 (real-value coordinates).

6.262.2 Constructor & Destructor Documentation

6.262.2.1 HxPointR2::HxPointR2 () `[inline]`

Constructor.

```
68                      : _data(0,0)
69 {
70 }
```

6.262.2.2 HxPointR2::HxPointR2 (double *d1*, double *d2*) [inline]

Constructor.

```
73                                      : _data(d1, d2)
74 {
75 }
```

6.262.2.3 HxPointR2::HxPointR2 (const HxVec2Double & *v*) [inline]

Copy constructor.

```
78                                       : _data(v)
79 {
80 }
```

6.262.3 Member Function Documentation

6.262.3.1 double HxPointR2::x () const [inline]

Get the x coordinate of the point.

```
84 {
85     return _data.x();
86 }
```

6.262.3.2 double HxPointR2::y () const [inline]

Get the y coordinate of the point.

```
90 {
91     return _data.y();
92 }
```

6.262.3.3 HxPointR2 HxPointR2::add (const HxVectorR2 & *arg*) const

Add the given vector to this point.

```
16 {
17     return HxPointR2(_data + arg._data );
18
19 }
```

6.262.3.4 HxPointR2 HxPointR2::sub (const HxVectorR2 & *arg*) const

Subtract the given vector from this point.

```
23 {
24     return HxPointR2(_data - arg._data);
25 }
```

6.262.3.5 STD_OSTREAM & HxPointR2::put (STD_OSTREAM & *os*) const [inline]

Put the arrow on the given stream.

```
102 {
103     return os << _data;
104 }
```

The documentation for this class was generated from the following files:

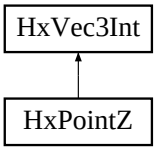
- **HxPointR2.h**
- HxPointR2.c

6.263 HxPointZ Class Reference

Definition of a point in Z3 space using integer coordinates.

`#include <HxPointZ.h>`

Inheritance diagram for HxPointZ::



Public Methods

- **HxPointZ** (int x=0, int y=0, int z=0)

  Constructor.

- **HxPointZ** (const **HxVec3Int** &rhs)

  Constructor.

- **HxPointZ** (const **HxVec3Double** &rhs)

  Constructor.

6.263.1 Detailed Description

Definition of a point in Z3 space using integer coordinates.

HxPointZ is used primarily to specify the position of a pixel in an image.

6.263.2 Constructor & Destructor Documentation

6.263.2.1 HxPointZ::HxPointZ (int *x* = 0, int *y* = 0, int *z* = 0) [inline]

Constructor.

```
34                                      : HxVec3Int(x, y, z)
35 {
36 }
```

6.263.2.2 HxPointZ::HxPointZ (const HxVec3Int & *rhs*) [inline]

Constructor.

```
40     : HxVec3Int(rhs.x(), rhs.y(), rhs.z())
41 {
42 }
```

6.263.2.3 HxPointZ::HxPointZ (const HxVec3Double & *rhs*) [inline]

Constructor.

```
46     : HxVec3Int(rhs.x()+0.5, rhs.y()+0.5, rhs.z()+0.5)
47 {
48 }
```

The documentation for this class was generated from the following file:

- **HxPointZ.h**

6.264 HxPointZList Class Reference

Class definition for list of **HxPointZ** (p. 644)'s.

`#include <HxPointZList.h>`

Public Methods

- HxPointZList & **operator**<< (const **HxPointZ** &)
  Add point to the list.
- void **eraseAll** ()
  Remove all points from the list.

6.264.1 Detailed Description

Class definition for list of **HxPointZ** (p. 644)'s.

Specialization of list from STL.

6.264.2 Member Function Documentation

6.264.2.1 HxPointZList & HxPointZList::operator<< (const HxPointZ & *s*) [inline]

Add point to the list.

```
97 {
98     push_back(s);
99     return *this;
100 }
```

6.264.2.2 void HxPointZList::eraseAll () [inline]

Remove all points from the list.

```
104 {
105     erase(begin(), end());
106 }
```

The documentation for this class was generated from the following file:

- **HxPointZList.h**

6.265 HxPolyline2d Class Reference

A simple class for storing polylines and polygons.

`#include <HxPolyline2d.h>`

Public Methods

- **HxPolyline2d** ()

  Construct an empty polyline.

- **HxPolyline2d** (const HxPointSetR2 &v, int close)

  Construct a polyline or polygon (closed) from the given set of points.

- **HxPolyline2d** (double *px, double *py, int np, int close)

  Construct a polyline or polygon (closed) from the given set of coordinates.

- **~HxPolyline2d** ()

  Destructor.

- int **ident** () const

  Get the identifier of this object.

- HxPointSetR2 **getPoints** () const

  Get the points of the object.

- int **getClosed** () const

  Indicate whether this is a polyline or polygon.

- int **getNrPoints** () const

  Get the number of points.

- **HxPointR2 getPoint** (int index) const

  Get the points at the given index.

- void **getPoints** (double *px, double *py) const

  Fill the given arrays with the coordinates of the points of this object.

- STD_OSTREAM & **put** (STD_OSTREAM &) const

  Put this object on the given output stream.

6.265.1 Detailed Description

A simple class for storing polylines and polygons.

The difference between a polyline and polygon is the closed parameter. The number of points is the same, i.e. in a polygon the start/end point is NOT repeated in the list of points.

6.265.2 Constructor & Destructor Documentation

6.265.2.1 HxPolyline2d::HxPolyline2d () [inline]

Construct an empty polyline.

```
80 {
81     _ident = _nr++;
82 }
```

6.265.2.2 HxPolyline2d::HxPolyline2d (const HxPointSetR2 & *v*, int *close*) [inline]

Construct a polyline or polygon (closed) from the given set of points.

```
85                                                      :
86     _points(v), _closed(close)
87 {
88     _ident = _nr++;
89 }
```

6.265.2.3 HxPolyline2d::HxPolyline2d (double * *px*, double * *py*, int *np*, int *close*) [inline]

Construct a polyline or polygon (closed) from the given set of coordinates.

```
93 {
94     for (int i=0 ; i<np ; i++)
95         _points.push_back(HxPointR2(*px++, *py++));
96     _closed = close;
97     _ident = _nr++;
98 }
```

6.265.2.4 HxPolyline2d::~HxPolyline2d () [inline]

Destructor.

```
102 {
103 }
```

6.265.3 Member Function Documentation

6.265.3.1 int HxPolyline2d::ident () const [inline]

Get the identifier of this object.

```
107 {
108     return _ident;
109 }
```

6.265.3.2 HxPointSetR2 HxPolyline2d::getPoints () const [inline]

Get the points of the object.

```
113 {
114     return _points;
115 }
```

6.265.3.3 int HxPolyline2d::getClosed () const [inline]

Indicate whether this is a polyline or polygon.

```
119 {
120     return _closed;
121 }
```

6.265.3.4 int HxPolyline2d::getNrPoints () const [inline]

Get the number of points.

```
125 {
126     return _points.size();
127 }
```

6.265.3.5 HxPointR2 HxPolyline2d::getPoint (int *index*) const [inline]

Get the points at the given index.

```
131 {
132     return _points[index];
133 }
```

6.265.3.6 void HxPolyline2d::getPoints (double * *px*, double * *py*) const

Fill the given arrays with the coordinates of the points of this object.

The size of the araays is assumed to match the number of points.

```
18 {
19     HxPointSetR2::const_iterator it = _points.begin();
20     while (it != _points.end()) {
21         *px++ = (*it).x();
22         *py++ = (*it).y();
23         it++;
24     }
25 }
```

6.265.3.7 STD_OSTREAM & HxPolyline2d::put (STD_OSTREAM & *os*) const

Put this object on the given output stream.

```
29 {
30     HxPointSetR2::const_iterator it = _points.begin();
31     while (it != _points.end()) {
32         os << (*it) << ", ";
33         it++;
34     }
35     return os << "closed: " << _closed << STD_ENDL;
36 }
```

The documentation for this class was generated from the following files:

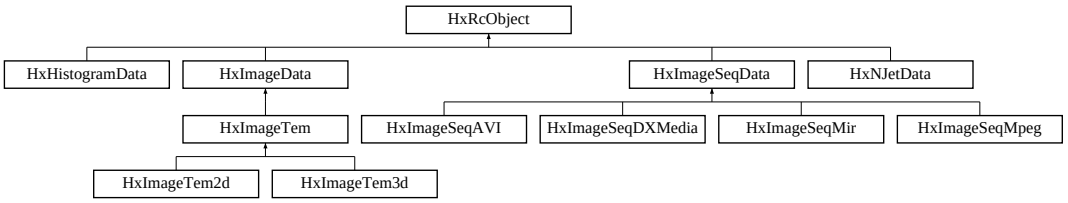
- **HxPolyline2d.h**
- HxPolyline2d.c

6.266 HxRcObject Class Reference

Base class for reference counted objects.

`#include <HxRcObject.h>`

Inheritance diagram for HxRcObject::



Public Methods

- HxRcObject * **addRef** ()
- void **removeRef** ()
- HxRcObject * **assign** (HxRcObject *rhs)
- int **isShared** () const
- HxRcObject * **getUnshared** ()
- HxRcObject * **doGetUnshared** ()
- virtual HxRcObject * **clone** () const
- int **refCnt** () const

Protected Methods

- **HxRcObject** ()
- virtual ~**HxRcObject** ()

6.266.1 Detailed Description

Base class for reference counted objects.

Use with **HxRcPtr** (p. 650).

The documentation for this class was generated from the following files:

- **HxRcObject.h**
- HxRcObject.c

6.267 HxRcPtr Class Template Reference

Template class for (smart) pointers to reference counted objects, i.e.

`#include <HxRcPtr.h>`

Public Methods

- **HxRcPtr** (const T *ptr=0)
- **HxRcPtr** (const HxRcPtr &rhs)
- ~**HxRcPtr** ()
- HxRcPtr & **operator=** (const HxRcPtr &rhs)
- HxRcPtr & **operator=** (T *rhs)
- T * **operator** -> () const

- T & **operator** * () const
- **operator int** () const
- T * **pointee** () const
- int **isShared** () const
- void **getUnshared** ()
- int **refCnt** () const

6.267.1 Detailed Description

template<class T> class HxRcPtr< T >

Template class for (smart) pointers to reference counted objects, i.e.

objects derived from **HxRcObject** (p. 649).

The documentation for this class was generated from the following file:

- **HxRcPtr.h**

6.268 HxRgbBinary Class Template Reference

Binary display.

`#include <HxRgbBinary.h>`

Public Methods

- **HxRgbBinary** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.268.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbBinary< ValT, ValDoubleT >

Binary display.

pixV is 0 or 1.

The documentation for this class was generated from the following file:

- **HxRgbBinary.h**

6.269 HxRgbCMY Class Template Reference

CMY display.

`#include <HxRgbCMY.h>`

Public Methods

- **HxRgbCMY** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.269.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbCMY< ValT, ValDoubleT >

CMY display.

pixV is CMY color.

The documentation for this class was generated from the following file:

- **HxRgbCMY.h**

6.270 HxRgbDirect Class Template Reference

Direct display mapping.

`#include <HxRgbDirect.h>`

Public Methods

- **HxRgbDirect** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.270.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbDirect< ValT, ValDoubleT >

Direct display mapping.

pixV is RGB value in the range 0-255. If not, pixV is clipped.

The documentation for this class was generated from the following file:

- **HxRgbDirect.h**

6.271 HxRgbDirectNC Class Template Reference

Direct display mapping pixV is RGB value in the range 0-255.

`#include <HxRgbDirect.h>`

Public Methods

- **HxRgbDirectNC** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.271.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbDirectNC< ValT, ValDoubleT >

Direct display mapping pixV is RGB value in the range 0-255.

No clipping if pixV is out of range.

The documentation for this class was generated from the following file:

- **HxRgbDirect.h**

6.272 HxRgbHSI Class Template Reference

HSI display.

`#include <HxRgbHSI.h>`

Public Methods

- **HxRgbHSI** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.272.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbHSI< ValT, ValDoubleT >

HSI display.

pixV is HSI color.

The documentation for this class was generated from the following file:

- **HxRgbHSI.h**

6.273 HxRgbLab Class Template Reference

Lab display.

`#include <HxRgbLab.h>`

Public Methods

- **HxRgbLab** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.273.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLab< ValT, ValDoubleT >

Lab display.

pixV is Lab color.

The documentation for this class was generated from the following file:

- **HxRgbLab.h**

6.274 HxRgbLabel Class Template Reference

Labeled display.

`#include <HxRgbLabel.h>`

Public Methods

- **HxRgbLabel** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.274.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLabel< ValT, ValDoubleT >

Labeled display.

PixV is mapped onto 1 of 8 colors.

The documentation for this class was generated from the following file:

- **HxRgbLabel.h**

6.275 HxRgbLogMag Class Template Reference

Log manitude display.

`#include <HxRgbLogMag.h>`

Public Methods

- **HxRgbLogMag** (**HxTagList** &tags)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.275.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLogMag< ValT, ValDoubleT >

Log manitude display.

log(1.0+norm2(pixV)) is stretched between lowVal and highVal.

The documentation for this class was generated from the following file:

- **HxRgbLogMag.h**

6.276 HxRgbLuv Class Template Reference

Luv display.

`#include <HxRgbLuv.h>`

Public Methods

- **HxRgbLuv** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.276.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLuv< ValT, ValDoubleT >

Luv display.

pixV is Luv color.

The documentation for this class was generated from the following file:

- **HxRgbLuv.h**

6.277 HxRgbOOO Class Template Reference

OOO display.

`#include <HxRgbOOO.h>`

Public Methods

- **HxRgbOOO** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.277.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbOOO< ValT, ValDoubleT >

OOO display.

pixV is OOO color.

The documentation for this class was generated from the following file:

- **HxRgbOOO.h**

6.278 HxRgbStretch Class Template Reference

Stretched display.

`#include <HxRgbStretch.h>`

Public Methods

- **HxRgbStretch** (**HxTagList** &tags)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.278.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbStretch< ValT, ValDoubleT >

Stretched display.

pixV is strectched between lowVal and highVal.

The documentation for this class was generated from the following file:

- **HxRgbStretch.h**

6.279 HxRgbXYZ Class Template Reference

XYZ display.

`#include <HxRgbXYZ.h>`

Public Methods

- **HxRgbXYZ** (**HxTagList** &)
- int **doIt** (const ValT &pixV)
- int **doItDouble** (const ValDoubleT &pixV)

Static Public Methods

- **HxString className** ()

6.279.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbXYZ< ValT, ValDoubleT >

XYZ display.

pixV is XYZ color.

The documentation for this class was generated from the following file:

- **HxRgbXYZ.h**

6.280 HxSampledBSplineCurve Class Reference

Class definition for sampled BSpline curves.

`#include <HxSampledBSplineCurve.h>`

Public Methods

- **HxSampledBSplineCurve** ()

  Construct default curve.

- **HxSampledBSplineCurve** (const **HxBSplineCurve** &curve, int nSamples, HxBSplineSamplingAlg algorithm=samplingResolution)

  Construct a curve with given number of samples.

- **HxSampledBSplineCurve** (const **HxBSplineCurve** &curve, double pathInterval, HxBSpline-SamplingAlg algorithm)

  Construct a curve with given path interval between samples.

- ~**HxSampledBSplineCurve** ()

  Destructor.

- int **ident** () const

  Get the identifier.

- **HxBSplineCurve continuousCurve** () const

  Get the continuous curve of this object.

- HxBSplineType **curveType** () const

  Get the type of this curve.

- HxBSplineSamplingAlg **samplingAlg** () const

  Get the sampling algorithm.

- int **nSamples** () const

  Get the number of samples.

- double **sampledT** (int j) const

  Get value of t for sample j.

- vector< double > **allSampledT** () const

  Get value of t for all samples.

- int **indexOfT** (double t) const

Get index of sample corresponding to given t.

- double **dT** (int j) const

  sampling interval at sample j .

- **HxSampledBSplineInterval sampledInterval** (int j1, int j2) const

  Get curve interval defined by two curve samples j1, j2.

- vector< int > **samplesAffectedBy** (int i) const

  Get list of curve samples affected by control point i.

- **HxSampledBSplineInterval intervalAffectedBy** (int i) const

  Get curve interval affected by control point i.

- vector< int > **PThatAffectSample** (int i) const

  Get all control points with influence on the position of curve sample i.

- double **B** (int i, int j) const

  Get value of basis i for sample j.

- vector< double > **BAll** (int j) const

  Get value of all basis that affect sample j.

- double **dB** (int order, int i, int j) const

  Get derivative of basis i for sample j.

- vector< double > **dBAll** (int order, int j) const

  Get derivative of all basis that affect sample j (given order).

- **HxPointR2 C** (int j) const

  Get curve point for sample j.

- HxPointSetR2 **AllC** () const

  Get all sampled curve points.

- **HxPolyline2d CPoly** () const

  Polyline with all sampled curve points.

- **HxVectorR2 dC** (int order, int j) const

  Get curve derivative at sample j.

- vector< **HxVectorR2** > **dCAll** (int order) const

  Get curve derivative at all samples.

- double **kAtC** (int j) const

  Get curvature at sample j.

- vector< double > **kAtCAll** () const

  Get curve derivative at all samples.

- double **dTurnAngleAtC** (int j) const
  Derivative of turning angle at sample j.
- vector< double > **dTurnAngleAtCAll** () const
  Derivative of turning angle at all samples.
- double **length** () const
  total curve length.
- double **length** (int j1, int j2) const
  length of interval between given samples.
- double **length** (const **HxSampledBSplineInterval** &interval) const
  length of given interval.
- int **closestSample** (const **HxPointR2** &p) const
  Get index of sample that is closest to the given point.
- int **numP** () const
  Get number of control points.
- HxPointSetR2 **allP** () const
  Get all control points.
- **HxPolyline2d controlP** () const
  Get the control polygon.
- HxSampledBSplineCurve **changeAllP** (const HxPointSetR2 &p) const
  Replace all control points by given points.
- HxSampledBSplineCurve **translateCurve** (const **HxVectorR2** &v, const **HxSampledBSpline-Interval** &interval) const
  Add vector to control points that affect the curve in the given interval.
- STD_OSTREAM & **dump** (STD_OSTREAM &) const
  Dump the curve on the given stream.

Static Public Methods

- HxSampledBSplineCurve **makeUniform** (**HxPolyline2d** cp, int degree, double distance)
  Make a curve with uniform knots.
- HxSampledBSplineCurve **makeInterpolating** (**HxPolyline2d** cp, double distance)
  Make an interpolating curve.

6.280.1 Detailed Description

Class definition for sampled BSpline curves.

6.280.2 Constructor & Destructor Documentation

6.280.2.1 HxSampledBSplineCurve::HxSampledBSplineCurve ()

Construct default curve.

```
19 {
20     _ident = _nr++;
21     makeDefault();
22 }
```

6.280.2.2 HxSampledBSplineCurve::HxSampledBSplineCurve (const HxBSplineCurve & *curve*, int *n*, HxBSplineSamplingAlg *algorithm* = samplingResolution)

Construct a curve with given number of samples.

```
25                                          : _curve(curve)
26 {
27     _ident = _nr++;
28     if ( algorithm != samplingResolution ) {
29         message("(constructor) invalid parameters - using default");
30         makeDefault();
31     }
32     else
33         sampleWithResolution(n);
34 }
```

6.280.2.3 HxSampledBSplineCurve::HxSampledBSplineCurve (const HxBSplineCurve & *curve*, double *delta*, HxBSplineSamplingAlg *algorithm*)

Construct a curve with given path interval between samples.

```
37                                                  : _curve(curve)
38 {
39     _ident = _nr++;
40     if ( algorithm != samplingInterval ||
41         delta < 0 || delta > (_curve.maxT() -_curve.minT()) ) {
42         message("(constructor) invalid sampling interval - using default");
43         makeDefault();
44     }
45     else
46         sampleWithInterval(delta);
47 }
```

6.280.2.4 HxSampledBSplineCurve::~HxSampledBSplineCurve ()

Destructor.

```
73 {
74 }
```

6.280.3 Member Function Documentation

6.280.3.1 HxSampledBSplineCurve HxSampledBSplineCurve::makeUniform (HxPolyline2d *cp*, int *degree*, double *distance*) [static]

Make a curve with uniform knots.

```
52 {
53     HxBSplineCurve cc = HxBSplineCurve::makeUniform(cp, degree);
54     double l = cc.length();
55     int n = int(l / distance);
56     if (n < cc.numP())
57         n = cc.numP();
58     return HxSampledBSplineCurve(cc, n);
59 }
```

6.280.3.2 HxSampledBSplineCurve HxSampledBSplineCurve::makeInterpolating (HxPolyline2d *cp*, double *distance*) [static]

Make an interpolating curve.

```
63 {
64     HxBSplineCurve cc = HxBSplineCurve::makeInterpolating(cp);
65     double l = cc.length();
66     int n = int(l / distance);
67     if (n < cc.numP())
68         n = cc.numP();
69     return HxSampledBSplineCurve(cc, n);
70 }
```

6.280.3.3 int HxSampledBSplineCurve::ident () const [inline]

Get the identifier.

```
219 {
220     return _ident;
221 }
```

6.280.3.4 HxBSplineCurve HxSampledBSplineCurve::continuousCurve () const [inline]

Get the continuous curve of this object.

```
225 {
226     return _curve;
227 }
```

6.280.3.5 HxBSplineType HxSampledBSplineCurve::curveType () const [inline]

Get the type of this curve.

```
231 {
232     return _curve.curveType();
233 }
```

6.280.3.6 HxBSplineSamplingAlg HxSampledBSplineCurve::samplingAlg () const [inline]

Get the sampling algorithm.

```
255 {
256     return _samplingAlg;
257 }
```

6.280.3.7 int HxSampledBSplineCurve::nSamples () const [inline]

Get the number of samples.

```
261 {
262     return _t.size();
263 }
```

6.280.3.8 double HxSampledBSplineCurve::sampledT (int *j*) const

Get value of t for sample j.

```
110 {
111     if ( j < 0 || j >= nSamples() ) {
112         message("(sampledT) invalid j - setting to 0");
113         j = 0;
114     }
115     return _t[j];
116 }
```

6.280.3.9 vector< double > HxSampledBSplineCurve::allSampledT () const [inline]

Get value of t for all samples.

```
267 {
268     return _t;
269 }
```

6.280.3.10 int HxSampledBSplineCurve::indexOfT (double *t*) const

Get index of sample corresponding to given t.

```
83 {
84     if ( t < _curve.minT() ) {
85         message("(indexOfT) invalid t - setting to minT()");
86         return 0;
87     }
88     if ( t >=  _curve.maxT() ) {
89         message("(indexOfT) invalid t - setting near to maxT()");
90         return nSamples()-1;
91     }
92     int last = _t.size()-1;
93     int j;
```

```
94     for ( j=0; j < last && t > _t[j]; j++ );
95     if ( t == _t[j] )
96         return j;
97     else    {       // get closest
98         double da = absolute(t - _t[j]);
99         double db = absolute(t - _t[j-1]);
100         return (da <= db) ? j : j-1;
101     }
102 }
```

6.280.3.11 double HxSampledBSplineCurve::dT (int *j*) const [inline]

sampling interval at sample j .

fixed value -> only works for uniform sampling!

```
297 {
298     if ( j < 0 || j >= nSamples() ) {
299         message("(dT) invalid j - setting to 0");
300         j = 0;
301     }
302     return _dt; // only for uniform!
303 }
```

6.280.3.12 HxSampledBSplineInterval HxSampledBSplineCurve::sampledInterval (int *j1*, int *j2*) const

Get curve interval defined by two curve samples j1, j2.

```
124 {
125     if ( j1 >= j2 && curveType() != closed ) {
126         message("(sampledInterval) invalid interval - setting to complete curve]");
127         j1 = 0;
128         j2 = nSamples()-1;
129     }
130     return HxSampledBSplineInterval(j1,j2,
131             nSamples()-1, curveType());
132 }
```

6.280.3.13 vector< int > HxSampledBSplineCurve::samplesAffectedBy (int *i*) const

Get list of curve samples affected by control point i.

```
141 {
142     HxBSplineInterval tmp = _curve.pathAffectedBy(i);
143     vector<int> vec;
144
145     for ( int j=0; j < nSamples(); j++ )
146         if ( tmp.contains(_t[j]) )
147             vec.push_back(j);
148
149     return vec;
150 }
```

6.280.3.14 HxSampledBSplineInterval HxSampledBSplineCurve::intervalAffectedBy (int *j*) const

Get curve interval affected by control point i.

```
158 {
159     vector<int> tmp = samplesAffectedBy(j);
160     int j1;
161     int j2;
162 
163     if ( curveType() == closed ) {
164                     // special treatment for wrapping
165        for ( int i=0; i<tmp.size()-1; i++ )
166         if ( (tmp[i+1] - tmp[i]) != 1 ) {
167             j1 = tmp[i+1];
168             j2 = tmp[i];
169             return HxSampledBSplineInterval(j1,j2,
170                 nSamples(), curveType());
171         }
172     }
173     j1 = tmp[0];
174     j2 = tmp[ tmp.size()-1 ];
175     return HxSampledBSplineInterval(j1,j2,
176         nSamples(), curveType());
177 }
```

6.280.3.15 vector< int > HxSampledBSplineCurve::PThatAffectSample (int *i*) const [inline]

Get all control points with influence on the position of curve sample i.

```
273 {
274     return _curve.PThatAffectCAt(_t[i]);
275 }
```

6.280.3.16 double HxSampledBSplineCurve::B (int *i*, int *j*) const [inline]

Get value of basis i for sample j.

```
287 {
288     if ( j < 0 || j >= nSamples() ) {
289         message("(B) invalid j - setting to 0");
290         j = 0;
291     }
292     return _curve.B( i, _t[j]);
293 }
```

6.280.3.17 vector< double > HxSampledBSplineCurve::BAll (int *i*) const

Get value of all basis that affect sample j.

```
185 {
186     vector<double> tmp(nSamples());
187     for ( int j=0; j < nSamples(); j++)
188         tmp[j] = _curve.B(i, _t[j]);
189     return tmp;
190 }
```

6.280.3.18 double HxSampledBSplineCurve::dB (int *order*, int *i*, int *j*) const [inline]

Get derivative of basis i for sample j.

```
307 {
308     if ( j < 0 || j >= nSamples() ) {
309         message("(B) invalid j - setting to 0");
310         j = 0;
311     }
312     return _curve.dB( order, i, _t[j]);
313 }
```

6.280.3.19 vector< double > HxSampledBSplineCurve::dBAll (int *order*, int *i*) const

Get derivative of all basis that affect sample j (given order).

```
199 {
200     vector<double> tmp(nSamples());
201     for ( int j=0; j < nSamples(); j++)
202         tmp[j] = _curve.dB(order, i, _t[j]);
203     return tmp;
204 }
```

6.280.3.20 HxPointR2 HxSampledBSplineCurve::C (int *j*) const [inline]

Get curve point for sample j.

```
317 {
318     if ( j < 0 || j >= nSamples() ) {
319         message("(C) invalid j - setting to 0");
320         j = 0;
321     }
322     return _curve.C(_t[j]);
323 }
```

6.280.3.21 HxPointSetR2 HxSampledBSplineCurve::AllC () const

Get all sampled curve points.

```
212 {
213     vector<HxPointR2> tmp(nSamples());
214     for ( int j=0; j < nSamples(); j++)
215         tmp[j] = _curve.C(_t[j]);
216     return tmp;
217 }
```

6.280.3.22 HxPolyline2d HxSampledBSplineCurve::CPoly () const [inline]

Polyline with all sampled curve points.

```
327 {
328     return HxPolyline2d(AllC(), (curveType() == closed));
329 }
```

6.280.3.23 HxVectorR2 HxSampledBSplineCurve::dC (int *order*, int *j*) const [inline]

Get curve derivative at sample j.

```
333 {
334     if ( j < 0 || j >= nSamples() ) {
335         message("(C) invalid j - setting to 0");
336         j = 0;
337     }
338     return _curve.dC(order, _t[j]);
339 }
```

6.280.3.24 vector< HxVectorR2 > HxSampledBSplineCurve::dCAll (int *order*) const

Get curve derivative at all samples.

```
225 {
226     vector<HxVectorR2> tmp(nSamples());
227     for ( int j=0; j < nSamples(); j++)
228         tmp[j] = _curve.dC(order, _t[j]);
229     return tmp;
230 }
```

6.280.3.25 double HxSampledBSplineCurve::kAtC (int *j*) const [inline]

Get curvature at sample j.

```
343 {
344     if ( j < 0 || j >= nSamples() ) {
345         message("(kAtC) invalid j - setting to 0");
346         j = 0;
347     }
348     return _curve.kAtC(_t[j]);
349 }
```

6.280.3.26 vector< double > HxSampledBSplineCurve::kAtCAll () const

Get curve derivative at all samples.

```
238 {
239     vector<double> tmp(nSamples());
240     for ( int j=0; j < nSamples(); j++)
241         tmp[j] = _curve.kAtC(_t[j]);
242     return tmp;
243 }
```

6.280.3.27 double HxSampledBSplineCurve::dTurnAngleAtC (int *j*) const [inline]

Derivative of turning angle at sample j.

```
353 {
354     if ( j < 0 || j >= nSamples() ) {
355         message("(dTurnAngleAtC) invalid j - setting to 0");
356         j = 0;
357     }
358     return _curve.dTurnAngleAtC(_t[j]);
359 }
```

6.280.3.28 vector< double > HxSampledBSplineCurve::dTurnAngleAtCAll () const

Derivative of turning angle at all samples.

```
252 {
253     vector<double> tmp(nSamples());
254     for ( int j=0; j < nSamples(); j++)
255         tmp[j] = _curve.dTurnAngleAtC(_t[j]);
256     return tmp;
257 }
```

6.280.3.29 double HxSampledBSplineCurve::length () const [inline]

total curve length.

```
363 {
364     return _curve.length(nSamples());
365 }
```

6.280.3.30 double HxSampledBSplineCurve::length (int *j1*, int *j2*) const [inline]

length of interval between given samples.

```
369 {
370     return length( HxSampledBSplineInterval(j1, j2,
371             nSamples(), curveType()) );
372 }
```

6.280.3.31 double HxSampledBSplineCurve::length (const HxSampledBSplineInterval & *interval*) const

length of given interval.

```
266 {
267     HxBSplineInterval tmp(_t[interval.begin()], _t[interval.end()]+EPS,
268         _curve.minT(), _curve.maxT(), curveType());
269
270     return _curve.length(tmp, interval.size());
271 }
```

6.280.3.32 int HxSampledBSplineCurve::closestSample (const HxPointR2 & *p*) const

Get index of sample that is closest to the given point.

```
280 {
281     vector<HxPointR2> s = AllC();
282     int iMin = 0;
283     double dMin = HxVectorR2(p,s[iMin]).magnitude();
284     for ( int i=1; i < s.size(); i++ ){
285         HxVectorR2 v(p,s[i]);
286         double d = v.magnitude();
287         if ( d < dMin ) {
288             dMin = d;
289             iMin = i;
290         }
291     }
292 
293     return iMin;
294 }
```

6.280.3.33 int HxSampledBSplineCurve::numP () const [inline]

Get number of control points.

```
237 {
238     return _curve.numP();
239 }
```

6.280.3.34 HxPointSetR2 HxSampledBSplineCurve::allP () const [inline]

Get all control points.

```
243 {
244     return _curve.allP();
245 }
```

6.280.3.35 HxPolyline2d HxSampledBSplineCurve::controlP () const [inline]

Get the control polygon.

```
249 {
250     return HxPolyline2d(allP(), (curveType() == closed));
251 }
```

6.280.3.36 HxSampledBSplineCurve HxSampledBSplineCurve::changeAllP (const HxPointSetR2 & *p*) const [inline]

Replace all control points by given points.

```
279 {
280     HxSampledBSplineCurve tmp = *this;
281     tmp._curve = _curve.changeAllP(p);
282     return tmp;
283 }
```

6.280.3.37 HxSampledBSplineCurve HxSampledBSplineCurve::translateCurve (const HxVectorR2 & *translVec*, const HxSampledBSplineInterval & *interval*) const

Add vector to control points that affect the curve in the given interval.

This will translate a part of the curve in the given direction.

```
305 {
306     HxVectorR2 v = translVec;
307     HxPointSetR2 P = _curve.allP();
308     HxBSplineInterval tmp(sampledT(interval.begin()), sampledT(interval.end()),
309         _curve.minT(), _curve.maxT(), curveType());
310     vector<int> index = _curve.PThatAffectCAt(tmp);
311     for ( int i =0; i < index.size(); i++) {
312         int j = index[i];
313         P[j] =  P[j].add(v);
314     }
315
316     return changeAllP(P);
317 }
```

6.280.3.38 STD_OSTREAM & HxSampledBSplineCurve::dump (STD_OSTREAM & *os*) const

Dump the curve on the given stream.

```
325 {
326     _curve.dump(os);
327
328     os << "Sampling Algorithm: " << samplingAlg();
329
330     if ( _t.empty() ) {
331         os << "\nNo Samples";
332     } else {
333         os << "\nT Vector: " << _t.size() << " samples\n";
334         for ( int i = 0; i < _t.size(); i++ ) {
335             os << _t[i] << ",";
336         }
337     }
338
339     os << STD_ENDL;
340     return os;
341 }
```

The documentation for this class was generated from the following files:

- **HxSampledBSplineCurve.h**
- HxSampledBSplineCurve.c

6.281 HxSampledBSplineInterval Class Reference

Class definition for path interval in sampled curves (oriented).

`#include <HxSampledBSplineInterval.h>`

Public Methods

- **HxSampledBSplineInterval** ()
  Default constructor.
- **HxSampledBSplineInterval** (int b, int e, int max, HxBSplineType type)
  Construct given interval.
- **~HxSampledBSplineInterval** ()
  Destructor.
- int **begin** () const
  First sample.
- int **end** () const
  Last sample.
- int **next** (int i) const
  Next sample index in the interval (consider wrapping).
- int **contains** (int i) const
  Determine if the sample is inside the interval.
- int **middle** () const
  central sample in interval.
- int **ratio** (double r) const
  Index of sample inside the interval with given distance from begin.
- int **size** () const
  number of samples in the interval.

6.281.1 Detailed Description

Class definition for path interval in sampled curves (oriented).

Interval = [first, second]. Values correspond to indices, and not to the path parameters. Possible interval: [0,max]. For open paths, first < second (always). For closed paths, first > second indicates wrap around 0.

6.281.2 Constructor & Destructor Documentation

6.281.2.1 HxSampledBSplineInterval::HxSampledBSplineInterval () [inline]

Default constructor.

```
71                                                  : pair<int,int>()
72 {
73     _max = 0;
74     _wrap = 0;
75 }
```

6.281.2.2 HxSampledBSplineInterval::HxSampledBSplineInterval (int *b*, int *e*, int *max*, HxBSplineType *type*) [inline]

Construct given interval.

```
79                                  : pair<int,int>(b,e)
80 {
81     _max = max;
82     _wrap = (type == closed);
83 }
```

6.281.2.3 HxSampledBSplineInterval::~HxSampledBSplineInterval () [inline]

Destructor.

```
87 {
88 }
```

6.281.3 Member Function Documentation

6.281.3.1 int HxSampledBSplineInterval::begin () const [inline]

First sample.

```
92 {
93     return first;
94 }
```

6.281.3.2 int HxSampledBSplineInterval::end () const [inline]

Last sample.

```
98 {
99     return second;
100 }
```

6.281.3.3 int HxSampledBSplineInterval::next (int *i*) const [inline]

Next sample index in the interval (consider wrapping).

```
104 {
105     if ( _wrap ) {
106         if ( i >= _max )    // is wrapped?
107             return _max-i;
108         else    return i+1;
109     } else  if ( i >= _max )
110             return _max-1;
111         else    return i+1;
112 }
```

6.281.3.4 int HxSampledBSplineInterval::contains (int *i*) const [inline]

Determine if the sample is inside the interval.

Consider wrapping for closed curves.

```
116 {
117     if ( first == second ) // special case?
118         return (i >= 0 && i < _max);
119     if ( first > second )   // is wrapped?
120         return (i >= first && i < _max) || (i >= 0 && i <= second);
121     else    return (i >= first && i <= second);
122 }
```

6.281.3.5 int HxSampledBSplineInterval::middle () const [inline]

central sample in interval.

```
126 {
127     return ratio(0.5);
128 }
```

6.281.3.6 int HxSampledBSplineInterval::ratio (double *r*) const [inline]

Index of sample inside the interval with given distance from begin.

```
132 {
133     int index = size() * r + first;
134     if ( index >= _max )
135         index -= _max;
136     return index;
137 }
```

6.281.3.7 int HxSampledBSplineInterval::size () const [inline]

number of samples in the interval.

```
142 {
143     if ( first <= second ) {
144         return second - first + 1;
145     } else  {
146         int count = 0;
147         for ( int i=first; i != second; i = next(i) )
148             count++;
149         return count+1;
150     }
151 }
```

The documentation for this class was generated from the following file:

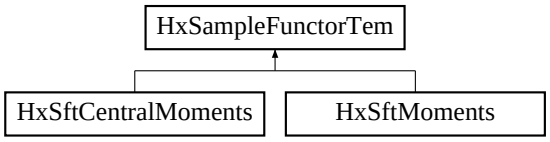
- **HxSampledBSplineInterval.h**

6.282 HxSampleFunctorTem Class Template Reference

Class definition for sample functor.

`#include <HxSampleFunctorTem.h>`

Inheritance diagram for HxSampleFunctorTem::



Public Methods

- **HxSampleFunctorTem** (**HxString** name)

  Constructor.

- virtual ~**HxSampleFunctorTem** ()
- virtual void **init** ()=0

  Initialize the functor.

- virtual void **next** (ArgType pixV, ArgType maskV, **HxPoint** p)=0

  Process the next element.

- virtual void **result** (**HxValueListBackInserter** res)=0

  Produce result (s).

- virtual int **hasPhase2** ()

  Does functor have a second phase?.

- virtual void **init2** ()

  Initialize phase 2.

- virtual void **next2** (ArgType pixV, ArgType maskV, **HxPoint** p)

  Process the next element in phase 2.

- virtual void **result2** (**HxValueListBackInserter** res)

  Produce result(s) phase 2.

6.282.1 Detailed Description

template<class ArgType, class ResType> class HxSampleFunctorTem< ArgType, ResType >

Class definition for sample functor.

6.282.2 Constructor & Destructor Documentation

6.282.2.1 template<class ArgType, class ResType> HxSampleFunctorTem< ArgType, ResType >::HxSampleFunctorTem (HxString *name*)

Constructor.

```
20 {
21     HxSampleFunctorTable<ArgType, ResType>::instance()->insert(name, this);
22     _name = name;
23 }
```

6.282.3 Member Function Documentation

6.282.3.1 template<class ArgType, class ResType> virtual void HxSampleFunctorTem< ArgType, ResType >::init () [pure virtual]

Initialize the functor.

6.282.3.2 template<class ArgType, class ResType> virtual void HxSampleFunctorTem< ArgType, ResType >::next (ArgType *pixV*, ArgType *maskV*, HxPoint *pnt*) [pure virtual]

Process the next element.

6.282.3.3 template<class ArgType, class ResType> virtual void HxSampleFunctorTem< ArgType, ResType >::result (HxValueListBackInserter *res*) [pure virtual]

Produce result (s).

6.282.3.4 template<class ArgType, class ResType> int HxSampleFunctorTem< ArgType, ResType >::hasPhase2 () [virtual]

Does functor have a second phase?.

Default: 0.

```
39 {
40     return 0;
41 }
```

6.282.3.5 template<class ArgType, class ResType> void HxSampleFunctorTem< ArgType, ResType >::init2 () [virtual]

Initialize phase 2.

Default: ignore.

```
46 {
47 }
```

6.282.3.6 template<class ArgType, class ResType> void HxSampleFunctorTem< ArgType, ResType >::next2 (ArgType *pixV*, ArgType *maskV*, HxPoint *pnt*) [virtual]

Process the next element in phase 2.

Default: ignore.

```
52 {
53 }
```

6.282.3.7 template<class ArgType, class ResType> void HxSampleFunctorTem< ArgType, ResType >::result2 (HxValueListBackInserter *res*) [virtual]

Produce result(s) phase 2.

Default: none.

```
58 {
59 }
```

The documentation for this class was generated from the following files:

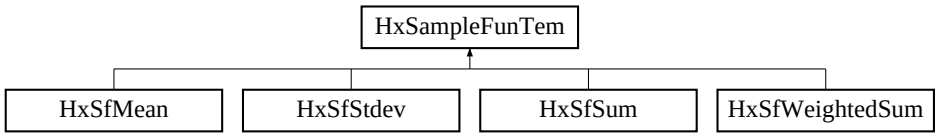
- **HxSampleFunctorTem.h**
- HxSampleFunctorTem.c

6.283 HxSampleFunTem Class Template Reference

Class definition for sample function.

`#include <HxSampleFunTem.h>`

Inheritance diagram for HxSampleFunTem::



Public Methods

- **HxSampleFunTem** (**HxString**)
  Constructor.
- virtual void **init** ()=0
  Initialize the functor.
- virtual void **next** (ArgType pixV, ArgType maskV, **HxPoint** p)=0
  Process the next element.
- virtual ResType **result** ()=0
  Produce a result.

6.283.1 Detailed Description

template<class ArgType, class ResType> class HxSampleFunTem< ArgType, ResType >

Class definition for sample function.

6.283.2 Constructor & Destructor Documentation

6.283.2.1 template<class ArgType, class ResType> HxSampleFunTem< ArgType, ResType >::HxSampleFunTem (HxString *name*)

Constructor.

```
20 {
21     HxSampleFunTable<ArgType, ResType>::instance()->insert(name, this);
22 }
```

6.283.3 Member Function Documentation

6.283.3.1 template<class ArgType, class ResType> virtual void HxSampleFunTem< ArgType, ResType >::init () [pure virtual]

Initialize the functor.

6.283.3.2 template<class ArgType, class ResType> virtual void HxSampleFunTem< ArgType, ResType >::next (ArgType *pixV*, ArgType *maskV*, HxPoint *p*) [pure virtual]

Process the next element.

6.283.3.3 template<class ArgType, class ResType> virtual ResType HxSampleFunTem< ArgType, ResType >::result () [pure virtual]

Produce a result.

The documentation for this class was generated from the following files:

- **HxSampleFunTem.h**
- HxSampleFunTem.c

6.284 HxScalarDouble Class Reference

Class definition scalar double.

`#include <HxScalarDouble.h>`

Constructors

- **HxScalarDouble** ()

  Default constructor.

- **HxScalarDouble** (double v)
  Conversion from native type.
- **HxScalarDouble** (const HxScalarDouble &rhs)
  Copy constructor.

Inquiry

- int **dim** () const
  Dimensionality.
- double **x** () const
  Value of (first) element.
- double **getValue** (int dimension) const
  Element in given dimension.
- void **setValue** (int dimension, double value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 785).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 766).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 824).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 804).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxScalarDouble &v) const
  Equal.

- int **operator!=** (const HxScalarDouble &v) const
  Not equal.
- int **operator<** (const HxScalarDouble &v) const
  Less than.
- int **operator<=** (const HxScalarDouble &v) const
  Less equal.
- int **operator>** (const HxScalarDouble &v) const
  Greater than.
- int **operator>=** (const HxScalarDouble &v) const
  Greater equal.
- const HxScalarDouble **SMALL_VAL** = -1e300
  A small value w.r.t to the comparison operators "<" and ">".
- const HxScalarDouble **LARGE_VAL** = 1e300
  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxScalarDouble **operator-** () const
  Negation.
- HxScalarDouble **complement** () const
  Complement.
- HxScalarDouble **abs** () const
  Absolute value.
- HxScalarDouble **ceil** () const
  Ceiling.
- HxScalarDouble **floor** () const
  Floor.
- HxScalarDouble **round** () const
  Round.
- HxScalarDouble **sum** () const
  Sum.
- HxScalarDouble **product** () const
  Product.

- HxScalarDouble **min** () const
  Minimum.
- HxScalarDouble **max** () const
  Maximum.
- HxScalarDouble **norm1** () const
  L1 norm.
- HxScalarDouble **norm2** () const
  L2 norm.
- HxScalarDouble **normInf** () const
  L infinity norm.
- HxScalarDouble **sqrt** () const
  Square root.
- HxScalarDouble **sin** () const
  Sine.
- HxScalarDouble **cos** () const
  Cosine.
- HxScalarDouble **tan** () const
  Tangent.
- HxScalarDouble **asin** () const
  Arc sine.
- HxScalarDouble **acos** () const
  Arc cosine.
- HxScalarDouble **atan** () const
  Arc tangent.
- HxScalarDouble **atan2** () const
  Arc tangent.
- HxScalarDouble **sinh** () const
  Hyperbolic sine.
- HxScalarDouble **cosh** () const
  Hyperbolic cosine.
- HxScalarDouble **tanh** () const
  Hyperbolic tangent.
- HxScalarDouble **exp** () const

Exponent.

- HxScalarDouble **log** () const
  Natural logarithm.
- HxScalarDouble **log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxScalarDouble & **operator+=** (const HxScalarDouble &v)
  Addition and assignment.
- HxScalarDouble & **operator-=** (const HxScalarDouble &v)
  Subtraction and assignment.
- HxScalarDouble & **operator** *=** (const HxScalarDouble &v)
  Multiplication and assignment.
- HxScalarDouble & **operator/=** (const HxScalarDouble &v)
  Division and assignment.
- HxScalarDouble **min** (const HxScalarDouble &v) const
  Minimum.
- HxScalarDouble & **minAssign** (const HxScalarDouble &v)
  Minimum and assignment.
- HxScalarDouble **max** (const HxScalarDouble &v) const
  Maximum.
- HxScalarDouble & **maxAssign** (const HxScalarDouble &v)
  Maximum and assignment.
- HxScalarDouble **inf** (const HxScalarDouble &v) const
  Infimum.
- HxScalarDouble & **infAssign** (const HxScalarDouble &v)
  Infimum and assignment.
- HxScalarDouble **sup** (const HxScalarDouble &v) const
  Supremum.
- HxScalarDouble & **supAssign** (const HxScalarDouble &v)
  Supremum and assignment.
- HxScalarDouble **pow** (const HxScalarDouble &v) const

Power.

- HxScalarDouble **mod** (const HxScalarDouble &v) const

  Modulo.

- HxScalarDouble **and** (const HxScalarDouble &v) const

  And.

- HxScalarDouble **or** (const HxScalarDouble &v) const

  Or.

- HxScalarDouble **xor** (const HxScalarDouble &v) const

  Xor.

- HxScalarDouble **leftShift** (const HxScalarDouble &v) const

  Left shift.

- HxScalarDouble **rightShift** (const HxScalarDouble &v) const

  Right shift.

- HxScalarDouble **dot** (const HxScalarDouble &v) const

  Dot product.

- HxScalarDouble **cross** (const HxScalarDouble &v) const

  Cross product.

- HxScalarDouble **operator+** (const HxScalarDouble &v1, const HxScalarDouble &v2)

  Addition.

- HxScalarDouble **operator-** (const HxScalarDouble &v1, const HxScalarDouble &v2)

  Subtraction.

- HxScalarDouble **operator** * (const HxScalarDouble &v1, const HxScalarDouble &v2)

  Multiplication.

- HxScalarDouble **operator/** (const HxScalarDouble &v1, const HxScalarDouble &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void * **operator new** (size_t, void *=0)
- HxScalarDouble & **operator=** (const double &v)
- HxScalarDouble & **operator=** (const HxScalarDouble &rhs)

6.284.1 Detailed Description

Class definition scalar double.

6.284.2 Constructor & Destructor Documentation

6.284.2.1 HxScalarDouble::HxScalarDouble () [inline]

Default constructor.

```
320 {
321 }
```

6.284.2.2 HxScalarDouble::HxScalarDouble (double *v*) [inline]

Conversion from native type.

```
325     : _value(v)
326 {
327 }
```

6.284.2.3 HxScalarDouble::HxScalarDouble (const HxScalarDouble & *rhs*) [inline]

Copy constructor.

```
331     : _value(rhs._value)
332 {
333 }
```

6.284.3 Member Function Documentation

6.284.3.1 int HxScalarDouble::dim () const [inline]

Dimensionality.

```
357 {
358     return 1;
359 }
```

6.284.3.2 double HxScalarDouble::x () const [inline]

Value of (first) element.

```
363 {
364     return _value;
365 }
```

6.284.3.3 double HxScalarDouble::getValue (int *dimension*) const [inline]

Element in given dimension.

```
369 {
370     return _value;
371 }
```

6.284.3.4 HxScalarDouble::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
26 {
27     return int(_value);
28 }
```

6.284.3.5 HxScalarDouble::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
31 {
32     return HxVec2Int(int(_value), int(_value));
33 }
```

6.284.3.6 HxScalarDouble::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
36 {
37     return HxVec2Double(_value, _value);
38 }
```

6.284.3.7 HxScalarDouble::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
41 {
42     return HxVec3Int(int(_value), int(_value), int(_value));
43 }
```

6.284.3.8 HxScalarDouble::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
46 {
47     return HxVec3Double(_value, _value, _value);
48 }
```

6.284.3.9 HxScalarDouble::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
51 {
52     return HxComplex(_value, 0.0);
53 }
```

6.284.3.10 int HxScalarDouble::operator== (const HxScalarDouble & *v*) const [inline]

Equal.

```
387 {
388     return (_value == v._value);
389 }
```

6.284.3.11 int HxScalarDouble::operator!= (const HxScalarDouble & *v*) const [inline]

Not equal.

```
393 {
394     return (_value != v._value);
395 }
```

6.284.3.12 int HxScalarDouble::operator< (const HxScalarDouble & *v*) const [inline]

Less than.

```
399 {
400     return (_value < v._value);
401 }
```

6.284.3.13 int HxScalarDouble::operator<= (const HxScalarDouble & *v*) const [inline]

Less equal.

```
405 {
406     return (_value <= v._value);
407 }
```

6.284.3.14 int HxScalarDouble::operator> (const HxScalarDouble & *v*) const [inline]

Greater than.

```
411 {
412     return (_value > v._value);
413 }
```

6.284.3.15 int HxScalarDouble::operator>= (const HxScalarDouble & *v*) const [inline]

Greater equal.

```
417 {
418     return (_value >= v._value);
419 }
```

6.284.3.16 HxScalarDouble HxScalarDouble::operator- () const [inline]

Negation.

```
423 {
424     return HxScalarDouble(-_value);
425 }
```

6.284.3.17 HxScalarDouble HxScalarDouble::complement () const [inline]

Complement.

```
429 {
430     return HxScalarDouble(-_value);
431 }
```

6.284.3.18 HxScalarDouble HxScalarDouble::abs () const [inline]

Absolute value.

```
435 {
436     return HxScalarDouble(fabs(_value));
437 }
```

6.284.3.19 HxScalarDouble HxScalarDouble::ceil () const [inline]

Ceiling.

```
441 {
442     return HxScalarDouble(::ceil(_value));
443 }
```

6.284.3.20 HxScalarDouble HxScalarDouble::floor () const [inline]

Floor.

```
447 {
448     return HxScalarDouble(::floor(_value));
449 }
```

6.284.3.21 HxScalarDouble HxScalarDouble::round () const [inline]

Round.

```
453 {
454     return HxScalarDouble((int) (_value + ((_value >= 0) ? 0.5 : -0.5)));
455 }
```

6.284.3.22 HxScalarDouble HxScalarDouble::sum () const [inline]

Sum.

```
459 {
460     return *this;
461 }
```

6.284.3.23 HxScalarDouble HxScalarDouble::product () const [inline]

Product.

```
465 {
466     return *this;
467 }
```

6.284.3.24 HxScalarDouble HxScalarDouble::min () const [inline]

Minimum.

```
471 {
472     return *this;
473 }
```

6.284.3.25 HxScalarDouble HxScalarDouble::max () const [inline]

Maximum.

```
477 {
478     return *this;
479 }
```

6.284.3.26 HxScalarDouble HxScalarDouble::norm1 () const [inline]

L1 norm.

```
483 {
484     return HxScalarDouble(fabs(_value));
485 }
```

6.284.3.27 HxScalarDouble HxScalarDouble::norm2 () const

L2 norm.

```
57 {
58     return HxScalarDouble(fabs(_value));
59 }
```

6.284.3.28 HxScalarDouble HxScalarDouble::normInf () const [inline]

L infinity norm.

```
489 {
490     return HxScalarDouble(fabs(_value));
491 }
```

6.284.3.29 HxScalarDouble HxScalarDouble::sqrt () const [inline]

Square root.

```
495 {
496     return HxScalarDouble(::sqrt(_value));
497 }
```

6.284.3.30 HxScalarDouble HxScalarDouble::sin () const [inline]

Sine.

```
501 {
502     return HxScalarDouble(::sin(_value));
503 }
```

6.284.3.31 HxScalarDouble HxScalarDouble::cos () const [inline]

Cosine.

```
507 {
508     return HxScalarDouble(::cos(_value));
509 }
```

6.284.3.32 HxScalarDouble HxScalarDouble::tan () const [inline]

Tangent.

```
513 {
514     return HxScalarDouble(::tan(_value));
515 }
```

6.284.3.33 HxScalarDouble HxScalarDouble::asin () const [inline]

Arc sine.

```
519 {
520     return HxScalarDouble(::asin(_value));
521 }
```

6.284.3.34 HxScalarDouble HxScalarDouble::acos () const [inline]

Arc cosine.

```
525 {
526     return HxScalarDouble(::acos(_value));
527 }
```

6.284.3.35 HxScalarDouble HxScalarDouble::atan () const [inline]

Arc tangent.

```
531 {
532     return HxScalarDouble(::atan(_value));
533 }
```

6.284.3.36 HxScalarDouble HxScalarDouble::atan2 () const

Arc tangent.

```
63 {
64     return HxScalarDouble(::atan(_value));
65 }
```

6.284.3.37 HxScalarDouble HxScalarDouble::sinh () const [inline]

Hyperbolic sine.

```
537 {
538     return HxScalarDouble(::sinh(_value));
539 }
```

6.284.3.38 HxScalarDouble HxScalarDouble::cosh () const [inline]

Hyperbolic cosine.

```
543 {
544     return HxScalarDouble(::cosh(_value));
545 }
```

6.284.3.39 HxScalarDouble HxScalarDouble::tanh () const [inline]

Hyperbolic tangent.

```
549 {
550     return HxScalarDouble(::tanh(_value));
551 }
```

6.284.3.40 HxScalarDouble HxScalarDouble::exp () const [inline]

Exponent.

```
555 {
556     return HxScalarDouble(::exp(_value));
557 }
```

6.284.3.41 HxScalarDouble HxScalarDouble::log () const [inline]

Natural logarithm.

```
561 {
562     return HxScalarDouble(::log(_value));
563 }
```

6.284.3.42 HxScalarDouble HxScalarDouble::log10 () const [inline]

Base 10 logarithm.

```
567 {
568     return HxScalarDouble(::log10(_value));
569 }
```

6.284.3.43 HxScalarDouble & HxScalarDouble::operator+= (const HxScalarDouble & *v*) [inline]

Addition and assignment.

```
573 {
574     _value += v._value;
575     return *this;
576 }
```

6.284.3.44 HxScalarDouble & HxScalarDouble::operator-= (const HxScalarDouble & *v*) [inline]

Subtraction and assignment.

```
580 {
581     _value -= v._value;
582     return *this;
583 }
```

6.284.3.45 HxScalarDouble & HxScalarDouble::operator *= (const HxScalarDouble & *v*) [inline]

Multiplication and assignment.

```
587 {
588     _value *= v._value;
589     return *this;
590 }
```

6.284.3.46 HxScalarDouble & HxScalarDouble::operator/= (const HxScalarDouble & *v*) [inline]

Division and assignment.

```
594 {
595     _value /= v._value;
596     return *this;
597 }
```

6.284.3.47 HxScalarDouble HxScalarDouble::min (const HxScalarDouble & *v*) const [inline]

Minimum.

```
625 {
626     return (operator<(v)) ? (*this) : v;
627 }
```

6.284.3.48 HxScalarDouble & HxScalarDouble::minAssign (const HxScalarDouble & *v*) [inline]

Minimum and assignment.

```
631 {
632     if (operator<(v))
633         return *this;
634     operator=(v);
635     return *this;
636 }
```

6.284.3.49 HxScalarDouble HxScalarDouble::max (const HxScalarDouble & *v*) const [inline]

Maximum.

```
640 {
641     return (operator>(v)) ? (*this) : v;
642 }
```

6.284.3.50 HxScalarDouble & HxScalarDouble::maxAssign (const HxScalarDouble & *v*) [inline]

Maximum and assignment.

```
646 {
647     if (operator>(v))
648         return *this;
649     operator=(v);
650     return *this;
651 }
```

6.284.3.51 HxScalarDouble HxScalarDouble::inf (const HxScalarDouble & *v*) const [inline]

Infimum.

```
655 {
656     return (operator<(v)) ? (*this) : v;
657 }
```

6.284.3.52 HxScalarDouble & HxScalarDouble::infAssign (const HxScalarDouble & *v*) [inline]

Infimum and assignment.

```
661 {
662     _value = (_value < v._value) ? _value : v._value;
663     return *this;
664 }
```

6.284.3.53 HxScalarDouble HxScalarDouble::sup (const HxScalarDouble & *v*) const [inline]

Supremum.

```
668 {
669     return (operator>(v)) ? (*this) : v;
670 }
```

6.284.3.54 HxScalarDouble & HxScalarDouble::supAssign (const HxScalarDouble & *v*) [inline]

Supremum and assignment.

```
674 {
675     _value = (_value > v._value) ? _value : v._value;
676     return *this;
677 }
```

6.284.3.55 HxScalarDouble HxScalarDouble::pow (const HxScalarDouble & *v*) const [inline]

Power.

```
681 {
682     return HxScalarDouble(::pow(_value, v._value));
683 }
```

6.284.3.56 HxScalarDouble HxScalarDouble::mod (const HxScalarDouble & *v*) const [inline]

Modulo.

```
687 {
688     return (*this);
689 }
```

6.284.3.57 HxScalarDouble HxScalarDouble::and (const HxScalarDouble & *v*) const [inline]

And.

```
693 {
694     return (*this);
695 }
```

6.284.3.58 HxScalarDouble HxScalarDouble::or (const HxScalarDouble & *v*) const [inline]

Or.

```
699 {
700     return (*this);
701 }
```

6.284.3.59 HxScalarDouble HxScalarDouble::xor (const HxScalarDouble & *v*) const [inline]

Xor.

```
705 {
706     return (*this);
707 }
```

6.284.3.60 HxScalarDouble HxScalarDouble::leftShift (const HxScalarDouble & *v*) const [inline]

Left shift.

```
711 {
712     return (*this);
713 }
```

6.284.3.61 HxScalarDouble HxScalarDouble::rightShift (const HxScalarDouble & *v*) const [inline]

Right shift.

```
717 {
718     return (*this);
719 }
```

6.284.3.62 HxScalarDouble HxScalarDouble::dot (const HxScalarDouble & *v*) const

Dot product.

```
69 {
70     return _value * v._value;
71 }
```

6.284.3.63 HxScalarDouble HxScalarDouble::cross (const HxScalarDouble & *v*) const [inline]

Cross product.

```
723 {
724     return HxScalarDouble(0.0);
725 }
```

6.284.3.64 STD_OSTREAM & HxScalarDouble::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
75 {
76     return os << _value;
77 }
```

6.284.3.65 HxString HxScalarDouble::toString () const [inline]

Value as a string.

```
729 {
730     return makeString(_value);
731 }
```

6.284.4 Friends And Related Function Documentation

6.284.4.1 HxScalarDouble operator+ (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Addition.

```
601 {
602     return HxScalarDouble(v1._value + v2._value);
603 }
```

6.284.4.2 HxScalarDouble operator- (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Subtraction.

```
607 {
608     return HxScalarDouble(v1._value - v2._value);
609 }
```

6.284.4.3 HxScalarDouble operator * (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Multiplication.

```
613 {
614     return HxScalarDouble(v1._value * v2._value);
615 }
```

6.284.4.4 HxScalarDouble operator/ (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Division.

```
619 {
620     return HxScalarDouble(v1._value / v2._value);
621 }
```

6.284.5 Member Data Documentation

6.284.5.1 const HxScalarDouble HxScalarDouble::SMALL_VAL = -1e300 [static]

A small value w.r.t to the comparison operators ”<” and ”>”.

Not actually the minimum to avoid overflow.

6.284.5.2 const HxScalarDouble HxScalarDouble::LARGE_VAL = 1e300 [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxScalarDouble.h**
- HxScalarDouble.c

6.285 HxScalarInt Class Reference

Class definition scalar integer.

`#include <HxScalarInt.h>`

Constructors

- **HxScalarInt** ()

  Default constructor.

- **HxScalarInt** (int v)

  Conversion from native type.

- **HxScalarInt** (const HxScalarInt &v)

  Copy constructor.

Inquiry

- int **dim** () const

  Dimensionality.

- int **x** () const

  Value of (first) element.

- int **getValue** (int dimension) const

  Element in given dimension.

- void **setValue** (int dimension, int value)

Conversion

- **operator HxScalarDouble** () const

  Cast to **HxScalarDouble** (p. 677).

- **operator HxVec2Int** () const

  Cast to **HxVec2Int** (p. 785).

- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 766).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 824).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 804).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxScalarInt &v) const
  Equal.
- int **operator!=** (const HxScalarInt &v) const
  Not equal.
- int **operator<** (const HxScalarInt &v) const
  Less than.
- int **operator<=** (const HxScalarInt &v) const
  Less equal.
- int **operator>** (const HxScalarInt &v) const
  Greater than.
- int **operator>=** (const HxScalarInt &v) const
  Greater equal.
- const HxScalarInt **SMALL_VAL** = -200000000
  A small value w.r.t to the comparison operators "<" and ">".
- const HxScalarInt **LARGE_VAL** = 200000000
  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxScalarInt **operator-** () const
  Negation.

- HxScalarInt **complement** () const
  Complement.
- HxScalarInt **abs** () const
  Absolute value.
- HxScalarInt **ceil** () const
  Ceiling.
- HxScalarInt **floor** () const
  Floor.
- HxScalarInt **round** () const
  Round.
- HxScalarInt **sum** () const
  Sum.
- HxScalarInt **product** () const
  Product.
- HxScalarInt **min** () const
  Minimum.
- HxScalarInt **max** () const
  Maximum.
- HxScalarInt **norm1** () const
  L1 norm.
- **HxScalarDouble norm2** () const
  L2 norm.
- HxScalarInt **normInf** () const
  L infinity norm.
- **HxScalarDouble sqrt** () const
  Square root.
- **HxScalarDouble sin** () const
  Sine.
- **HxScalarDouble cos** () const
  Cosine.
- **HxScalarDouble tan** () const
  Tangent.
- **HxScalarDouble asin** () const

Arc sine.

- **HxScalarDouble acos** () const
  Arc cosine.
- **HxScalarDouble atan** () const
  Arc tangent.
- **HxScalarDouble atan2** () const
  Arc tangent.
- **HxScalarDouble sinh** () const
  Hyperbolic sine.
- **HxScalarDouble cosh** () const
  Hyperbolic cosine.
- **HxScalarDouble tanh** () const
  Hyperbolic tangent.
- **HxScalarDouble exp** () const
  Exponent.
- **HxScalarDouble log** () const
  Natural logarithm.
- **HxScalarDouble log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxScalarInt & **operator+=** (const HxScalarInt &v)
  Addition and assignment.
- HxScalarInt & **operator-=** (const HxScalarInt &v)
  Subtraction and assignment.
- HxScalarInt & **operator** *= (const HxScalarInt &v)
  Multiplication and assignment.
- HxScalarInt & **operator/=** (const HxScalarInt &v)
  Division and assignment.
- HxScalarInt **min** (const HxScalarInt &v) const
  Minimum.
- HxScalarInt & **minAssign** (const HxScalarInt &v)

Minimum and assignment.

- HxScalarInt **max** (const HxScalarInt &v) const
  Maximum.
- HxScalarInt & **maxAssign** (const HxScalarInt &v)
  Maximum and assignment.
- HxScalarInt **inf** (const HxScalarInt &v) const
  Infimum.
- HxScalarInt & **infAssign** (const HxScalarInt &v)
  Infimum and assignment.
- HxScalarInt **sup** (const HxScalarInt &v) const
  Supremum.
- HxScalarInt & **supAssign** (const HxScalarInt &v)
  Supremum and assignment.
- HxScalarInt **pow** (const HxScalarInt &v) const
  Power.
- HxScalarInt **mod** (const HxScalarInt &v) const
  Modulo.
- HxScalarInt **and** (const HxScalarInt &v) const
  And.
- HxScalarInt **or** (const HxScalarInt &v) const
  Or.
- HxScalarInt **xor** (const HxScalarInt &v) const
  Xor.
- HxScalarInt **leftShift** (const HxScalarInt &v) const
  Left shift.
- HxScalarInt **rightShift** (const HxScalarInt &v) const
  Right shift.
- HxScalarInt **dot** (const HxScalarInt &v) const
  Dot product.
- HxScalarInt **cross** (const HxScalarInt &v) const
  Cross product.
- HxScalarInt **operator+** (const HxScalarInt &v1, const HxScalarInt &v2)
  Addition.

- HxScalarInt **operator-** (const HxScalarInt &v1, const HxScalarInt &v2)

  Subtraction.

- HxScalarInt **operator** ∗ (const HxScalarInt &v1, const HxScalarInt &v2)

  Multiplication.

- HxScalarInt **operator/** (const HxScalarInt &v1, const HxScalarInt &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void ∗ **operator new** (size_t, void ∗=0)

6.285.1 Detailed Description

Class definition scalar integer.

6.285.2 Constructor & Destructor Documentation

6.285.2.1 HxScalarInt::HxScalarInt () [inline]

Default constructor.

```
317 {
318 }
```

6.285.2.2 HxScalarInt::HxScalarInt (int *v*) [inline]

Conversion from native type.

```
322 {
323     _value = v;
324 }
```

6.285.2.3 HxScalarInt::HxScalarInt (const HxScalarInt & *v*) [inline]

Copy constructor.

```
328 {
329     _value = v._value;
330 }
```

6.285.3 Member Function Documentation

6.285.3.1 int HxScalarInt::dim () const [inline]

Dimensionality.

```
340 {
341     return 1;
342 }
```

6.285.3.2 int HxScalarInt::x () const [inline]

Value of (first) element.

```
346 {
347     return _value;
348 }
```

6.285.3.3 int HxScalarInt::getValue (int *dimension*) const [inline]

Element in given dimension.

```
352 {
353     return _value;
354 }
```

6.285.3.4 HxScalarInt::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
26 {
27     return HxScalarDouble(_value);
28 }
```

6.285.3.5 HxScalarInt::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
31 {
32     return HxVec2Int(_value, _value);
33 }
```

6.285.3.6 HxScalarInt::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
36 {
37     return HxVec2Double(_value, _value);
38 }
```

6.285.3.7 HxScalarInt::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
41 {
42     return HxVec3Int(_value, _value, _value);
43 }
```

6.285.3.8 HxScalarInt::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
46 {
47     return HxVec3Double(_value, _value, _value);
48 }
```

6.285.3.9 HxScalarInt::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
51 {
52     return HxComplex(_value, 0.0);
53 }
```

6.285.3.10 int HxScalarInt::operator== (const HxScalarInt & *v*) const [inline]

Equal.

```
370 {
371     return (_value == v._value);
372 }
```

6.285.3.11 int HxScalarInt::operator!= (const HxScalarInt & *v*) const [inline]

Not equal.

```
376 {
377     return (_value != v._value);
378 }
```

6.285.3.12 int HxScalarInt::operator< (const HxScalarInt & *v*) const [inline]

Less than.

```
382 {
383     return (_value < v._value);
384 }
```

6.285.3.13 int HxScalarInt::operator<= (const HxScalarInt & *v*) const [inline]

Less equal.

```
388 {
389     return (_value <= v._value);
390 }
```

6.285.3.14 int HxScalarInt::operator> (const HxScalarInt & *v*) const [inline]

Greater than.

```
394 {
395     return (_value > v._value);
396 }
```

6.285.3.15 int HxScalarInt::operator>= (const HxScalarInt & *v*) const [inline]

Greater equal.

```
400 {
401     return (_value >= v._value);
402 }
```

6.285.3.16 HxScalarInt HxScalarInt::operator- () const [inline]

Negation.

```
406 {
407     return HxScalarInt(-_value);
408 }
```

6.285.3.17 HxScalarInt HxScalarInt::complement () const [inline]

Complement.

```
412 {
413     return HxScalarInt(~_value);
414 }
```

6.285.3.18 HxScalarInt HxScalarInt::abs () const [inline]

Absolute value.

```
418 {
419     return HxScalarInt(::abs(_value));
420 }
```

6.285.3.19 HxScalarInt HxScalarInt::ceil () const [inline]

Ceiling.

```
424 {
425     return *this;
426 }
```

6.285.3.20 HxScalarInt HxScalarInt::floor () const [inline]

Floor.

```
430 {
431     return *this;
432 }
```

6.285.3.21 HxScalarInt HxScalarInt::round () const [inline]

Round.

```
436 {
437     return *this;
438 }
```

6.285.3.22 HxScalarInt HxScalarInt::sum () const [inline]

Sum.

```
442 {
443     return *this;
444 }
```

6.285.3.23 HxScalarInt HxScalarInt::product () const [inline]

Product.

```
448 {
449     return *this;
450 }
```

6.285.3.24 HxScalarInt HxScalarInt::min () const [inline]

Minimum.

```
454 {
455     return *this;
456 }
```

6.285.3.25 HxScalarInt HxScalarInt::max () const [inline]

Maximum.

```
460 {
461     return *this;
462 }
```

6.285.3.26 HxScalarInt HxScalarInt::norm1 () const [inline]

L1 norm.

```
466 {
467     return HxScalarInt(::abs(_value));
468 }
```

6.285.3.27 HxScalarDouble HxScalarInt::norm2 () const

L2 norm.

```
57 {
58     return HxScalarDouble(::abs(_value));
59 }
```

6.285.3.28 HxScalarInt HxScalarInt::normInf () const [inline]

L infinity norm.

```
472 {
473     return HxScalarInt(::abs(_value));
474 }
```

6.285.3.29 HxScalarDouble HxScalarInt::sqrt () const

Square root.

```
63 {
64     return HxScalarDouble(::sqrt(double(_value)));
65 }
```

6.285.3.30 HxScalarDouble HxScalarInt::sin () const

Sine.

```
69 {
70     return HxScalarDouble(::sin(double(_value)));
71 }
```

6.285.3.31 HxScalarDouble HxScalarInt::cos () const

Cosine.

```
75 {
76     return HxScalarDouble(::cos(double(_value)));
77 }
```

6.285.3.32 HxScalarDouble HxScalarInt::tan () const

Tangent.

```
81 {
82     return HxScalarDouble(::tan(double(_value)));
83 }
```

6.285.3.33 HxScalarDouble HxScalarInt::asin () const

Arc sine.

```
87 {
88     return HxScalarDouble(::asin(double(_value)));
89 }
```

6.285.3.34 HxScalarDouble HxScalarInt::acos () const

Arc cosine.

```
93 {
94     return HxScalarDouble(::acos(double(_value)));
95 }
```

6.285.3.35 HxScalarDouble HxScalarInt::atan () const

Arc tangent.

```
99 {
100     return HxScalarDouble(::atan(double(_value)));
101 }
```

6.285.3.36 HxScalarDouble HxScalarInt::atan2 () const

Arc tangent.

```
105 {
106     return HxScalarDouble(::atan(double(_value)));
107 }
```

6.285.3.37 HxScalarDouble HxScalarInt::sinh () const

Hyperbolic sine.

```
111 {
112     return HxScalarDouble(::sinh(double(_value)));
113 }
```

6.285.3.38 HxScalarDouble HxScalarInt::cosh () const

Hyperbolic cosine.

```
117 {
118     return HxScalarDouble(::cosh(double(_value)));
119 }
```

6.285.3.39 HxScalarDouble HxScalarInt::tanh () const

Hyperbolic tangent.

```
123 {
124     return HxScalarDouble(::tanh(double(_value)));
125 }
```

6.285.3.40 HxScalarDouble HxScalarInt::exp () const

Exponent.

```
129 {
130     return HxScalarDouble(::exp(double(_value)));
131 }
```

6.285.3.41 HxScalarDouble HxScalarInt::log () const

Natural logarithm.

```
135 {
136     return HxScalarDouble(::log(double(_value)));
137 }
```

6.285.3.42 HxScalarDouble HxScalarInt::log10 () const

Base 10 logarithm.

```
141 {
142     return HxScalarDouble(::log10(double(_value)));
143 }
```

6.285.3.43 HxScalarInt & HxScalarInt::operator+= (const HxScalarInt & *v*) [inline]

Addition and assignment.

```
478 {
479     _value += v._value;
480     return *this;
481 }
```

6.285.3.44 HxScalarInt & HxScalarInt::operator-= (const HxScalarInt & *v*) [inline]

Subtraction and assignment.

```
485 {
486     _value -= v._value;
487     return *this;
488 }
```

6.285.3.45 HxScalarInt & HxScalarInt::operator *= (const HxScalarInt & *v*) [inline]

Multiplication and assignment.

```
492 {
493     _value *= v._value;
494     return *this;
495 }
```

6.285.3.46 HxScalarInt & HxScalarInt::operator/= (const HxScalarInt & *v*) [inline]

Division and assignment.

```
499 {
500     _value /= v._value;
501     return *this;
502 }
```

6.285.3.47 HxScalarInt HxScalarInt::min (const HxScalarInt & *v*) const [inline]

Minimum.

```
530 {
531     return (operator<(v)) ? (*this) : v;
532 }
```

6.285.3.48 HxScalarInt & HxScalarInt::minAssign (const HxScalarInt & *v*) [inline]

Minimum and assignment.

```
536 {
537     if (operator<(v))
538         return *this;
539     operator=(v);
540     return *this;
541 }
```

6.285.3.49 HxScalarInt HxScalarInt::max (const HxScalarInt & *v*) const [inline]

Maximum.

```
545 {
546     return (operator>(v)) ? (*this) : v;
547 }
```

6.285.3.50 HxScalarInt & HxScalarInt::maxAssign (const HxScalarInt & *v*) [inline]

Maximum and assignment.

```
551 {
552     if (operator>(v))
553         return *this;
554     operator=(v);
555     return *this;
556 }
```

6.285.3.51 HxScalarInt HxScalarInt::inf (const HxScalarInt & *v*) const [inline]

Infimum.

```
560 {
561     return (operator<(v)) ? (*this) : v;
562 }
```

6.285.3.52 HxScalarInt & HxScalarInt::infAssign (const HxScalarInt & *v*) [inline]

Infimum and assignment.

```
566 {
567     _value = (_value < v._value) ? _value : v._value;
568     return *this;
569 }
```

6.285.3.53 HxScalarInt HxScalarInt::sup (const HxScalarInt & *v*) const [inline]

Supremum.

```
573 {
574     return (operator>(v)) ? (*this) : v;
575 }
```

6.285.3.54 HxScalarInt & HxScalarInt::supAssign (const HxScalarInt & *v*) [inline]

Supremum and assignment.

```
579 {
580     _value = (_value > v._value) ? _value : v._value;
581     return *this;
582 }
```

6.285.3.55 HxScalarInt HxScalarInt::pow (const HxScalarInt & *v*) const [inline]

Power.

```
586 {
587     return HxScalarInt((int) (::pow(_value, v._value) + 0.5));
588 }
```

6.285.3.56 HxScalarInt HxScalarInt::mod (const HxScalarInt & *v*) const [inline]

Modulo.

```
592 {
593     return HxScalarInt(_value % v._value);
594 }
```

6.285.3.57 HxScalarInt HxScalarInt::and (const HxScalarInt & *v*) const [inline]

And.

```
598 {
599     return HxScalarInt(_value & v._value);
600 }
```

6.285.3.58 HxScalarInt HxScalarInt::or (const HxScalarInt & *v*) const [inline]

Or.

```
604 {
605     return HxScalarInt(_value | v._value);
606 }
```

6.285.3.59 HxScalarInt HxScalarInt::xor (const HxScalarInt & *v*) const [inline]

Xor.

```
610 {
611     return HxScalarInt(_value ^ v._value);
612 }
```

6.285.3.60 HxScalarInt HxScalarInt::leftShift (const HxScalarInt & *v*) const [inline]

Left shift.

```
616 {
617     return HxScalarInt(_value << v._value);
618 }
```

6.285.3.61 HxScalarInt HxScalarInt::rightShift (const HxScalarInt & *v*) const [inline]

Right shift.

```
622 {
623     return HxScalarInt(_value >> v._value);
624 }
```

6.285.3.62 HxScalarInt HxScalarInt::dot (const HxScalarInt & *v*) const

Dot product.

```
147 {
148     return _value * v._value;
149 }
```

6.285.3.63 HxScalarInt HxScalarInt::cross (const HxScalarInt & *v*) const [inline]

Cross product.

```
628 {
629     return 0;
630 }
```

6.285.3.64 STD_OSTREAM & HxScalarInt::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
153 {
154     return os << _value;
155 }
```

6.285.3.65 HxString HxScalarInt::toString () const [inline]

Value as a string.

```
634 {
635     return makeString(_value);
636 }
```

6.285.4 Friends And Related Function Documentation

6.285.4.1 HxScalarInt operator+ (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Addition.

```
506 {
507     return HxScalarInt(v1._value + v2._value);
508 }
```

6.285.4.2 HxScalarInt operator- (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Subtraction.

```
512 {
513     return HxScalarInt(v1._value - v2._value);
514 }
```

6.285.4.3 HxScalarInt operator * (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Multiplication.

```
518 {
519     return HxScalarInt(v1._value * v2._value);
520 }
```

6.285.4.4 HxScalarInt operator/ (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Division.

```
524 {
525     return HxScalarInt(v1._value / v2._value);
526 }
```

6.285.5 Member Data Documentation

6.285.5.1 const HxScalarInt HxScalarInt::SMALL_VAL = -200000000 [static]

A small value w.r.t to the comparison operators ”<” and ”>”.

Not actually the minimum to avoid overflow.

6.285.5.2 const HxScalarInt HxScalarInt::LARGE_VAL = 200000000 [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxScalarInt.h**
- HxScalarInt.c

6.286 HxStatFunctorTem Class Template Reference

Class definition for statistics functor.

`#include <HxStatFunctorTem.h>`

Public Methods

- **HxStatFunctorTem** (**HxString** name)

  Constructor.

- virtual ~**HxStatFunctorTem** ()
- virtual void **init** ()=0

  Initialize the functor.

- virtual void **next** (ArgType pixV)=0

  Process the next element.

- virtual ResType **result** ()=0

  Produce result.

6.286.1 Detailed Description

template<class ArgType, class ResType> class HxStatFunctorTem< ArgType, ResType >

Class definition for statistics functor.

6.286.2 Constructor & Destructor Documentation

6.286.2.1 template<class ArgType, class ResType> HxStatFunctorTem< ArgType, ResType >::HxStatFunctorTem (HxString *name*)

Constructor.

```
20 {
21     HxStatFunctorTable<ArgType, ResType>::instance()->insert(name, this);
22     _name = name;
23 }
```

6.286.3 Member Function Documentation

6.286.3.1 template<class ArgType, class ResType> virtual void HxStatFunctorTem< ArgType, ResType >::init () [pure virtual]

Initialize the functor.

6.286.3.2 template<class ArgType, class ResType> virtual void HxStatFunctorTem< ArgType, ResType >::next (ArgType *pixV*) [pure virtual]

Process the next element.

6.286.3.3 template<class ArgType, class ResType> virtual ResType HxStatFunctorTem< ArgType, ResType >::result () [pure virtual]

Produce result.

The documentation for this class was generated from the following files:

- **HxStatFunctorTem.h**
- HxStatFunctorTem.c

6.287 HxStringList Class Reference

Class definition for list of HxString's.

`#include <HxStringList.h>`

Public Types

- typedef std::back_insert_iterator< HxStringList > **back_insert_iterator**

  back inserter.

Public Methods

- **HxStringList** ()
- **HxStringList** (const std::list< std::string > &l)
- **HxStringList** (const std::vector< std::string > &l)
- HxStringList & **operator**<< (const **HxString** &)

  Add string to the list.

- void **eraseAll** ()

  Remove all value from the list.

6.287.1 Detailed Description

Class definition for list of HxString's.

Specialization of list from STL.

6.287.2 Member Typedef Documentation

6.287.2.1 typedef std::back_insert_iterator<HxStringList> HxStringList::back_insert_iterator

back inserter.

6.287.3 Member Function Documentation

6.287.3.1 HxStringList & HxStringList::operator<< (const HxString & *s*) [inline]

Add string to the list.

```
52 {
53     push_back(s);
54     return *this;
55 }
```

6.287.3.2 void HxStringList::eraseAll () [inline]

Remove all value from the list.

```
59 {
60     erase(begin(), end());
61 }
```

The documentation for this class was generated from the following file:

- **HxStringList.h**

6.288 HxTagList Class Reference

A list of tags.

`#include <HxTagList.h>`

Public Types

- typedef HxTag * **TagPtr**
- typedef std::list< TagPtr > **List**

Public Methods

- **HxTagList** ()

  Constructor.

- **HxTagList** (const HxTagList &)

  Copy constructor.

- ~**HxTagList** ()

  Destructor.

- HxTagList & **operator=** (const HxTagList &)

  Assignment operator.

- void **erase** ()

  Make the tag list empty.

- void **addTag** (HxTag ∗)

  Add a tag, use via **HxAddTag** (p. 156).

- HxTag ∗ **getTag** (**HxString** name) const

  Get a tag, use via **HxGetTag** (p. 156).

- **HxString toString** () const
- std::ostream & **put** (std::ostream &) const

  To put the list on an ostream.

6.288.1 Detailed Description

A list of tags.

Use **HxAddTag** (p. 156) and **HxGetTag** (p. 156) to manipulate tags in the list.

6.288.2 Constructor & Destructor Documentation

6.288.2.1 HxTagList::HxTagList () [inline]

Constructor.

```
30 {}
```

6.288.2.2 HxTagList::HxTagList (const HxTagList & *rhs*)

Copy constructor.

```
16 {
17     List::const_iterator ptr;
18     for (ptr = rhs._list.begin(); ptr != rhs._list.end(); ptr++)
```

```
19     {
20         _list.push_back((*ptr)->clone());
21     }
22 }
```

6.288.2.3 HxTagList::~HxTagList ()

Destructor.

```
39 {
40     List::iterator ptr;
41 
42     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
43         delete (*ptr);
44 }
```

6.288.3 Member Function Documentation

6.288.3.1 HxTagList & HxTagList::operator= (const HxTagList & *rhs*)

Assignment operator.

```
26 {
27     List::const_iterator ptr;
28     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
29         delete (*ptr);
30     _list.erase(_list.begin(), _list.end());
31     for (ptr = rhs._list.begin(); ptr != rhs._list.end(); ptr++)
32     {
33         _list.push_back((*ptr)->clone());
34     }
35     return *this;
36 }
```

6.288.3.2 void HxTagList::erase ()

Make the tag list empty.

```
48 {
49     List::const_iterator ptr;
50     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
51         delete (*ptr);
52     _list.erase(_list.begin(), _list.end());
53 }
```

6.288.3.3 void HxTagList::addTag (HxTag * *tag*)

Add a tag, use via **HxAddTag** (p. 156).

```
77 {
78     List::iterator ptr = findTag(tag->getName());
79 
```

```
80     if (ptr != _list.end() )
81     {
82         delete (*ptr);
83         _list.erase(ptr);
84     }
85     _list.push_front(tag);
86 }
```

6.288.3.4 HxTag * HxTagList::getTag (HxString *name*) const

Get a tag, use via **HxGetTag** (p. 156).

```
90 {
91     List::const_iterator ptr = findTag(name);
92     return ptr != _list.end() ? (*ptr) : 0;
93 }
```

6.288.3.5 std::ostream & HxTagList::put (std::ostream & *os*) const

To put the list on an ostream.

```
114 {
115     List::const_iterator ptr;
116     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
117         os << (*ptr) << STD_ENDL;
118     return os;
119 }
```

The documentation for this class was generated from the following files:

- **HxTagList.h**
- HxTagList.c

6.289 HxUpoAbs Class Template Reference

Pixel functor for computation of absolute value.

`#include <HxUpoAbs.h>`

Public Methods

- **HxUpoAbs** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.abs() #.

Static Public Methods

- **HxString className** ()

  The name : ”abs”.

6.289.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAbs< DstValT, SrcValT >

Pixel functor for computation of absolute value.

6.289.2 Constructor & Destructor Documentation

6.289.2.1 template<class DstValT, class SrcValT> HxUpoAbs< DstValT, SrcValT >::HxUpoAbs (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.289.3 Member Function Documentation

6.289.3.1 template<class DstValT, class SrcValT> DstValT HxUpoAbs< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.abs() #.

```
29                          { return x.abs(); }
```

6.289.3.2 template<class DstValT, class SrcValT> HxString HxUpoAbs< DstValT, SrcValT >::className () [inline, static]

The name : ”abs”.

```
33                          { return HxString("abs"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAbs.h**

6.290 HxUpoAcos Class Template Reference

Pixel functor for computation of arc cosine.

`#include <HxUpoAcos.h>`

Public Methods

- **HxUpoAcos** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.acos() #.

Static Public Methods

- **HxString className** ()
  The name : "acos".

6.290.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAcos< DstValT, SrcValT >

Pixel functor for computation of arc cosine.

6.290.2 Constructor & Destructor Documentation

6.290.2.1 template<class DstValT, class SrcValT> HxUpoAcos< DstValT, SrcValT >::HxUpoAcos (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.290.3 Member Function Documentation

6.290.3.1 template<class DstValT, class SrcValT> DstValT HxUpoAcos< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.acos() #.

```
29                      { return x.acos(); }
```

6.290.3.2 template<class DstValT, class SrcValT> HxString HxUpoAcos< DstValT, SrcValT >::className () [inline, static]

The name : "acos".

```
33                      { return HxString("acos"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAcos.h**

6.291 HxUpoArg Class Template Reference

Pixel functor for computation of argument.

`#include <HxUpoArg.h>`

Public Methods

- **HxUpoArg** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.arg() #.

Static Public Methods

- **HxString className** ()

  The name : "arg".

6.291.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoArg< DstValT, SrcValT >

Pixel functor for computation of argument.

6.291.2 Constructor & Destructor Documentation

6.291.2.1 template<class DstValT, class SrcValT> HxUpoArg< DstValT, SrcValT >::HxUpoArg (HxTagList &) [inline]

Constructor : empty.

```
24                      {}
```

6.291.3 Member Function Documentation

6.291.3.1 template<class DstValT, class SrcValT> DstValT HxUpoArg< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.arg() #.

```
28                      { return x.arg(); }
```

6.291.3.2 template<class DstValT, class SrcValT> HxString HxUpoArg< DstValT, SrcValT >::className () [inline, static]

The name : ”arg”.

```
32                      { return HxString("arg"); }
```

The documentation for this class was generated from the following file:

- **HxUpoArg.h**

6.292 HxUpoAsin Class Template Reference

Pixel functor for computation of arc sine.

`#include <HxUpoAsin.h>`

Public Methods

- **HxUpoAsin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.asin() #.

Static Public Methods

- **HxString className** ()
  The name : ”asin”.

6.292.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAsin< DstValT, SrcValT >

Pixel functor for computation of arc sine.

6.292.2 Constructor & Destructor Documentation

6.292.2.1 template<class DstValT, class SrcValT> HxUpoAsin< DstValT, SrcValT >::HxUpoAsin (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.292.3 Member Function Documentation

6.292.3.1 template<class DstValT, class SrcValT> DstValT HxUpoAsin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.asin() #.

```
29                          { return x.asin(); }
```

6.292.3.2 template<class DstValT, class SrcValT> HxString HxUpoAsin< DstValT, SrcValT >::className () [inline, static]

The name : ”asin”.

```
33                          { return HxString("asin"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAsin.h**

6.293 HxUpoAtan Class Template Reference

Pixel functor for computation of arc tangent.

`#include <HxUpoAtan.h>`

Public Methods

- **HxUpoAtan** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.atan() #.

Static Public Methods

- **HxString className** ()
  The name : ”atan”.

6.293.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAtan< DstValT, SrcValT >

Pixel functor for computation of arc tangent.

6.293.2 Constructor & Destructor Documentation

6.293.2.1 template<class DstValT, class SrcValT> HxUpoAtan< DstValT, SrcValT >::HxUpoAtan (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.293.3 Member Function Documentation

6.293.3.1 template<class DstValT, class SrcValT> DstValT HxUpoAtan< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.atan() #.

```
29                          { return x.atan(); }
```

6.293.3.2 template<class DstValT, class SrcValT> HxString HxUpoAtan< DstValT, SrcValT >::className () [inline, static]

The name : ”atan”.

```
33                          { return HxString("atan"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAtan.h**

6.294 HxUpoAtan2 Class Template Reference

Pixel functor for computation of arc tangent.

`#include <HxUpoAtan2.h>`

Public Methods

- **HxUpoAtan2** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.atan2() #.

Static Public Methods

- **HxString className** ()
  The name : ”atan2”.

6.294.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAtan2< DstValT, SrcValT >

Pixel functor for computation of arc tangent.

6.294.2 Constructor & Destructor Documentation

6.294.2.1 template<class DstValT, class SrcValT> HxUpoAtan2< DstValT, SrcValT >::HxUpoAtan2 (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.294.3 Member Function Documentation

6.294.3.1 template<class DstValT, class SrcValT> DstValT HxUpoAtan2< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.atan2() #.

```
29                          { return x.atan2(); }
```

6.294.3.2 template<class DstValT, class SrcValT> HxString HxUpoAtan2< DstValT, SrcValT >::className () [inline, static]

The name : ”atan2”.

```
33                          { return HxString("atan2"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAtan2.h**

6.295 HxUpoCeil Class Template Reference

Pixel functor for computation of ceiling.

`#include <HxUpoCeil.h>`

Public Methods

- **HxUpoCeil** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.ceil() #.

Static Public Methods

- **HxString className** ()

  The name : ”ceil”.

6.295.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCeil< DstValT, SrcValT >

Pixel functor for computation of ceiling.

6.295.2 Constructor & Destructor Documentation

6.295.2.1 template<class DstValT, class SrcValT> HxUpoCeil< DstValT, SrcValT >::HxUpoCeil (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.295.3 Member Function Documentation

6.295.3.1 template<class DstValT, class SrcValT> DstValT HxUpoCeil< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.ceil() #.

```
29                          { return x.ceil(); }
```

6.295.3.2 template<class DstValT, class SrcValT> HxString HxUpoCeil< DstValT, SrcValT >::className () [inline, static]

The name : ”ceil”.

```
33                          { return HxString("ceil"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCeil.h**

6.296 HxUpoColSpace Class Template Reference

Pixel functor for color space conversion.

`#include <HxUpoColSpace.h>`

Public Methods

- **HxUpoColSpace** (**HxTagList** &tags)

  Constructor.

- DstValT **doIt** (const SrcValT &x)

  Actual operation :.

Static Public Methods

- **HxString className** ()

  The name : "colorSpace".

6.296.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoColSpace< DstValT, SrcValT >

Pixel functor for color space conversion.

6.296.2 Constructor & Destructor Documentation

6.296.2.1 template<class DstValT, class SrcValT> HxUpoColSpace< DstValT, SrcValT >::HxUpoColSpace (HxTagList & *tags*) [inline]

Constructor.

```
25                          {
26                              _fromSpace = HxGetTag<HxColorModel>(tags,
27                                  "fromColorSpace");
28                              _toSpace = HxGetTag<HxColorModel>(tags,
29                                  "toColorSpace");
30                          }
```

6.296.3 Member Function Documentation

6.296.3.1 template<class DstValT, class SrcValT> DstValT HxUpoColSpace< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation :.

```
34                          { return HxColor(x,_fromSpace).convert(_toSpace).value();}
```

6.296.3.2 template<class DstValT, class SrcValT> HxString HxUpoColSpace< DstValT, SrcValT >::className () [inline, static]

The name : "colorSpace".

```
38                          { return HxString("colorSpace"); }
```

The documentation for this class was generated from the following file:

- **HxUpoColSpace.h**

6.297 HxUpoComplement Class Template Reference

Pixel functor for computation of comlement.

`#include <HxUpoComplement.h>`

Public Methods

- **HxUpoComplement** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.complement() #.

Static Public Methods

- **HxString className** ()
  The name : "complement".

6.297.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoComplement< DstValT, SrcValT >

Pixel functor for computation of comlement.

6.297.2 Constructor & Destructor Documentation

6.297.2.1 template<class DstValT, class SrcValT> HxUpoComplement< DstValT, SrcValT >::HxUpoComplement (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.297.3 Member Function Documentation

6.297.3.1 template<class DstValT, class SrcValT> DstValT HxUpoComplement< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.complement() #.

```
29                          { return x.complement(); }
```

6.297.3.2 template<class DstValT, class SrcValT> HxString HxUpoComplement< DstValT, SrcValT >::className () [inline, static]

The name : ”complement”.

```
33                          { return HxString("complement"); }
```

The documentation for this class was generated from the following file:

- **HxUpoComplement.h**

6.298 HxUpoConjugate Class Template Reference

Pixel functor for computation of conjugate.

`#include <HxUpoConjugate.h>`

Public Methods

- **HxUpoConjugate** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.conjugate() #.

Static Public Methods

- **HxString className** ()
  The name : ”conjugate”.

6.298.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoConjugate< DstValT, SrcValT >

Pixel functor for computation of conjugate.

6.298.2 Constructor & Destructor Documentation

6.298.2.1 template<class DstValT, class SrcValT> HxUpoConjugate< DstValT, SrcValT >::HxUpoConjugate (HxTagList &) [inline]

Constructor : empty.

```
24                          {}
```

6.298.3 Member Function Documentation

6.298.3.1 template<class DstValT, class SrcValT> DstValT HxUpoConjugate< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.conjugate() #.

```
28                          { return x.conjugate(); }
```

6.298.3.2 template<class DstValT, class SrcValT> HxString HxUpoConjugate< DstValT, SrcValT >::className () [inline, static]

The name : ”conjugate”.

```
32                          { return HxString("conjugate"); }
```

The documentation for this class was generated from the following file:

- **HxUpoConjugate.h**

6.299 HxUpoCos Class Template Reference

Pixel functor for computation of cosine.

`#include <HxUpoCos.h>`

Public Methods

- **HxUpoCos** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.cos() #.

Static Public Methods

- **HxString className** ()
  The name : ”cos”.

6.299.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCos< DstValT, SrcValT >

Pixel functor for computation of cosine.

6.299.2 Constructor & Destructor Documentation

6.299.2.1 template<class DstValT, class SrcValT> HxUpoCos< DstValT, SrcValT >::HxUpoCos (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.299.3 Member Function Documentation

6.299.3.1 template<class DstValT, class SrcValT> DstValT HxUpoCos< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.cos() #.

```
29                          { return x.cos(); }
```

6.299.3.2 template<class DstValT, class SrcValT> HxString HxUpoCos< DstValT, SrcValT >::className () [inline, static]

The name : ”cos”.

```
33                          { return HxString("cos"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCos.h**

6.300 HxUpoCosh Class Template Reference

Pixel functor for computation of hyperbolic consine.

`#include <HxUpoCosh.h>`

Public Methods

- **HxUpoCosh** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.cosh() #.

Static Public Methods

- **HxString className** ()
  The name : ”cosh”.

6.300.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCosh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic consine.

6.300.2 Constructor & Destructor Documentation

6.300.2.1 template<class DstValT, class SrcValT> HxUpoCosh< DstValT, SrcValT >::HxUpoCosh (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.300.3 Member Function Documentation

6.300.3.1 template<class DstValT, class SrcValT> DstValT HxUpoCosh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.cosh() #.

```
29                          { return x.cosh(); }
```

6.300.3.2 template<class DstValT, class SrcValT> HxString HxUpoCosh< DstValT, SrcValT >::className () [inline, static]

The name : ”cosh”.

```
33                          { return HxString("cosh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCosh.h**

6.301 HxUpoExp Class Template Reference

Pixel functor for computation of exponent.

`#include <HxUpoExp.h>`

Public Methods

- **HxUpoExp** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.exp() #.

Static Public Methods

- **HxString className** ()

  The name : ”exp”.

6.301.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoExp< DstValT, SrcValT >

Pixel functor for computation of exponent.

6.301.2 Constructor & Destructor Documentation

6.301.2.1 template<class DstValT, class SrcValT> HxUpoExp< DstValT, SrcValT >::HxUpoExp (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.301.3 Member Function Documentation

6.301.3.1 template<class DstValT, class SrcValT> DstValT HxUpoExp< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.exp() #.

```
29                          { return x.exp(); }
```

6.301.3.2 template<class DstValT, class SrcValT> HxString HxUpoExp< DstValT, SrcValT >::className () [inline, static]

The name : ”exp”.

```
33                          { return HxString("exp"); }
```

The documentation for this class was generated from the following file:

- **HxUpoExp.h**

6.302 HxUpoFloor Class Template Reference

Pixel functor for computation of floor.

`#include <HxUpoFloor.h>`

Public Methods

- **HxUpoFloor** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.floor() #.

Static Public Methods

- **HxString className** ()
  The name : "floor".

6.302.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoFloor< DstValT, SrcValT >

Pixel functor for computation of floor.

6.302.2 Constructor & Destructor Documentation

6.302.2.1 template<class DstValT, class SrcValT> HxUpoFloor< DstValT, SrcValT >::HxUpoFloor (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.302.3 Member Function Documentation

6.302.3.1 template<class DstValT, class SrcValT> DstValT HxUpoFloor< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.floor() #.

```
29                      { return x.floor(); }
```

6.302.3.2 template<class DstValT, class SrcValT> HxString HxUpoFloor< DstValT, SrcValT >::className () [inline, static]

The name : "floor".

```
33                      { return HxString("floor"); }
```

The documentation for this class was generated from the following file:

- **HxUpoFloor.h**

6.303 HxUpoLog Class Template Reference

Pixel functor for computation of natural logarithm.

`#include <HxUpoLog.h>`

Public Methods

- **HxUpoLog** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.log() #.

Static Public Methods

- **HxString className** ()

  The name : "log".

6.303.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoLog< DstValT, SrcValT >

Pixel functor for computation of natural logarithm.

6.303.2 Constructor & Destructor Documentation

6.303.2.1 template<class DstValT, class SrcValT> HxUpoLog< DstValT, SrcValT >::HxUpoLog (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.303.3 Member Function Documentation

6.303.3.1 template<class DstValT, class SrcValT> DstValT HxUpoLog< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.log() #.

```
29                      { return x.log(); }
```

6.303.3.2 template<class DstValT, class SrcValT> HxString HxUpoLog< DstValT, SrcValT >::className () [inline, static]

The name : ”log”.

```
33                      { return HxString("log"); }
```

The documentation for this class was generated from the following file:

- **HxUpoLog.h**

6.304 HxUpoLog10 Class Template Reference

Pixel functor for computation of base 10 logarithm.

`#include <HxUpoLog10.h>`

Public Methods

- **HxUpoLog10** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.log10() #.

Static Public Methods

- **HxString className** ()
  The name : ”log10”.

6.304.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoLog10< DstValT, SrcValT >

Pixel functor for computation of base 10 logarithm.

6.304.2 Constructor & Destructor Documentation

6.304.2.1 template<class DstValT, class SrcValT> HxUpoLog10< DstValT, SrcValT >::HxUpoLog10 (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.304.3 Member Function Documentation

6.304.3.1 template<class DstValT, class SrcValT> DstValT HxUpoLog10< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.log10() #.

```
29                          { return x.log10(); }
```

6.304.3.2 template<class DstValT, class SrcValT> HxString HxUpoLog10< DstValT, SrcValT >::className () [inline, static]

The name : ”log10”.

```
33                          { return HxString("log10"); }
```

The documentation for this class was generated from the following file:

- **HxUpoLog10.h**

6.305 HxUpoMax Class Template Reference

Pixel functor for computation of maximum.

`#include <HxUpoMax.h>`

Public Methods

- **HxUpoMax** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.max() #.

Static Public Methods

- **HxString className** ()

  The name : ”max”.

6.305.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoMax< DstValT, SrcValT >

Pixel functor for computation of maximum.

6.305.2 Constructor & Destructor Documentation

6.305.2.1 template<class DstValT, class SrcValT> HxUpoMax< DstValT, SrcValT >::HxUpoMax (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.305.3 Member Function Documentation

6.305.3.1 template<class DstValT, class SrcValT> DstValT HxUpoMax< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.max() #.

```
29                          { return x.max(); }
```

6.305.3.2 template<class DstValT, class SrcValT> HxString HxUpoMax< DstValT, SrcValT >::className () [inline, static]

The name : ”max”.

```
33                          { return HxString("max"); }
```

The documentation for this class was generated from the following file:

- **HxUpoMax.h**

6.306 HxUpoMin Class Template Reference

Pixel functor for computation of minimum.

`#include <HxUpoMin.h>`

Public Methods

- **HxUpoMin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.min() #.

Static Public Methods

- **HxString className** ()
  The name : ”min”.

6.306.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoMin< DstValT, SrcValT >

Pixel functor for computation of minimum.

6.306.2 Constructor & Destructor Documentation

6.306.2.1 template<class DstValT, class SrcValT> HxUpoMin< DstValT, SrcValT >::HxUpoMin (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.306.3 Member Function Documentation

6.306.3.1 template<class DstValT, class SrcValT> DstValT HxUpoMin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.min() #.

```
29                          { return x.min(); }
```

6.306.3.2 template<class DstValT, class SrcValT> HxString HxUpoMin< DstValT, SrcValT >::className () [inline, static]

The name : ”min”.

```
33                          { return HxString("min"); }
```

The documentation for this class was generated from the following file:

- **HxUpoMin.h**

6.307 HxUpoNegate Class Template Reference

Pixel functor for computation of negation.

`#include <HxUpoNegate.h>`

Public Methods

- **HxUpoNegate** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return -x #.

Static Public Methods

- **HxString className** ()

  The name : ”negate”.

6.307.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNegate< DstValT, SrcValT >

Pixel functor for computation of negation.

6.307.2 Constructor & Destructor Documentation

6.307.2.1 template<class DstValT, class SrcValT> HxUpoNegate< DstValT, SrcValT >::HxUpoNegate (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.307.3 Member Function Documentation

6.307.3.1 template<class DstValT, class SrcValT> DstValT HxUpoNegate< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return -x #.

```
29                          { return -x; }
```

6.307.3.2 template<class DstValT, class SrcValT> HxString HxUpoNegate< DstValT, SrcValT >::className () [inline, static]

The name : ”negate”.

```
33                          { return HxString("negate"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNegate.h**

6.308 HxUpoNorm1 Class Template Reference

Pixel functor for computation of L1 norm.

`#include <HxUpoNorm1.h>`

Public Methods

- **HxUpoNorm1** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.norm1() #.

Static Public Methods

- **HxString className** ()
  The name : "norm1".

6.308.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm1< DstValT, SrcValT >

Pixel functor for computation of L1 norm.

6.308.2 Constructor & Destructor Documentation

6.308.2.1 template<class DstValT, class SrcValT> HxUpoNorm1< DstValT, SrcValT >::HxUpoNorm1 (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.308.3 Member Function Documentation

6.308.3.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm1< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.norm1() #.

```
29                      { return x.norm1(); }
```

6.308.3.2 template<class DstValT, class SrcValT> HxString HxUpoNorm1< DstValT, SrcValT >::className () [inline, static]

The name : "norm1".

```
33                      { return HxString("norm1"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm1.h**

6.309 HxUpoNorm2 Class Template Reference

Pixel functor for computation of L2 norm.

`#include <HxUpoNorm2.h>`

Public Methods

- **HxUpoNorm2** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.norm2() #.

Static Public Methods

- **HxString className** ()

  The name : "norm2".

6.309.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm2< DstValT, SrcValT >

Pixel functor for computation of L2 norm.

6.309.2 Constructor & Destructor Documentation

6.309.2.1 template<class DstValT, class SrcValT> HxUpoNorm2< DstValT, SrcValT >::HxUpoNorm2 (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.309.3 Member Function Documentation

6.309.3.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm2< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.norm2() #.

```
29                      { return x.norm2(); }
```

6.309.3.2 template<class DstValT, class SrcValT> HxString HxUpoNorm2< DstValT, SrcValT >::className () [inline, static]

The name : ”norm2”.

```
33                      { return HxString("norm2"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm2.h**

6.310 HxUpoNorm2Sqr Class Template Reference

Pixel functor for computation of squared L2 norm.

`#include <HxUpoNorm2Sqr.h>`

Public Methods

- **HxUpoNorm2Sqr** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return (x∗x).sum() #.

Static Public Methods

- **HxString className** ()

  The name : ”norm2sqr”.

6.310.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm2Sqr< DstValT, SrcValT >

Pixel functor for computation of squared L2 norm.

6.310.2 Constructor & Destructor Documentation

6.310.2.1 template<class DstValT, class SrcValT> HxUpoNorm2Sqr< DstValT, SrcValT >::HxUpoNorm2Sqr (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.310.3 Member Function Documentation

6.310.3.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm2Sqr< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return (x∗x).sum() #.

```
29                          { return (x*x).sum(); }
```

6.310.3.2 template<class DstValT, class SrcValT> HxString HxUpoNorm2Sqr< DstValT, SrcValT >::className () [inline, static]

The name : ”norm2sqr”.

```
33                          { return HxString("norm2sqr"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm2Sqr.h**

6.311 HxUpoNormInf Class Template Reference

Pixel functor for computation of L inf norm.

`#include <HxUpoNormInf.h>`

Public Methods

- **HxUpoNormInf** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.normInf() #.

Static Public Methods

- **HxString className** ()
  The name : ”normInf”.

6.311.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNormInf< DstValT, SrcValT >

Pixel functor for computation of L inf norm.

6.311.2 Constructor & Destructor Documentation

6.311.2.1 template<class DstValT, class SrcValT> HxUpoNormInf< DstValT, SrcValT >::HxUpoNormInf (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.311.3 Member Function Documentation

6.311.3.1 template<class DstValT, class SrcValT> DstValT HxUpoNormInf< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.normInf() #.

```
29                          { return x.normInf(); }
```

6.311.3.2 template<class DstValT, class SrcValT> HxString HxUpoNormInf< DstValT, SrcValT >::className () [inline, static]

The name : ”normInf”.

```
33                          { return HxString("normInf"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNormInf.h**

6.312 HxUpoProduct Class Template Reference

Pixel functor for computation of unary product.

`#include <HxUpoProduct.h>`

Public Methods

- **HxUpoProduct** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.product() #.

Static Public Methods

- **HxString className** ()
  The name : ”product”.

6.312.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoProduct< DstValT, SrcValT >

Pixel functor for computation of unary product.

6.312.2 Constructor & Destructor Documentation

6.312.2.1 template<class DstValT, class SrcValT> HxUpoProduct< DstValT, SrcValT >::HxUpoProduct (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.312.3 Member Function Documentation

6.312.3.1 template<class DstValT, class SrcValT> DstValT HxUpoProduct< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.product() #.

```
29                      { return x.product(); }
```

6.312.3.2 template<class DstValT, class SrcValT> HxString HxUpoProduct< DstValT, SrcValT >::className () [inline, static]

The name : ”product”.

```
33                      { return HxString("product"); }
```

The documentation for this class was generated from the following file:

- **HxUpoProduct.h**

6.313 HxUpoRound Class Template Reference

Pixel functor for computation of round.

`#include <HxUpoRound.h>`

Public Methods

- **HxUpoRound** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.round() #.

Static Public Methods

- **HxString className** ()

  The name : ”round”.

6.313.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoRound< DstValT, SrcValT >

Pixel functor for computation of round.

6.313.2 Constructor & Destructor Documentation

6.313.2.1 template<class DstValT, class SrcValT> HxUpoRound< DstValT, SrcValT >::HxUpoRound (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.313.3 Member Function Documentation

6.313.3.1 template<class DstValT, class SrcValT> DstValT HxUpoRound< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.round() #.

```
29                          { return x.round(); }
```

6.313.3.2 template<class DstValT, class SrcValT> HxString HxUpoRound< DstValT, SrcValT >::className () [inline, static]

The name : ”round”.

```
33                          { return HxString("round"); }
```

The documentation for this class was generated from the following file:

- **HxUpoRound.h**

6.314 HxUpoSin Class Template Reference

Pixel functor for computation of sine.

`#include <HxUpoSin.h>`

Public Methods

- **HxUpoSin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sin() #.

Static Public Methods

- **HxString className** ()
  The name : "sin".

6.314.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSin< DstValT, SrcValT >

Pixel functor for computation of sine.

6.314.2 Constructor & Destructor Documentation

6.314.2.1 template<class DstValT, class SrcValT> HxUpoSin< DstValT, SrcValT >::HxUpoSin (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.314.3 Member Function Documentation

6.314.3.1 template<class DstValT, class SrcValT> DstValT HxUpoSin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sin() #.

```
29                      { return x.sin(); }
```

6.314.3.2 template<class DstValT, class SrcValT> HxString HxUpoSin< DstValT, SrcValT >::className () [inline, static]

The name : "sin".

```
33                      { return HxString("sin"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSin.h**

6.315 HxUpoSinh Class Template Reference

Pixel functor for computation of hyperbolic sine.

`#include <HxUpoSinh.h>`

Public Methods

- **HxUpoSinh** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.sinh() #.

Static Public Methods

- **HxString className** ()

  The name : "sinh".

6.315.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSinh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic sine.

6.315.2 Constructor & Destructor Documentation

6.315.2.1 template<class DstValT, class SrcValT> HxUpoSinh< DstValT, SrcValT >::HxUpoSinh (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.315.3 Member Function Documentation

6.315.3.1 template<class DstValT, class SrcValT> DstValT HxUpoSinh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sinh() #.

```
29                      { return x.sinh(); }
```

6.315.3.2 template<class DstValT, class SrcValT> HxString HxUpoSinh< DstValT, SrcValT >::className () [inline, static]

The name : ”sinh”.

```
33                      { return HxString("sinh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSinh.h**

6.316 HxUpoSqrt Class Template Reference

Pixel functor for computation of square root.

`#include <HxUpoSqrt.h>`

Public Methods

- **HxUpoSqrt** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sqrt() #.

Static Public Methods

- **HxString className** ()
  The name : ”sqrt”.

6.316.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSqrt< DstValT, SrcValT >

Pixel functor for computation of square root.

6.316.2 Constructor & Destructor Documentation

6.316.2.1 template<class DstValT, class SrcValT> HxUpoSqrt< DstValT, SrcValT >::HxUpoSqrt (HxTagList &) [inline]

Constructor : empty.

```
25                      {}
```

6.316.3 Member Function Documentation

6.316.3.1 template<class DstValT, class SrcValT> DstValT HxUpoSqrt< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sqrt() #.

```
29                          { return x.sqrt(); }
```

6.316.3.2 template<class DstValT, class SrcValT> HxString HxUpoSqrt< DstValT, SrcValT >::className () [inline, static]

The name : ”sqrt”.

```
33                          { return HxString("sqrt"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSqrt.h**

6.317 HxUpoSum Class Template Reference

Pixel functor for computation of unary sum.

`#include <HxUpoSum.h>`

Public Methods

- **HxUpoSum** (**HxTagList** &)

  Constructor : empty.

- DstValT **doIt** (const SrcValT &x)

  Actual operation : # return x.sum() #.

Static Public Methods

- **HxString className** ()

  The name : ”sum”.

6.317.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSum< DstValT, SrcValT >

Pixel functor for computation of unary sum.

6.317.2 Constructor & Destructor Documentation

6.317.2.1 template<class DstValT, class SrcValT> HxUpoSum< DstValT, SrcValT >::HxUpoSum (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.317.3 Member Function Documentation

6.317.3.1 template<class DstValT, class SrcValT> DstValT HxUpoSum< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sum() #.

```
29                          { return x.sum(); }
```

6.317.3.2 template<class DstValT, class SrcValT> HxString HxUpoSum< DstValT, SrcValT >::className () [inline, static]

The name : ”sum”.

```
33                          { return HxString("sum"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSum.h**

6.318 HxUpoTan Class Template Reference

Pixel functor for computation of tangent.

`#include <HxUpoTan.h>`

Public Methods

- **HxUpoTan** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.tan() #.

Static Public Methods

- **HxString className** ()
  The name : ”tan”.

6.318.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTan< DstValT, SrcValT >

Pixel functor for computation of tangent.

6.318.2 Constructor & Destructor Documentation

6.318.2.1 template<class DstValT, class SrcValT> HxUpoTan< DstValT, SrcValT >::HxUpoTan (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.318.3 Member Function Documentation

6.318.3.1 template<class DstValT, class SrcValT> DstValT HxUpoTan< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.tan() #.

```
29                          { return x.tan(); }
```

6.318.3.2 template<class DstValT, class SrcValT> HxString HxUpoTan< DstValT, SrcValT >::className () [inline, static]

The name : ”tan”.

```
33                          { return HxString("tan"); }
```

The documentation for this class was generated from the following file:

- **HxUpoTan.h**

6.319 HxUpoTanh Class Template Reference

Pixel functor for computation of hyperbolic tangent.

`#include <HxUpoTanh.h>`

Public Methods

- **HxUpoTanh** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.tanh() #.

Static Public Methods

- **HxString className** ()

  The name : "tanh".

6.319.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTanh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic tangent.

6.319.2 Constructor & Destructor Documentation

6.319.2.1 template<class DstValT, class SrcValT> HxUpoTanh< DstValT, SrcValT >::HxUpoTanh (HxTagList &) [inline]

Constructor : empty.

```
25                          {}
```

6.319.3 Member Function Documentation

6.319.3.1 template<class DstValT, class SrcValT> DstValT HxUpoTanh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.tanh() #.

```
29                          { return x.tanh(); }
```

6.319.3.2 template<class DstValT, class SrcValT> HxString HxUpoTanh< DstValT, SrcValT >::className () [inline, static]

The name : "tanh".

```
33                          { return HxString("tanh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoTanh.h**

6.320 HxUpoTriStateThreshold Class Template Reference

Pixel functor for computation of tri state threshold.

Public Methods

- **HxUpoTriStateThreshold** (**HxTagList** &)

  Constructor : get parameters from taglist.

- DstValT **doIt** (const SrcValT &x)

  Actual operation.

Static Public Methods

- **HxString className** ()

  The name : "triStateThreshold".

6.320.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTriStateThreshold< DstValT, SrcValT >

Pixel functor for computation of tri state threshold.

6.320.2 Constructor & Destructor Documentation

6.320.2.1 template<class DstValT, class SrcValT> HxUpoTriStateThreshold< DstValT, SrcValT >::HxUpoTriStateThreshold (HxTagList & *tl*)

Constructor : get parameters from taglist.

```
40 {
41     _level = HxGetTag<HxValue>(tl, "level");
42     _v1 = HxGetTag<HxValue>(tl, "v1");
43     _v2 = HxGetTag<HxValue>(tl, "v2");
44     _v3 = HxGetTag<HxValue>(tl, "v3");
45 }
```

6.320.3 Member Function Documentation

6.320.3.1 template<class DstValT, class SrcValT> DstValT HxUpoTriStateThreshold< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation.

```
50 {
51     if (x < _level)
52         return _v1;
53     if (x > _level)
54         return _v3;
55     return _v2;
56 }
```

6.320.3.2 template<class DstValT, class SrcValT> HxString HxUpoTriStateThreshold< DstValT, SrcValT >::className () [static]

The name : ”triStateThreshold”.

```
61 {
62     return HxString("triStateThreshold");
63 }
```

The documentation for this class was generated from the following file:

- HxTriStateThreshold.c

6.321 HxValue Class Reference

Class definition of a value as used in Horus.

`#include <HxValue.h>`

Constructors

- **HxValue** (int i=0)

  Construction from native integer.

- **HxValue** (double d)

  Construction from native double.

- **HxValue** (**HxScalarInt** i)

  Construction from scalar integer.

- **HxValue** (**HxScalarDouble** d)

  Construction from scalar double.

- **HxValue** (**HxVec2Int** v)

  Construction from vector of 2 integers.

- **HxValue** (**HxVec2Double** v)

  Construction from vector of 2 doubles.

- **HxValue** (**HxVec3Int** v)

  Construction from vector of 3 integers.

- **HxValue** (**HxVec3Double** v)

  Construction from vector of 3 doubles.

- **HxValue** (**HxComplex** v)

  Construction from complex.

Inquiry

- **Tag tag** () const

  The type of the value stored.

Data access

- **HxScalarInt** & **HxScalarIntValue** ()

  Access value as **HxScalarInt** (p. 696).

- **HxScalarDouble** & **HxScalarDoubleValue** ()

  Access value as **HxScalarDouble** (p. 677).

- **HxVec2Int** & **HxVec2IntValue** ()

  Access value as **HxVec2Int** (p. 785).

- **HxVec2Double** & **HxVec2DoubleValue** ()

  Access value as **HxVec2Double** (p. 766).

- **HxVec3Int** & **HxVec3IntValue** ()

  Access value as **HxVec3Int** (p. 824).

- **HxVec3Double** & **HxVec3DoubleValue** ()

  Access value as **HxVec3Double** (p. 804).

- **HxComplex** & **HxComplexValue** ()

  Access value as **HxComplex** (p. 239).

Conversion

- **operator HxScalarInt** () const

  Cast to **HxScalarInt** (p. 696).

- **operator HxScalarDouble** () const

  Cast to **HxScalarDouble** (p. 677).

- **operator HxVec2Int** () const

  Cast to **HxVec2Int** (p. 785).

- **operator HxVec2Double** () const

  Cast to **HxVec2Double** (p. 766).

- **operator HxVec3Int** () const

  Cast to **HxVec3Int** (p. 824).

- **operator HxVec3Double** () const

  Cast to **HxVec3Double** (p. 804).

- **operator HxComplex** () const

  Cast to **HxComplex** (p. 239).

Public Types

- enum **Tag** { **SI**, **SD**, **V2I**, **V2D**, **V3I**, **V3D**, **CPL** }

  The arithmetic data types that may be stored.

Public Methods

- **HxString toString** () const

Friends

- STD_OSTREAM & **operator**<< (STD_OSTREAM &os, const HxValue val)

6.321.1 Detailed Description

Class definition of a value as used in Horus.

An HxValue is capable of storing each of the arithmetic data types. HxValue is used to pass arbitrary values using a single function interface. HxValue is not meant to do arithmetic on arbitrary values and hence no such operations are defined in the interface.

6.321.2 Member Enumeration Documentation

6.321.2.1 enum HxValue::Tag

The arithmetic data types that may be stored.

SI: scalar integer, SD: scalar double, V2I: vector of 2 integers, V2D: vector of 2 doubles, V3I: vector of 3 integers, V3D: vector of 3 doubles. CPL: complex number (2 doubles).

```
42 { SI, SD, V2I, V2D, V3I, V3D, CPL };
```

6.321.3 Constructor & Destructor Documentation

6.321.3.1 HxValue::HxValue (int *v* = 0) [inline]

Construction from native integer.

```
159 {
160     _tag._tag = SI;
161     new(&_val[0]) HxScalarInt(v);
162 }
```

6.321.3.2 HxValue::HxValue (double *v*) [inline]

Construction from native double.

```
166 {
167     _tag._tag = SD;
168     new(&_val[0]) HxScalarDouble(v);
169 }
```

6.321.3.3 HxValue::HxValue (HxScalarInt *v*) [inline]

Construction from scalar integer.

```
173 {
174     _tag._tag = SI;
175     new(&_val[0]) HxScalarInt(v);
176 }
```

6.321.3.4 HxValue::HxValue (HxScalarDouble *v*) [inline]

Construction from scalar double.

```
180 {
181     _tag._tag = SD;
182     new(&_val[0]) HxScalarDouble(v);
183 }
```

6.321.3.5 HxValue::HxValue (HxVec2Int *v*) [inline]

Construction from vector of 2 integers.

```
187 {
188     _tag._tag = V2I;
189     new(&_val[0]) HxVec2Int(v);
190 }
```

6.321.3.6 HxValue::HxValue (HxVec2Double *v*) [inline]

Construction from vector of 2 doubles.

```
194 {
195     _tag._tag = V2D;
196     new(&_val[0]) HxVec2Double(v);
197 }
```

6.321.3.7 HxValue::HxValue (HxVec3Int *v*) [inline]

Construction from vector of 3 integers.

```
201 {
202     _tag._tag = V3I;
203     new(&_val[0]) HxVec3Int(v);
204 }
```

6.321.3.8 HxValue::HxValue (HxVec3Double *v*) [inline]

Construction from vector of 3 doubles.

```
208 {
209     _tag._tag = V3D;
210     new(&_val[0]) HxVec3Double(v);
211 }
```

6.321.3.9 HxValue::HxValue (HxComplex *v*) [inline]

Construction from complex.

```
215 {
216     _tag._tag = CPL;
217     new(&_val[0]) HxComplex(v);
218 }
```

6.321.4 Member Function Documentation

6.321.4.1 HxValue::Tag HxValue::tag () const [inline]

The type of the value stored.

```
222 {
223     return _tag._tag;
224 }
```

6.321.4.2 HxScalarInt & HxValue::HxScalarIntValue () [inline]

Access value as **HxScalarInt** (p. 696).

```
228 {
229     check(SI);
230     return *(HxScalarInt *) &_val[0];
231 }
```

6.321.4.3 HxScalarDouble & HxValue::HxScalarDoubleValue () [inline]

Access value as **HxScalarDouble** (p. 677).

```
235 {
236     check(SD);
237     return *(HxScalarDouble *) &_val[0];
238 }
```

6.321.4.4 HxVec2Int & HxValue::HxVec2IntValue () [inline]

Access value as **HxVec2Int** (p. 785).

```
242 {
243     check(V2I);
244     return *(HxVec2Int *) &_val[0];
245 }
```

6.321.4.5 HxVec2Double & HxValue::HxVec2DoubleValue () [inline]

Access value as **HxVec2Double** (p. 766).

```
249 {
250     check(V2D);
251     return *(HxVec2Double *) &_val[0];
252 }
```

6.321.4.6 HxVec3Int & HxValue::HxVec3IntValue () [inline]

Access value as **HxVec3Int** (p. 824).

```
256 {
257     check(V3I);
258     return *(HxVec3Int *) &_val[0];
259 }
```

6.321.4.7 HxVec3Double & HxValue::HxVec3DoubleValue () [inline]

Access value as **HxVec3Double** (p. 804).

```
263 {
264     check(V3D);
265     return *(HxVec3Double *) &_val[0];
266 }
```

6.321.4.8 HxComplex & HxValue::HxComplexValue () [inline]

Access value as **HxComplex** (p. 239).

```
270 {
271     check(CPL);
272     return *(HxComplex *) &_val[0];
273 }
```

6.321.4.9 HxValue::operator HxScalarInt () const [inline]

Cast to **HxScalarInt** (p. 696).

```
277 {
278     switch(_tag._tag)
279     {
280     case SI:    return (HxScalarInt) *(HxScalarInt *) &_val[0];
281     case SD:    return (*(HxScalarDouble *) &_val[0]).operator HxScalarInt();
282     case V2I:   return (HxScalarInt) *(HxVec2Int *) &_val[0];
283     case V2D:   return (HxScalarInt) *(HxVec2Double *) &_val[0];
284     case V3I:   return (HxScalarInt) *(HxVec3Int *) &_val[0];
285     case V3D:   return (HxScalarInt) *(HxVec3Double *) &_val[0];
286     case CPL:   return (HxScalarInt) *(HxComplex *) &_val[0];
287     default:    return (HxScalarInt) *(HxScalarInt *) &_val[0];
288     }
289 }
```

6.321.4.10 HxValue::operator HxScalarDouble () const [inline]

Cast to **HxScalarDouble** (p. 677).

```
293 {
294     switch(_tag._tag)
295     {
296     case SI:    return (HxScalarDouble) *(HxScalarInt *) &_val[0];
297     case SD:    return (HxScalarDouble) *(HxScalarDouble *) &_val[0];
298     case V2I:   return (HxScalarDouble) *(HxVec2Int *) &_val[0];
299     case V2D:   return (HxScalarDouble) *(HxVec2Double *) &_val[0];
300     case V3I:   return (HxScalarDouble) *(HxVec3Int *) &_val[0];
301     case V3D:   return (HxScalarDouble) *(HxVec3Double *) &_val[0];
302     case CPL:   return (HxScalarDouble) *(HxComplex *) &_val[0];
303     default:    return (HxScalarDouble) *(HxScalarDouble *) &_val[0];
304     }
305 }
```

6.321.4.11 HxValue::operator HxVec2Int () const [inline]

Cast to **HxVec2Int** (p. 785).

```
309 {
310     switch(_tag._tag)
311     {
312     case SI:    return (HxVec2Int) *(HxScalarInt *) &_val[0];
313     case SD:    return (HxVec2Int) *(HxScalarDouble *) &_val[0];
```

```
314     case V2I:   return (HxVec2Int) *(HxVec2Int *) &_val[0];
315     case V2D:   return (HxVec2Int) *(HxVec2Double *) &_val[0];
316     case V3I:   return (HxVec2Int) *(HxVec3Int *) &_val[0];
317     case V3D:   return (HxVec2Int) *(HxVec3Double *) &_val[0];
318     case CPL:   return (HxVec2Int) *(HxComplex *) &_val[0];
319     default:    return (HxVec2Int) *(HxVec2Int *) &_val[0];
320     }
321 }
```

6.321.4.12 HxValue::operator HxVec2Double () const [inline]

Cast to **HxVec2Double** (p. 766).

```
325 {
326     switch(_tag._tag)
327     {
328     case SI:    return (HxVec2Double) *(HxScalarInt *) &_val[0];
329     case SD:    return (HxVec2Double) *(HxScalarDouble *) &_val[0];
330     case V2I:   return (HxVec2Double) *(HxVec2Int *) &_val[0];
331     case V2D:   return (HxVec2Double) *(HxVec2Double *) &_val[0];
332     case V3I:   return (HxVec2Double) *(HxVec3Int *) &_val[0];
333     case V3D:   return (HxVec2Double) *(HxVec3Double *) &_val[0];
334     case CPL:   return (HxVec2Double) *(HxComplex *) &_val[0];
335     default:    return (HxVec2Double) *(HxVec2Double *) &_val[0];
336     }
337 }
```

6.321.4.13 HxValue::operator HxVec3Int () const [inline]

Cast to **HxVec3Int** (p. 824).

```
341 {
342     switch(_tag._tag)
343     {
344     case SI:    return (HxVec3Int) *(HxScalarInt *) &_val[0];
345     case SD:    return (HxVec3Int) *(HxScalarDouble *) &_val[0];
346     case V2I:   return (HxVec3Int) *(HxVec2Int *) &_val[0];
347     case V2D:   return (HxVec3Int) *(HxVec2Double *) &_val[0];
348     case V3I:   return (HxVec3Int) *(HxVec3Int *) &_val[0];
349     case V3D:   return (HxVec3Int) *(HxVec3Double *) &_val[0];
350     case CPL:   return (HxVec3Int) *(HxComplex *) &_val[0];
351     default:    return (HxVec3Int) *(HxVec3Int *) &_val[0];
352     }
353 }
```

6.321.4.14 HxValue::operator HxVec3Double () const [inline]

Cast to **HxVec3Double** (p. 804).

```
357 {
358     switch(_tag._tag)
359     {
360     case SI:    return (HxVec3Double) *(HxScalarInt *) &_val[0];
361     case SD:    return (HxVec3Double) *(HxScalarDouble *) &_val[0];
362     case V2I:   return (HxVec3Double) *(HxVec2Int *) &_val[0];
```

```
363     case V2D:   return (HxVec3Double) *(HxVec2Double *) &_val[0];
364     case V3I:   return (HxVec3Double) *(HxVec3Int *) &_val[0];
365     case V3D:   return (HxVec3Double) *(HxVec3Double *) &_val[0];
366     case CPL:   return (HxVec3Double) *(HxComplex *) &_val[0];
367     default:    return (HxVec3Double) *(HxVec3Double *) &_val[0];
368     }
369 }
```

6.321.4.15 HxValue::operator HxComplex () const [inline]

Cast to **HxComplex** (p. 239).

```
373 {
374     switch(_tag._tag)
375     {
376     case SI:    return (HxComplex) *(HxScalarInt *) &_val[0];
377     case SD:    return (HxComplex) *(HxScalarDouble *) &_val[0];
378     case V2I:   return (HxComplex) *(HxVec2Int *) &_val[0];
379     case V2D:   return (HxComplex) *(HxVec2Double *) &_val[0];
380     case V3I:   return (HxComplex) *(HxVec3Int *) &_val[0];
381     case V3D:   return (HxComplex) *(HxVec3Double *) &_val[0];
382     case CPL:   return (HxComplex) *(HxComplex *) &_val[0];
383     default:    return (HxComplex) *(HxComplex *) &_val[0];
384     }
385 }
```

The documentation for this class was generated from the following files:

- **HxValue.h**
- HxValue.c

6.322 HxValueList Class Reference

Class definition for list of **HxValue** (p. 757)'s.

`#include <HxValueList.h>`

Public Types

- typedef std::back_insert_iterator< HxValueList > **back_insert_iterator**

  back inserter.

Public Methods

- HxValueList & **operator<<** (const **HxValue** &)

  Add value to the list.

- void **eraseAll** ()

  Remove all values from the list.

6.322.1 Detailed Description

Class definition for list of **HxValue** (p. 757)'s.

Specialization of list from STL.

6.322.2 Member Typedef Documentation

6.322.2.1 typedef std::back_insert_iterator<HxValueList> HxValueList::back_insert_iterator

back inserter.

6.322.3 Member Function Documentation

6.322.3.1 HxValueList & HxValueList::operator<< (const HxValue & *s*) [inline]

Add value to the list.

```
48 {
49     push_back(s);
50     return *this;
51 }
```

6.322.3.2 void HxValueList::eraseAll () [inline]

Remove all values from the list.

```
55 {
56     erase(begin(), end());
57 }
```

The documentation for this class was generated from the following file:

- **HxValueList.h**

6.323 HxVec2Double Class Reference

Class definition vector of 2 doubles.

`#include <HxVec2Double.h>`

Constructors

- **HxVec2Double** ()

  Default constructor.

- **HxVec2Double** (double x, double y)

  Conversion from native type.

- **HxVec2Double** (const HxVec2Double &rhs)
  Copy constructor.

Inquiry

- int **dim** () const
  Dimensionality.
- double **x** () const
  Value of first element.
- double **y** () const
  Value of second element.
- double **getValue** (int dimension) const
  Element in given dimension.
- void **setValue** (int dimension, double value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 677).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 785).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 824).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 804).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxVec2Double &v) const
  Equal.
- int **operator!=** (const HxVec2Double &v) const

Not equal.

- int **operator<** (const HxVec2Double &v) const
  Less than.
- int **operator<=** (const HxVec2Double &v) const
  Less equal.
- int **operator>** (const HxVec2Double &v) const
  Greater than.
- int **operator>=** (const HxVec2Double &v) const
  Greater equal.
- const HxVec2Double **SMALL_VAL** = HxVec2Double(0, 0)
  A small value w.r.t to the comparison operators "<" and ">".
- const HxVec2Double **LARGE_VAL** = HxVec2Double(1e300, 1e300)
  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxVec2Double **operator-** () const
  Negation.
- HxVec2Double **complement** () const
  Complement.
- HxVec2Double **abs** () const
  Absolute value.
- HxVec2Double **ceil** () const
  Ceiling.
- HxVec2Double **floor** () const
  Floor.
- HxVec2Double **round** () const
  Round.
- **HxScalarDouble sum** () const
  Sum.
- **HxScalarDouble product** () const
  Product.
- **HxScalarDouble min** () const

Minimum.

- **HxScalarDouble max** () const
  Maximum.
- **HxScalarDouble norm1** () const
  L1 norm.
- **HxScalarDouble norm2** () const
  L2 norm.
- **HxScalarDouble normInf** () const
  L infinity norm.
- HxVec2Double **sqrt** () const
  Square root.
- HxVec2Double **sin** () const
  Sine.
- HxVec2Double **cos** () const
  Cosine.
- HxVec2Double **tan** () const
  Tangent.
- HxVec2Double **asin** () const
  Arc sine.
- HxVec2Double **acos** () const
  Arc cosine.
- HxVec2Double **atan** () const
  Arc tangent.
- **HxScalarDouble atan2** () const
  Arc tangent.
- HxVec2Double **sinh** () const
  Hyperbolic sine.
- HxVec2Double **cosh** () const
  Hyperbolic cosine.
- HxVec2Double **tanh** () const
  Hyperbolic tangent.
- HxVec2Double **exp** () const
  Exponent.

- HxVec2Double **log** () const
  Natural logarithm.
- HxVec2Double **log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxVec2Double & **operator+=** (const HxVec2Double &v)
  Addition and assignment.
- HxVec2Double & **operator-=** (const HxVec2Double &v)
  Subtraction and assignment.
- HxVec2Double & **operator** *= (const HxVec2Double &v)
  Multiplication and assignment.
- HxVec2Double & **operator/=** (const HxVec2Double &v)
  Division and assignment.
- HxVec2Double **min** (const HxVec2Double &v) const
  Minimum.
- HxVec2Double & **minAssign** (const HxVec2Double &v)
  Minimum and assignment.
- HxVec2Double **max** (const HxVec2Double &v) const
  Maximum.
- HxVec2Double & **maxAssign** (const HxVec2Double &v)
  Maximum and assignment.
- HxVec2Double **inf** (const HxVec2Double &v) const
  Infimum.
- HxVec2Double & **infAssign** (const HxVec2Double &v)
  Infimum and assignment.
- HxVec2Double **sup** (const HxVec2Double &v) const
  Supremum.
- HxVec2Double & **supAssign** (const HxVec2Double &v)
  Supremum and assignment.
- HxVec2Double **pow** (const HxVec2Double &v) const
  Power.

- HxVec2Double **mod** (const HxVec2Double &v) const

  Modulo.

- HxVec2Double **and** (const HxVec2Double &v) const

  And.

- HxVec2Double **or** (const HxVec2Double &v) const

  Or.

- HxVec2Double **xor** (const HxVec2Double &v) const

  Xor.

- HxVec2Double **leftShift** (const HxVec2Double &v) const

  Left shift.

- HxVec2Double **rightShift** (const HxVec2Double &v) const

  Right shift.

- **HxScalarDouble dot** (const HxVec2Double &v) const

  Dot product.

- HxVec2Double **cross** (const HxVec2Double &v) const

  Cross product.

- HxVec2Double **operator+** (const HxVec2Double &v1, const HxVec2Double &v2)

  Addition.

- HxVec2Double **operator-** (const HxVec2Double &v1, const HxVec2Double &v2)

  Subtraction.

- HxVec2Double **operator** $*$ (const HxVec2Double &v1, const HxVec2Double &v2)

  Multiplication.

- HxVec2Double **operator/** (const HxVec2Double &v1, const HxVec2Double &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void * **operator new** (size_t, void *=0)
- HxVec2Double & **operator=** (const HxVec2Double &rhs)

6.323.1 Detailed Description

Class definition vector of 2 doubles.

6.323.2 Constructor & Destructor Documentation

6.323.2.1 HxVec2Double::HxVec2Double () [inline]

Default constructor.

```
323 {
324 }
```

6.323.2.2 HxVec2Double::HxVec2Double (double *x*, double *y*) [inline]

Conversion from native type.

```
328 {
329     _values[0] = x;
330     _values[1] = y;
331 }
```

6.323.2.3 HxVec2Double::HxVec2Double (const HxVec2Double & *v*) [inline]

Copy constructor.

```
335 {
336     _values[0] = v._values[0];
337     _values[1] = v._values[1];
338 }
```

6.323.3 Member Function Documentation

6.323.3.1 int HxVec2Double::dim () const [inline]

Dimensionality.

```
356 {
357     return 2;
358 }
```

6.323.3.2 double HxVec2Double::x () const [inline]

Value of first element.

```
362 {
363     return _values[0];
364 }
```

6.323.3.3 double HxVec2Double::y () const [inline]

Value of second element.

```
368 {
369     return _values[1];
370 }
```

6.323.3.4 double HxVec2Double::getValue (int *dim*) const [inline]

Element in given dimension.

```
374 {
375     return _values[dim - 1];
376 }
```

6.323.3.5 HxVec2Double::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
26 {
27     return (int) _values[0];
28 }
```

6.323.3.6 HxVec2Double::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
31 {
32     return _values[0];
33 }
```

6.323.3.7 HxVec2Double::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
36 {
37     return HxVec2Int(int(_values[0]), int(_values[1]));
38 }
```

6.323.3.8 HxVec2Double::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
41 {
42     return HxVec3Int(int(_values[0]), int(_values[1]), 0);
43 }
```

6.323.3.9 HxVec2Double::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
46 {
47     return HxVec3Double(_values[0], _values[1], 0);
48 }
```

6.323.3.10 HxVec2Double::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
51 {
52     return HxComplex(_values[0], _values[1]);
53 }
```

6.323.3.11 int HxVec2Double::operator== (const HxVec2Double & *v*) const [inline]

Equal.

```
386 {
387     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
388 }
```

6.323.3.12 int HxVec2Double::operator!= (const HxVec2Double & *v*) const [inline]

Not equal.

```
392 {
393     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
394 }
```

6.323.3.13 int HxVec2Double::operator< (const HxVec2Double & *v*) const [inline]

Less than.

```
398 {
399     return (fabs(_values[0]) + fabs(_values[1])) <
400            (fabs(v._values[0]) + fabs(v._values[1]));
401 }
```

6.323.3.14 int HxVec2Double::operator<= (const HxVec2Double & *v*) const [inline]

Less equal.

```
405 {
406     return (fabs(_values[0]) + fabs(_values[1])) <=
407            (fabs(v._values[0]) + fabs(v._values[1]));
408 }
```

6.323.3.15 int HxVec2Double::operator> (const HxVec2Double & *v*) const [inline]

Greater than.

```
412 {
413     return (fabs(_values[0]) + fabs(_values[1])) >
414            (fabs(v._values[0]) + fabs(v._values[1]));
415 }
```

6.323.3.16 int HxVec2Double::operator>= (const HxVec2Double & *v*) const [inline]

Greater equal.

```
419 {
420     return (fabs(_values[0]) + fabs(_values[1])) >=
421            (fabs(v._values[0]) + fabs(v._values[1]));
422 }
```

6.323.3.17 HxVec2Double HxVec2Double::operator- () const [inline]

Negation.

```
426 {
427     return HxVec2Double(-_values[0], -_values[1]);
428 }
```

6.323.3.18 HxVec2Double HxVec2Double::complement () const [inline]

Complement.

```
432 {
433     return HxVec2Double(-_values[0], -_values[1]);
434 }
```

6.323.3.19 HxVec2Double HxVec2Double::abs () const [inline]

Absolute value.

```
438 {
439     return HxVec2Double(fabs(_values[0]), fabs(_values[1]));
440 }
```

6.323.3.20 HxVec2Double HxVec2Double::ceil () const [inline]

Ceiling.

```
444 {
445     return HxVec2Double(::ceil(_values[0]), ::ceil(_values[1]));
446 }
```

6.323.3.21 HxVec2Double HxVec2Double::floor () const [inline]

Floor.

```
450 {
451     return HxVec2Double(::floor(_values[0]), ::floor(_values[1]));
452 }
```

6.323.3.22 HxVec2Double HxVec2Double::round () const [inline]

Round.

```
456 {
457     return HxVec2Double((int) (_values[0] + ((_values[0] >= 0) ? 0.5 : -0.5)),
458                         (int) (_values[1] + ((_values[1] >= 0) ? 0.5 : -0.5)));
459 }
```

6.323.3.23 HxScalarDouble HxVec2Double::sum () const [inline]

Sum.

```
710 {
711     return _values[0] + _values[1];
712 }
```

6.323.3.24 HxScalarDouble HxVec2Double::product () const [inline]

Product.

```
716 {
717     return _values[0] * _values[1];
718 }
```

6.323.3.25 HxScalarDouble HxVec2Double::min () const [inline]

Minimum.

```
722 {
723     return (_values[0] < _values[1]) ? _values[0] : _values[1];
724 }
```

6.323.3.26 HxScalarDouble HxVec2Double::max () const [inline]

Maximum.

```
728 {
729     return (_values[0] > _values[1]) ? _values[0] : _values[1];
730 }
```

6.323.3.27 HxScalarDouble HxVec2Double::norm1 () const

L1 norm.

```
57 {
58     return fabs(_values[0]) + fabs(_values[1]);
59 }
```

6.323.3.28 HxScalarDouble HxVec2Double::norm2 () const

L2 norm.

```
63 {
64     return ::sqrt(_values[0]*_values[0] + _values[1]*_values[1]);
65 }
```

6.323.3.29 HxScalarDouble HxVec2Double::normInf () const

L infinity norm.

```
69 {
70     return (fabs(_values[0]) > fabs(_values[1])) ? fabs(_values[0]) :
71                                                    fabs(_values[1]);
72 }
```

6.323.3.30 HxVec2Double HxVec2Double::sqrt () const [inline]

Square root.

```
463 {
464     return HxVec2Double(::sqrt(_values[0]), ::sqrt(_values[1]));
465 }
```

6.323.3.31 HxVec2Double HxVec2Double::sin () const [inline]

Sine.

```
469 {
470     return HxVec2Double(::sin(_values[0]), ::sin(_values[1]));
471 }
```

6.323.3.32 HxVec2Double HxVec2Double::cos () const [inline]

Cosine.

```
475 {
476     return HxVec2Double(::cos(_values[0]), ::cos(_values[1]));
477 }
```

6.323.3.33 HxVec2Double HxVec2Double::tan () const [inline]

Tangent.

```
481 {
482     return HxVec2Double(::tan(_values[0]), ::tan(_values[1]));
483 }
```

6.323.3.34 HxVec2Double HxVec2Double::asin () const [inline]

Arc sine.

```
487 {
488     return HxVec2Double(::asin(_values[0]), ::asin(_values[1]));
489 }
```

6.323.3.35 HxVec2Double HxVec2Double::acos () const [inline]

Arc cosine.

```
493 {
494     return HxVec2Double(::acos(_values[0]), ::acos(_values[1]));
495 }
```

6.323.3.36 HxVec2Double HxVec2Double::atan () const [inline]

Arc tangent.

```
499 {
500     return HxVec2Double(::atan(_values[0]), ::atan(_values[1]));
501 }
```

6.323.3.37 HxScalarDouble HxVec2Double::atan2 () const

Arc tangent.

```
76 {
77     return HxScalarDouble(::atan2(_values[0], _values[1]));
78 }
```

6.323.3.38 HxVec2Double HxVec2Double::sinh () const [inline]

Hyperbolic sine.

```
505 {
506     return HxVec2Double(::sinh(_values[0]), ::sinh(_values[1]));
507 }
```

6.323.3.39 HxVec2Double HxVec2Double::cosh () const [inline]

Hyperbolic cosine.

```
511 {
512     return HxVec2Double(::cosh(_values[0]), ::cosh(_values[1]));
513 }
```

6.323.3.40 HxVec2Double HxVec2Double::tanh () const [inline]

Hyperbolic tangent.

```
517 {
518     return HxVec2Double(::tanh(_values[0]), ::tanh(_values[1]));
519 }
```

6.323.3.41 HxVec2Double HxVec2Double::exp () const [inline]

Exponent.

```
523 {
524     return HxVec2Double(::exp(_values[0]), ::exp(_values[1]));
525 }
```

6.323.3.42 HxVec2Double HxVec2Double::log () const [inline]

Natural logarithm.

```
529 {
530     return HxVec2Double(::log(_values[0]), ::log(_values[1]));
531 }
```

6.323.3.43 HxVec2Double HxVec2Double::log10 () const [inline]

Base 10 logarithm.

```
535 {
536     return HxVec2Double(::log10(_values[0]), ::log10(_values[1]));
537 }
```

6.323.3.44 HxVec2Double & HxVec2Double::operator+= (const HxVec2Double & *v*) [inline]

Addition and assignment.

```
541 {
542     _values[0] += v._values[0];
543     _values[1] += v._values[1];
544     return *this;
545 }
```

6.323.3.45 HxVec2Double & HxVec2Double::operator-= (const HxVec2Double & *v*) [inline]

Subtraction and assignment.

```
549 {
550     _values[0] -= v._values[0];
551     _values[1] -= v._values[1];
552     return *this;
553 }
```

6.323.3.46 HxVec2Double & HxVec2Double::operator *= (const HxVec2Double & *v*) [inline]

Multiplication and assignment.

```
557 {
558     _values[0] *= v._values[0];
559     _values[1] *= v._values[1];
560     return *this;
561 }
```

6.323.3.47 HxVec2Double & HxVec2Double::operator/= (const HxVec2Double & *v*) [inline]

Division and assignment.

```
565 {
566     _values[0] /= v._values[0];
567     _values[1] /= v._values[1];
568     return *this;
569 }
```

6.323.3.48 HxVec2Double HxVec2Double::min (const HxVec2Double & *v*) const [inline]

Minimum.

```
601 {
602     return (operator<(v)) ? (*this) : v;
603 }
```

6.323.3.49 HxVec2Double & HxVec2Double::minAssign (const HxVec2Double & *v*) [inline]

Minimum and assignment.

```
607 {
608     if (operator<(v))
609         return *this;
610     operator=(v);
611     return *this;
612 }
```

6.323.3.50 HxVec2Double HxVec2Double::max (const HxVec2Double & *v*) const [inline]

Maximum.

```
616 {
617     return (operator>(v)) ? (*this) : v;
618 }
```

6.323.3.51 HxVec2Double & HxVec2Double::maxAssign (const HxVec2Double & *v*) [inline]

Maximum and assignment.

```
622 {
623     if (operator>(v))
624         return *this;
625     operator=(v);
626     return *this;
627 }
```

6.323.3.52 HxVec2Double HxVec2Double::inf (const HxVec2Double & *v*) const [inline]

Infimum.

```
631 {
632     return HxVec2Double((_values[0] < v._values[0]) ? _values[0] : v._values[0],
633                         (_values[1] < v._values[1]) ? _values[1] : v._values[1]);
634 }
```

6.323.3.53 HxVec2Double & HxVec2Double::infAssign (const HxVec2Double & *v*) [inline]

Infimum and assignment.

```
638 {
639     _values[0] = (_values[0] < v._values[0]) ? _values[0] : v._values[0];
640     _values[1] = (_values[1] < v._values[1]) ? _values[1] : v._values[1];
641     return *this;
642 }
```

6.323.3.54 HxVec2Double HxVec2Double::sup (const HxVec2Double & *v*) const [inline]

Supremum.

```
646 {
647     return HxVec2Double((_values[0] > v._values[0]) ? _values[0] : v._values[0],
648                         (_values[1] > v._values[1]) ? _values[1] : v._values[1]);
649 }
```

6.323.3.55 HxVec2Double & HxVec2Double::supAssign (const HxVec2Double & *v*) [inline]

Supremum and assignment.

```
653 {
654     _values[0] = (_values[0] > v._values[0]) ? _values[0] : v._values[0];
655     _values[1] = (_values[1] > v._values[1]) ? _values[1] : v._values[1];
656     return *this;
657 }
```

6.323.3.56 HxVec2Double HxVec2Double::pow (const HxVec2Double & *v*) const [inline]

Power.

```
661 {
662     return HxVec2Double(::pow(_values[0], v._values[0]),
663                         ::pow(_values[1], v._values[1]));
664 }
```

6.323.3.57 HxVec2Double HxVec2Double::mod (const HxVec2Double & *v*) const [inline]

Modulo.

```
668 {
669     return (*this);
670 }
```

6.323.3.58 HxVec2Double HxVec2Double::and (const HxVec2Double & *v*) const [inline]

And.

```
674 {
675     return (*this);
676 }
```

6.323.3.59 HxVec2Double HxVec2Double::or (const HxVec2Double & *v*) const [inline]

Or.

```
680 {
681     return (*this);
682 }
```

6.323.3.60 HxVec2Double HxVec2Double::xor (const HxVec2Double & *v*) const [inline]

Xor.

```
686 {
687     return (*this);
688 }
```

6.323.3.61 HxVec2Double HxVec2Double::leftShift (const HxVec2Double & *v*) const [inline]

Left shift.

```
692 {
693     return (*this);
694 }
```

6.323.3.62 HxVec2Double HxVec2Double::rightShift (const HxVec2Double & *v*) const [inline]

Right shift.

```
698 {
699     return (*this);
700 }
```

6.323.3.63 HxScalarDouble HxVec2Double::dot (const HxVec2Double & *v*) const

Dot product.

```
82 {
83     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
84 }
```

6.323.3.64 HxVec2Double HxVec2Double::cross (const HxVec2Double & *v*) const [inline]

Cross product.

```
704 {
705     return HxVec2Double(0, 0);
706 }
```

6.323.3.65 STD_OSTREAM & HxVec2Double::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
88 {
89     return os << "(" << _values[0] << "," << _values[1] << ")";
90 }
```

6.323.3.66 HxString HxVec2Double::toString () const

Value as a string.

```
93                              {
94     return HxString("(") + makeString(_values[0]) + ", "
95             + makeString(_values[1]) + ")";
96 }
```

6.323.4 Friends And Related Function Documentation

6.323.4.1 HxVec2Double operator+ (const HxVec2Double & *v1*, const HxVec2Double & *v2*) [friend]

Addition.

```
573 {
574     return HxVec2Double(v1._values[0] + v2._values[0],
575                         v1._values[1] + v2._values[1]);
576 }
```

6.323.4.2 HxVec2Double operator- (const HxVec2Double & *v1*, const HxVec2Double & *v2*) [friend]

Subtraction.

```
580 {
581     return HxVec2Double(v1._values[0] - v2._values[0],
582                         v1._values[1] - v2._values[1]);
583 }
```

6.323.4.3 HxVec2Double operator * (const HxVec2Double & *v1*, const HxVec2Double & *v2*) [friend]

Multiplication.

```
587 {
588     return HxVec2Double(v1._values[0] * v2._values[0],
589                         v1._values[1] * v2._values[1]);
590 }
```

6.323.4.4 HxVec2Double operator/ (const HxVec2Double & *v1*, const HxVec2Double & *v2*) [friend]

Division.

```
594 {
595     return HxVec2Double(v1._values[0] / v2._values[0],
596                         v1._values[1] / v2._values[1]);
597 }
```

6.323.5 Member Data Documentation

6.323.5.1 const HxVec2Double HxVec2Double::SMALL_VAL = HxVec2Double(0, 0) [static]

A small value w.r.t to the comparison operators "<" and ">".

Not actually the minimum to avoid overflow.

6.323.5.2 const HxVec2Double HxVec2Double::LARGE_VAL = HxVec2Double(1e300, 1e300) [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxVec2Double.h**
- HxVec2Double.c

6.324 HxVec2Int Class Reference

Class definition vector of 2 integers.

`#include <HxVec2Int.h>`

Constructors

- **HxVec2Int** ()

  Default constructor.

- **HxVec2Int** (int x, int y)

  Conversion from native type.

- **HxVec2Int** (const HxVec2Int &v)

  Copy constructor.

Inquiry

- int **dim** () const

  Dimensionality.

- int **x** () const

  Value of first element.

- int **y** () const

  Value of second element.

- int **getValue** (int dimension) const

Element in given dimension.

- void **setValue** (int dimension, int value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 677).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 766).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 824).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 804).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxVec2Int &v) const
  Equal.
- int **operator!=** (const HxVec2Int &v) const
  Not equal.
- int **operator<** (const HxVec2Int &v) const
  Less than.
- int **operator<=** (const HxVec2Int &v) const
  Less equal.
- int **operator>** (const HxVec2Int &v) const
  Greater than.
- int **operator>=** (const HxVec2Int &v) const
  Greater equal.
- const HxVec2Int **SMALL_VAL** = HxVec2Int(0, 0)
  A small value w.r.t to the comparison operators "<" and ">".

- const HxVec2Int **LARGE_VAL** = HxVec2Int(200000000, 200000000)

  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxVec2Int **operator-** () const

  Negation.

- HxVec2Int **complement** () const

  Complement.

- HxVec2Int **abs** () const

  Absolute value.

- HxVec2Int **ceil** () const

  Ceiling.

- HxVec2Int **floor** () const

  Floor.

- HxVec2Int **round** () const

  Round.

- **HxScalarInt sum** () const

  Sum.

- **HxScalarInt product** () const

  Product.

- **HxScalarInt min** () const

  Minimum.

- **HxScalarInt max** () const

  Maximum.

- **HxScalarInt norm1** () const

  L1 norm.

- **HxScalarDouble norm2** () const

  L2 norm.

- **HxScalarInt normInf** () const

  L infinity norm.

- **HxVec2Double sqrt** () const

Square root.

- **HxVec2Double sin** () const
  Sine.
- **HxVec2Double cos** () const
  Cosine.
- **HxVec2Double tan** () const
  Tangent.
- **HxVec2Double asin** () const
  Arc sine.
- **HxVec2Double acos** () const
  Arc cosine.
- **HxVec2Double atan** () const
  Arc tangent.
- **HxScalarDouble atan2** () const
  Arc tangent.
- **HxVec2Double sinh** () const
  Hyperbolic sine.
- **HxVec2Double cosh** () const
  Hyperbolic cosine.
- **HxVec2Double tanh** () const
  Hyperbolic tangent.
- **HxVec2Double exp** () const
  Exponent.
- **HxVec2Double log** () const
  Natural logarithm.
- **HxVec2Double log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxVec2Int & **operator+=** (const HxVec2Int &v)
  Addition and assignment.
- HxVec2Int & **operator-=** (const HxVec2Int &v)

Subtraction and assignment.

- HxVec2Int & **operator** *= (const HxVec2Int &v)
  Multiplication and assignment.
- HxVec2Int & **operator/=** (const HxVec2Int &v)
  Division and assignment.
- HxVec2Int **min** (const HxVec2Int &v) const
  Minimum.
- HxVec2Int & **minAssign** (const HxVec2Int &v)
  Minimum and assignment.
- HxVec2Int **max** (const HxVec2Int &v) const
  Maximum.
- HxVec2Int & **maxAssign** (const HxVec2Int &v)
  Maximum and assignment.
- HxVec2Int **inf** (const HxVec2Int &v) const
  Infimum.
- HxVec2Int & **infAssign** (const HxVec2Int &v)
  Infimum and assignment.
- HxVec2Int **sup** (const HxVec2Int &v) const
  Supremum.
- HxVec2Int & **supAssign** (const HxVec2Int &v)
  Supremum and assignment.
- HxVec2Int **pow** (const HxVec2Int &v) const
  Power.
- HxVec2Int **mod** (const HxVec2Int &v) const
  Modulo.
- HxVec2Int **and** (const HxVec2Int &v) const
  And.
- HxVec2Int **or** (const HxVec2Int &v) const
  Or.
- HxVec2Int **xor** (const HxVec2Int &v) const
  Xor.
- HxVec2Int **leftShift** (const HxVec2Int &v) const
  Left shift.

- HxVec2Int **rightShift** (const HxVec2Int &v) const
  Right shift.
- **HxScalarInt dot** (const HxVec2Int &v) const
  Dot product.
- HxVec2Int **cross** (const HxVec2Int &v) const
  Cross product.
- HxVec2Int **operator+** (const HxVec2Int &v1, const HxVec2Int &v2)
  Addition.
- HxVec2Int **operator-** (const HxVec2Int &v1, const HxVec2Int &v2)
  Subtraction.
- HxVec2Int **operator** * (const HxVec2Int &v1, const HxVec2Int &v2)
  Multiplication.
- HxVec2Int **operator/** (const HxVec2Int &v1, const HxVec2Int &v2)
  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const
  Print value on stream.
- **HxString toString** () const
  Value as a string.

Public Methods

- void * **operator new** (size_t, void *=0)

6.324.1 Detailed Description

Class definition vector of 2 integers.

6.324.2 Constructor & Destructor Documentation

6.324.2.1 HxVec2Int::HxVec2Int () [inline]

Default constructor.

```
319 {
320 }
```

6.324.2.2 HxVec2Int::HxVec2Int (int *x*, int *y*) [inline]

Conversion from native type.

```
324 {
325     _values[0] = x;
326     _values[1] = y;
327 }
```

6.324.2.3 HxVec2Int::HxVec2Int (const HxVec2Int & *v*) [inline]

Copy constructor.

```
331 {
332     _values[0] = v._values[0];
333     _values[1] = v._values[1];
334 }
```

6.324.3 Member Function Documentation

6.324.3.1 int HxVec2Int::dim () const [inline]

Dimensionality.

```
344 {
345     return 2;
346 }
```

6.324.3.2 int HxVec2Int::x () const [inline]

Value of first element.

```
350 {
351     return _values[0];
352 }
```

6.324.3.3 int HxVec2Int::y () const [inline]

Value of second element.

```
356 {
357     return _values[1];
358 }
```

6.324.3.4 int HxVec2Int::getValue (int *dim*) const [inline]

Element in given dimension.

```
362 {
363     return _values[dim - 1];
364 }
```

6.324.3.5 HxVec2Int::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
26 {
27     return _values[0];
28 }
```

6.324.3.6 HxVec2Int::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
31 {
32     return (double) _values[0];
33 }
```

6.324.3.7 HxVec2Int::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
36 {
37     return HxVec2Double(_values[0], _values[1]);
38 }
```

6.324.3.8 HxVec2Int::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
41 {
42     return HxVec3Int(_values[0], _values[1], 0);
43 }
```

6.324.3.9 HxVec2Int::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
46 {
47     return HxVec3Double(_values[0], _values[1], 0);
48 }
```

6.324.3.10 HxVec2Int::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
51 {
52     return HxComplex(_values[0], _values[1]);
53 }
```

6.324.3.11 int HxVec2Int::operator== (const HxVec2Int & *v*) const [inline]

Equal.

```
374 {
375     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
376 }
```

6.324.3.12 int HxVec2Int::operator!= (const HxVec2Int & *v*) const [inline]

Not equal.

```
380 {
381     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
382 }
```

6.324.3.13 int HxVec2Int::operator< (const HxVec2Int & *v*) const [inline]

Less than.

```
386 {
387     return (::abs(_values[0]) + ::abs(_values[1])) <
388            (::abs(v._values[0]) + ::abs(v._values[1]));
389 }
```

6.324.3.14 int HxVec2Int::operator<= (const HxVec2Int & *v*) const [inline]

Less equal.

```
393 {
394     return (::abs(_values[0]) + ::abs(_values[1])) <=
395            (::abs(v._values[0]) + ::abs(v._values[1]));
396 }
```

6.324.3.15 int HxVec2Int::operator> (const HxVec2Int & *v*) const [inline]

Greater than.

```
400 {
401     return (::abs(_values[0]) + ::abs(_values[1])) >
402            (::abs(v._values[0]) + ::abs(v._values[1]));
403 }
```

6.324.3.16 int HxVec2Int::operator>= (const HxVec2Int & *v*) const [inline]

Greater equal.

```
407 {
408     return (::abs(_values[0]) + ::abs(_values[1])) >=
409            (::abs(v._values[0]) + ::abs(v._values[1]));
410 }
```

6.324.3.17 HxVec2Int HxVec2Int::operator- () const [inline]

Negation.

```
414 {
415     return HxVec2Int(-_values[0], -_values[1]);
416 }
```

6.324.3.18 HxVec2Int HxVec2Int::complement () const [inline]

Complement.

```
420 {
421     return HxVec2Int(~_values[0], ~_values[1]);
422 }
```

6.324.3.19 HxVec2Int HxVec2Int::abs () const [inline]

Absolute value.

```
426 {
427     return HxVec2Int(::abs(_values[0]), ::abs(_values[1]));
428 }
```

6.324.3.20 HxVec2Int HxVec2Int::ceil () const [inline]

Ceiling.

```
432 {
433     return *this;
434 }
```

6.324.3.21 HxVec2Int HxVec2Int::floor () const [inline]

Floor.

```
438 {
439     return *this;
440 }
```

6.324.3.22 HxVec2Int HxVec2Int::round () const [inline]

Round.

```
444 {
445     return *this;
446 }
```

6.324.3.23 HxScalarInt HxVec2Int::sum () const [inline]

Sum.

```
625 {
626     return _values[0] + _values[1];
627 }
```

6.324.3.24 HxScalarInt HxVec2Int::product () const [inline]

Product.

```
631 {
632     return _values[0] * _values[1];
633 }
```

6.324.3.25 HxScalarInt HxVec2Int::min () const [inline]

Minimum.

```
637 {
638     return (_values[0] < _values[1]) ? _values[0] : _values[1];
639 }
```

6.324.3.26 HxScalarInt HxVec2Int::max () const [inline]

Maximum.

```
643 {
644     return (_values[0] > _values[1]) ? _values[0] : _values[1];
645 }
```

6.324.3.27 HxScalarInt HxVec2Int::norm1 () const

L1 norm.

```
57 {
58     return ::abs(_values[0]) + ::abs(_values[1]);
59 }
```

6.324.3.28 HxScalarDouble HxVec2Int::norm2 () const

L2 norm.

```
63 {
64     return ::sqrt(double(_values[0])*_values[0] + double(_values[1])*_values[1]);
65 }
```

6.324.3.29 HxScalarInt HxVec2Int::normInf () const

L infinity norm.

```
69 {
70     return (::abs(_values[0]) > ::abs(_values[1])) ? ::abs(_values[0]) :
71                                                      ::abs(_values[1]);
72 }
```

6.324.3.30 HxVec2Double HxVec2Int::sqrt () const

Square root.

```
76 {
77     return HxVec2Double(::sqrt(double(_values[0])), ::sqrt(double(_values[1])));
78 }
```

6.324.3.31 HxVec2Double HxVec2Int::sin () const

Sine.

```
82 {
83     return HxVec2Double(::sin(double(_values[0])), ::sin(double(_values[1])));
84 }
```

6.324.3.32 HxVec2Double HxVec2Int::cos () const

Cosine.

```
88 {
89     return HxVec2Double(::cos(double(_values[0])), ::cos(double(_values[1])));
90 }
```

6.324.3.33 HxVec2Double HxVec2Int::tan () const

Tangent.

```
94 {
95     return HxVec2Double(::tan(double(_values[0])), ::tan(double(_values[1])));
96 }
```

6.324.3.34 HxVec2Double HxVec2Int::asin () const

Arc sine.

```
100 {
101     return HxVec2Double(::asin(double(_values[0])), ::asin(double(_values[1])));
102 }
```

6.324.3.35 HxVec2Double HxVec2Int::acos () const

Arc cosine.

```
106 {
107     return HxVec2Double(::acos(double(_values[0])), ::acos(double(_values[1])));
108 }
```

6.324.3.36 HxVec2Double HxVec2Int::atan () const

Arc tangent.

```
112 {
113     return HxVec2Double(::atan(double(_values[0])), ::atan(double(_values[1])));
114 }
```

6.324.3.37 HxScalarDouble HxVec2Int::atan2 () const

Arc tangent.

```
118 {
119     return HxScalarDouble(::atan2(double(_values[0]), double(_values[1])));
120 }
```

6.324.3.38 HxVec2Double HxVec2Int::sinh () const

Hyperbolic sine.

```
124 {
125     return HxVec2Double(::sinh(double(_values[0])), ::sinh(double(_values[1])));
126 }
```

6.324.3.39 HxVec2Double HxVec2Int::cosh () const

Hyperbolic cosine.

```
130 {
131     return HxVec2Double(::cosh(double(_values[0])), ::cosh(double(_values[1])));
132 }
```

6.324.3.40 HxVec2Double HxVec2Int::tanh () const

Hyperbolic tangent.

```
136 {
137     return HxVec2Double(::tanh(double(_values[0])), ::tanh(double(_values[1])));
138 }
```

6.324.3.41 HxVec2Double HxVec2Int::exp () const

Exponent.

```
142 {
143     return HxVec2Double(::exp(double(_values[0])), ::exp(double(_values[1])));
144 }
```

6.324.3.42 HxVec2Double HxVec2Int::log () const

Natural logarithm.

```
148 {
149     return HxVec2Double(::log(double(_values[0])), ::log(double(_values[1])));
150 }
```

6.324.3.43 HxVec2Double HxVec2Int::log10 () const

Base 10 logarithm.

```
154 {
155     return HxVec2Double(::log10(double(_values[0])), ::log10(double(_values[1])));
156 }
```

6.324.3.44 HxVec2Int & HxVec2Int::operator+= (const HxVec2Int & *v*) [inline]

Addition and assignment.

```
450 {
451     _values[0] += v._values[0];
452     _values[1] += v._values[1];
453     return *this;
454 }
```

6.324.3.45 HxVec2Int & HxVec2Int::operator-= (const HxVec2Int & *v*) [inline]

Subtraction and assignment.

```
458 {
459     _values[0] -= v._values[0];
460     _values[1] -= v._values[1];
461     return *this;
462 }
```

6.324.3.46 HxVec2Int & HxVec2Int::operator *= (const HxVec2Int & *v*) [inline]

Multiplication and assignment.

```
466 {
467     _values[0] *= v._values[0];
468     _values[1] *= v._values[1];
469     return *this;
470 }
```

6.324.3.47 HxVec2Int & HxVec2Int::operator/= (const HxVec2Int & *v*) [inline]

Division and assignment.

```
474 {
475     _values[0] /= v._values[0];
476     _values[1] /= v._values[1];
477     return *this;
478 }
```

6.324.3.48 HxVec2Int HxVec2Int::min (const HxVec2Int & *v*) const [inline]

Minimum.

```
510 {
511     return (operator<(v)) ? (*this) : v;
512 }
```

6.324.3.49 HxVec2Int & HxVec2Int::minAssign (const HxVec2Int & *v*) [inline]

Minimum and assignment.

```
516 {
517     if (operator<(v))
518         return *this;
519     operator=(v);
520     return *this;
521 }
```

6.324.3.50 HxVec2Int HxVec2Int::max (const HxVec2Int & *v*) const [inline]

Maximum.

```
525 {
526     return (operator>(v)) ? (*this) : v;
527 }
```

6.324.3.51 HxVec2Int & HxVec2Int::maxAssign (const HxVec2Int & *v*) [inline]

Maximum and assignment.

```
531 {
532     if (operator>(v))
533         return *this;
534     operator=(v);
535     return *this;
536 }
```

6.324.3.52 HxVec2Int HxVec2Int::inf (const HxVec2Int & *v*) const [inline]

Infimum.

```
540 {
541     return HxVec2Int((_values[0] < v._values[0]) ? _values[0] : v._values[0],
542                      (_values[1] < v._values[1]) ? _values[1] : v._values[1]);
543 }
```

6.324.3.53 HxVec2Int & HxVec2Int::infAssign (const HxVec2Int & *v*) [inline]

Infimum and assignment.

```
547 {
548     _values[0] = (_values[0] < v._values[0]) ? _values[0] : v._values[0];
549     _values[1] = (_values[1] < v._values[1]) ? _values[1] : v._values[1];
550     return *this;
551 }
```

6.324.3.54 HxVec2Int HxVec2Int::sup (const HxVec2Int & *v*) const [inline]

Supremum.

```
555 {
556     return HxVec2Int((_values[0] > v._values[0]) ? _values[0] : v._values[0],
557                      (_values[1] > v._values[1]) ? _values[1] : v._values[1]);
558 }
```

6.324.3.55 HxVec2Int & HxVec2Int::supAssign (const HxVec2Int & *v*) [inline]

Supremum and assignment.

```
562 {
563     _values[0] = (_values[0] > v._values[0]) ? _values[0] : v._values[0];
564     _values[1] = (_values[1] > v._values[1]) ? _values[1] : v._values[1];
565     return *this;
566 }
```

6.324.3.56 HxVec2Int HxVec2Int::pow (const HxVec2Int & *v*) const [inline]

Power.

```
570 {
571     return HxVec2Int((int) (::pow(_values[0], v._values[0]) + 0.5),
572                      (int) (::pow(_values[1], v._values[1]) + 0.5));
573 }
```

6.324.3.57 HxVec2Int HxVec2Int::mod (const HxVec2Int & *v*) const [inline]

Modulo.

```
577 {
578     return HxVec2Int(_values[0] % v._values[0],
579                      _values[1] % v._values[1]);
580 }
```

6.324.3.58 HxVec2Int HxVec2Int::and (const HxVec2Int & *v*) const [inline]

And.

```
584 {
585     return HxVec2Int(_values[0] & v._values[0],
586                      _values[1] & v._values[1]);
587 }
```

6.324.3.59 HxVec2Int HxVec2Int::or (const HxVec2Int & *v*) const [inline]

Or.

```
591 {
592     return HxVec2Int(_values[0] | v._values[0],
593                      _values[1] | v._values[1]);
594 }
```

6.324.3.60 HxVec2Int HxVec2Int::xor (const HxVec2Int & *v*) const [inline]

Xor.

```
598 {
599     return HxVec2Int(_values[0] ^ v._values[0],
600                      _values[1] ^ v._values[1]);
601 }
```

6.324.3.61 HxVec2Int HxVec2Int::leftShift (const HxVec2Int & *v*) const [inline]

Left shift.

```
605 {
606     return HxVec2Int(_values[0] << v._values[0],
607                      _values[1] << v._values[1]);
608 }
```

6.324.3.62 HxVec2Int HxVec2Int::rightShift (const HxVec2Int & *v*) const [inline]

Right shift.

```
612 {
613     return HxVec2Int(_values[0] >> v._values[0],
614                      _values[1] >> v._values[1]);
615 }
```

6.324.3.63 HxScalarInt HxVec2Int::dot (const HxVec2Int & *v*) const

Dot product.

```
160 {
161     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
162 }
```

6.324.3.64 HxVec2Int HxVec2Int::cross (const HxVec2Int & *v*) const [inline]

Cross product.

```
619 {
620     return HxVec2Int(0, 0);
621 }
```

6.324.3.65 STD_OSTREAM & HxVec2Int::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
166 {
167     return os << "(" << _values[0] << "," << _values[1] << ")";
168 }
```

6.324.3.66 HxString HxVec2Int::toString () const

Value as a string.

```
171                             {
172     return HxString("(") + makeString(_values[0]) + ", "
173             + makeString(_values[1]) + ")";
174 }
```

6.324.4 Friends And Related Function Documentation

6.324.4.1 HxVec2Int operator+ (const HxVec2Int & *v1*, const HxVec2Int & *v2*) [friend]

Addition.

```
482 {
483     return HxVec2Int(v1._values[0] + v2._values[0],
484                      v1._values[1] + v2._values[1]);
485 }
```

6.324.4.2 HxVec2Int operator- (const HxVec2Int & *v1*, const HxVec2Int & *v2*) [friend]

Subtraction.

```
489 {
490     return HxVec2Int(v1._values[0] - v2._values[0],
491                      v1._values[1] - v2._values[1]);
492 }
```

6.324.4.3 HxVec2Int operator ∗ (const HxVec2Int & *v1*, const HxVec2Int & *v2*) [friend]

Multiplication.

```
496 {
497     return HxVec2Int(v1._values[0] * v2._values[0],
498                      v1._values[1] * v2._values[1]);
499 }
```

6.324.4.4 HxVec2Int operator/ (const HxVec2Int & *v1*, const HxVec2Int & *v2*) [friend]

Division.

```
503 {
504     return HxVec2Int(v1._values[0] / v2._values[0],
505                      v1._values[1] / v2._values[1]);
506 }
```

6.324.5 Member Data Documentation

6.324.5.1 const HxVec2Int HxVec2Int::SMALL_VAL = HxVec2Int(0, 0) [static]

A small value w.r.t to the comparison operators ”<” and ”>”.

Not actually the minimum to avoid overflow.

6.324.5.2 const HxVec2Int HxVec2Int::LARGE_VAL = HxVec2Int(200000000, 200000000) [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxVec2Int.h**
- HxVec2Int.c

6.325 HxVec3Double Class Reference

Class definition vector of 3 doubles.

`#include <HxVec3Double.h>`

Constructors

- **HxVec3Double** ()

  Default constructor.

- **HxVec3Double** (double x, double y, double z)

  Conversion from native type.

- **HxVec3Double** (const HxVec3Double &v)

  Copy constructor.

Inquiry

- int **dim** () const

  Dimensionality.

- double **x** () const

  Value of first element.

- double **y** () const

  Value of second element.

- double **z** () const

  Value of third element.

- double **getValue** (int dimension) const

  Element in given dimension.

- void **setValue** (int dimension, double value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 677).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 785).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 766).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 824).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxVec3Double &v) const
  Equal.
- int **operator!=** (const HxVec3Double &v) const
  Not equal.
- int **operator<** (const HxVec3Double &v) const
  Less than.
- int **operator<=** (const HxVec3Double &v) const
  Less equal.
- int **operator>** (const HxVec3Double &v) const
  Greater than.
- int **operator>=** (const HxVec3Double &v) const
  Greater equal.
- const HxVec3Double **SMALL_VAL** = HxVec3Double(0, 0, 0)
  A small value w.r.t to the comparison operators "<" and ">".
- const HxVec3Double **LARGE_VAL** = HxVec3Double(1e300, 1e300, 1e300)
  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxVec3Double **operator-** () const
  Negation.
- HxVec3Double **complement** () const
  Complement.
- HxVec3Double **abs** () const
  Absolute value.
- HxVec3Double **ceil** () const
  Ceiling.
- HxVec3Double **floor** () const
  Floor.
- HxVec3Double **round** () const
  Round.
- **HxScalarDouble sum** () const
  Sum.
- **HxScalarDouble product** () const
  Product.
- **HxScalarDouble min** () const
  Minimum.
- **HxScalarDouble max** () const
  Maximum.
- **HxScalarDouble norm1** () const
  L1 norm.
- **HxScalarDouble norm2** () const
  L2 norm.
- **HxScalarDouble normInf** () const
  L infinity norm.
- HxVec3Double **sqrt** () const
  Square root.
- HxVec3Double **sin** () const
  Sine.
- HxVec3Double **cos** () const

Cosine.

- HxVec3Double **tan** () const
  Tangent.
- HxVec3Double **asin** () const
  Arc sine.
- HxVec3Double **acos** () const
  Arc cosine.
- HxVec3Double **atan** () const
  Arc tangent.
- **HxScalarDouble atan2** () const
  Arc tangent.
- HxVec3Double **sinh** () const
  Hyperbolic sine.
- HxVec3Double **cosh** () const
  Hyperbolic cosine.
- HxVec3Double **tanh** () const
  Hyperbolic tangent.
- HxVec3Double **exp** () const
  Exponent.
- HxVec3Double **log** () const
  Natural logarithm.
- HxVec3Double **log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxVec3Double & **operator+=** (const HxVec3Double &v)
  Addition and assignment.
- HxVec3Double & **operator-=** (const HxVec3Double &v)
  Subtraction and assignment.
- HxVec3Double & **operator** $*=$ (const HxVec3Double &v)
  Multiplication and assignment.
- HxVec3Double & **operator/=** (const HxVec3Double &v)

Division and assignment.

- HxVec3Double **min** (const HxVec3Double &v) const
  Minimum.
- HxVec3Double & **minAssign** (const HxVec3Double &v)
  Minimum and assignment.
- HxVec3Double **max** (const HxVec3Double &v) const
  Maximum.
- HxVec3Double & **maxAssign** (const HxVec3Double &v)
  Maximum and assignment.
- HxVec3Double **inf** (const HxVec3Double &v) const
  Infimum.
- HxVec3Double & **infAssign** (const HxVec3Double &v)
  Infimum and assignment.
- HxVec3Double **sup** (const HxVec3Double &v) const
  Supremum.
- HxVec3Double & **supAssign** (const HxVec3Double &v)
  Supremum and assignment.
- HxVec3Double **pow** (const HxVec3Double &v) const
  Power.
- HxVec3Double **mod** (const HxVec3Double &v) const
  Modulo.
- HxVec3Double **and** (const HxVec3Double &v) const
  And.
- HxVec3Double **or** (const HxVec3Double &v) const
  Or.
- HxVec3Double **xor** (const HxVec3Double &v) const
  Xor.
- HxVec3Double **leftShift** (const HxVec3Double &v) const
  Left shift.
- HxVec3Double **rightShift** (const HxVec3Double &v) const
  Right shift.
- **HxScalarDouble dot** (const HxVec3Double &v) const
  Dot product.

- HxVec3Double **cross** (const HxVec3Double &v) const

  Cross product.

- HxVec3Double **operator+** (const HxVec3Double &v1, const HxVec3Double &v2)

  Addition.

- HxVec3Double **operator-** (const HxVec3Double &v1, const HxVec3Double &v2)

  Subtraction.

- HxVec3Double **operator** * (const HxVec3Double &v1, const HxVec3Double &v2)

  Multiplication.

- HxVec3Double **operator/** (const HxVec3Double &v1, const HxVec3Double &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void * **operator new** (size_t, void *=0)

6.325.1 Detailed Description

Class definition vector of 3 doubles.

6.325.2 Constructor & Destructor Documentation

6.325.2.1 HxVec3Double::HxVec3Double () [inline]

Default constructor.

```
323 {
324 }
```

6.325.2.2 HxVec3Double::HxVec3Double (double *x*, double *y*, double *z*) [inline]

Conversion from native type.

```
328 {
329     _values[0] = x;
330     _values[1] = y;
331     _values[2] = z;
332 }
```

6.325.2.3 HxVec3Double::HxVec3Double (const HxVec3Double & *v*) [inline]

Copy constructor.

```
336 {
337     _values[0] = v._values[0];
338     _values[1] = v._values[1];
339     _values[2] = v._values[2];
340 }
```

6.325.3 Member Function Documentation

6.325.3.1 int HxVec3Double::dim () const [inline]

Dimensionality.

```
350 {
351     return 3;
352 }
```

6.325.3.2 double HxVec3Double::x () const [inline]

Value of first element.

```
356 {
357     return _values[0];
358 }
```

6.325.3.3 double HxVec3Double::y () const [inline]

Value of second element.

```
362 {
363     return _values[1];
364 }
```

6.325.3.4 double HxVec3Double::z () const [inline]

Value of third element.

```
368 {
369     return _values[2];
370 }
```

6.325.3.5 double HxVec3Double::getValue (int *dimension*) const [inline]

Element in given dimension.

```
374 {
375     return _values[dimension - 1];
376 }
```

6.325.3.6 HxVec3Double::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
28 {
29     return (int) _values[0];
30 }
```

6.325.3.7 HxVec3Double::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
33 {
34     return _values[0];
35 }
```

6.325.3.8 HxVec3Double::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
38 {
39     return HxVec2Int(int(_values[0]), int(_values[1]));
40 }
```

6.325.3.9 HxVec3Double::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
43 {
44     return HxVec2Double(_values[0], _values[1]);
45 }
```

6.325.3.10 HxVec3Double::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 824).

```
48 {
49     return HxVec3Int(int(_values[0]), int(_values[1]), int(_values[2]));
50 }
```

6.325.3.11 HxVec3Double::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
53 {
54     return HxComplex(_values[0], _values[1]);
55 }
```

6.325.3.12 int HxVec3Double::operator== (const HxVec3Double & *v*) const [inline]

Equal.

```
386 {
387     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]) &&
388            (_values[2] == v._values[2]);
389 }
```

6.325.3.13 int HxVec3Double::operator!= (const HxVec3Double & *v*) const [inline]

Not equal.

```
393 {
394     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]) ||
395            (_values[2] != v._values[2]);
396 }
```

6.325.3.14 int HxVec3Double::operator< (const HxVec3Double & *v*) const [inline]

Less than.

```
400 {
401     return (fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2])) <
402            (fabs(v._values[0]) + fabs(v._values[1]) + fabs(v._values[2]));
403 }
```

6.325.3.15 int HxVec3Double::operator<= (const HxVec3Double & *v*) const [inline]

Less equal.

```
407 {
408     return (fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2])) <=
409            (fabs(v._values[0]) + fabs(v._values[1]) + fabs(v._values[2]));
410 }
```

6.325.3.16 int HxVec3Double::operator> (const HxVec3Double & *v*) const [inline]

Greater than.

```
414 {
415     return (fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2])) >
416            (fabs(v._values[0]) + fabs(v._values[1]) + fabs(v._values[2]));
417 }
```

6.325.3.17 int HxVec3Double::operator>= (const HxVec3Double & *v*) const [inline]

Greater equal.

```
421 {
422     return (fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2])) >=
423            (fabs(v._values[0]) + fabs(v._values[1]) + fabs(v._values[2]));
424 }
```

6.325.3.18 HxVec3Double HxVec3Double::operator- () const [inline]

Negation.

```
428 {
429     return HxVec3Double(-_values[0], -_values[1], -_values[2]);
430 }
```

6.325.3.19 HxVec3Double HxVec3Double::complement () const [inline]

Complement.

```
434 {
435     return HxVec3Double(-_values[0], -_values[1], -_values[2]);
436 }
```

6.325.3.20 HxVec3Double HxVec3Double::abs () const [inline]

Absolute value.

```
440 {
441     return HxVec3Double(fabs(_values[0]), fabs(_values[1]), fabs(_values[2]));
442 }
```

6.325.3.21 HxVec3Double HxVec3Double::ceil () const [inline]

Ceiling.

```
446 {
447     return HxVec3Double(::ceil(_values[0]),
448                         ::ceil(_values[1]),
449                         ::ceil(_values[2]));
450 }
```

6.325.3.22 HxVec3Double HxVec3Double::floor () const [inline]

Floor.

```
454 {
455     return HxVec3Double(::floor(_values[0]),
456                         ::floor(_values[1]),
457                         ::floor(_values[2]));
458 }
```

6.325.3.23 HxVec3Double HxVec3Double::round () const [inline]

Round.

```
462 {
463     return HxVec3Double((int) (_values[0] + ((_values[0] >= 0) ? 0.5 : -0.5)),
464                         (int) (_values[1] + ((_values[1] >= 0) ? 0.5 : -0.5)),
465                         (int) (_values[2] + ((_values[2] >= 0) ? 0.5 : -0.5)));
466 }
```

6.325.3.24 HxScalarDouble HxVec3Double::sum () const [inline]

Sum.

```
470 {
471     return _values[0] + _values[1] + _values[2];
472 }
```

6.325.3.25 HxScalarDouble HxVec3Double::product () const [inline]

Product.

```
476 {
477     return _values[0] * _values[1] * _values[2];
478 }
```

6.325.3.26 HxScalarDouble HxVec3Double::min () const

Minimum.

```
59 {
60     return (_values[0] < _values[1]) ?
61         ((_values[0] < _values[2]) ? _values[0] : _values[2]) :
62         ((_values[1] < _values[2]) ? _values[1] : _values[2]);
63 }
```

6.325.3.27 HxScalarDouble HxVec3Double::max () const

Maximum.

```
67 {
68     return (_values[0] > _values[1]) ?
69         ((_values[0] > _values[2]) ? _values[0] : _values[2]) :
70         ((_values[1] > _values[2]) ? _values[1] : _values[2]);
71 }
```

6.325.3.28 HxScalarDouble HxVec3Double::norm1 () const

L1 norm.

```
75 {
76     return fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2]);
77 }
```

6.325.3.29 HxScalarDouble HxVec3Double::norm2 () const

L2 norm.

```
81 {
82     return ::sqrt(_values[0]*_values[0] +
83                   _values[1]*_values[1] +
84                   _values[2]*_values[2]);
85 }
```

6.325.3.30 HxScalarDouble HxVec3Double::normInf () const

L infinity norm.

```
89 {
90     return (fabs(_values[0]) > fabs(_values[1])) ?
91         ((fabs(_values[0]) > fabs(_values[2])) ? fabs(_values[0]) :
92                                                  fabs(_values[2])) :
93         ((fabs(_values[1]) > fabs(_values[2])) ? fabs(_values[1]) :
94                                                  fabs(_values[2]));
95 }
```

6.325.3.31 HxVec3Double HxVec3Double::sqrt () const [inline]

Square root.

```
482 {
483     return HxVec3Double(::sqrt(_values[0]),
484                         ::sqrt(_values[1]),
485                         ::sqrt(_values[2]));
486 }
```

6.325.3.32 HxVec3Double HxVec3Double::sin () const [inline]

Sine.

```
490 {
491     return HxVec3Double(::sin(_values[0]),
492                         ::sin(_values[1]),
493                         ::sin(_values[2]));
494 }
```

6.325.3.33 HxVec3Double HxVec3Double::cos () const [inline]

Cosine.

```
498 {
499     return HxVec3Double(::cos(_values[0]),
500                         ::cos(_values[1]),
501                         ::cos(_values[2]));
502 }
```

6.325.3.34 HxVec3Double HxVec3Double::tan () const [inline]

Tangent.

```
506 {
507     return HxVec3Double(::tan(_values[0]),
508                         ::tan(_values[1]),
509                         ::tan(_values[2]));
510 }
```

6.325.3.35 HxVec3Double HxVec3Double::asin () const [inline]

Arc sine.

```
514 {
515     return HxVec3Double(::asin(_values[0]),
516                         ::asin(_values[1]),
517                         ::asin(_values[2]));
518 }
```

6.325.3.36 HxVec3Double HxVec3Double::acos () const [inline]

Arc cosine.

```
522 {
523     return HxVec3Double(::acos(_values[0]),
524                         ::acos(_values[1]),
525                         ::acos(_values[2]));
526 }
```

6.325.3.37 HxVec3Double HxVec3Double::atan () const [inline]

Arc tangent.

```
530 {
531     return HxVec3Double(::atan(_values[0]),
532                         ::atan(_values[1]),
533                         ::atan(_values[2]));
534 }
```

6.325.3.38 HxScalarDouble HxVec3Double::atan2 () const

Arc tangent.

```
99 {
100     return HxScalarDouble(::atan2(_values[0], _values[1]));
101 }
```

6.325.3.39 HxVec3Double HxVec3Double::sinh () const [inline]

Hyperbolic sine.

```
538 {
539     return HxVec3Double(::sinh(_values[0]),
540                         ::sinh(_values[1]),
541                         ::sinh(_values[2]));
542 }
```

6.325.3.40 HxVec3Double HxVec3Double::cosh () const [inline]

Hyperbolic cosine.

```
546 {
547     return HxVec3Double(::cosh(_values[0]),
548                         ::cosh(_values[1]),
549                         ::cosh(_values[2]));
550 }
```

6.325.3.41 HxVec3Double HxVec3Double::tanh () const [inline]

Hyperbolic tangent.

```
554 {
555     return HxVec3Double(::tanh(_values[0]),
556                         ::tanh(_values[1]),
557                         ::tanh(_values[2]));
558 }
```

6.325.3.42 HxVec3Double HxVec3Double::exp () const [inline]

Exponent.

```
562 {
563     return HxVec3Double(::exp(_values[0]),
564                         ::exp(_values[1]),
565                         ::exp(_values[2]));
566 }
```

6.325.3.43 HxVec3Double HxVec3Double::log () const [inline]

Natural logarithm.

```
570 {
571     return HxVec3Double(::log(_values[0]),
572                         ::log(_values[1]),
573                         ::log(_values[2]));
574 }
```

6.325.3.44 HxVec3Double HxVec3Double::log10 () const [inline]

Base 10 logarithm.

```
578 {
579     return HxVec3Double(::log10(_values[0]),
580                         ::log10(_values[1]),
581                         ::log10(_values[2]));
582 }
```

6.325.3.45 HxVec3Double & HxVec3Double::operator+= (const HxVec3Double & *v*) [inline]

Addition and assignment.

```
586 {
587     _values[0] += v._values[0];
588     _values[1] += v._values[1];
589     _values[2] += v._values[2];
590     return *this;
591 }
```

6.325.3.46 HxVec3Double & HxVec3Double::operator-= (const HxVec3Double & *v*) [inline]

Subtraction and assignment.

```
595 {
596     _values[0] -= v._values[0];
597     _values[1] -= v._values[1];
598     _values[2] -= v._values[2];
599     return *this;
600 }
```

6.325.3.47 HxVec3Double & HxVec3Double::operator *= (const HxVec3Double & *v*) [inline]

Multiplication and assignment.

```
604 {
605     _values[0] *= v._values[0];
606     _values[1] *= v._values[1];
607     _values[2] *= v._values[2];
608     return *this;
609 }
```

6.325.3.48 HxVec3Double & HxVec3Double::operator/= (const HxVec3Double & *v*) [inline]

Division and assignment.

```
613 {
614     _values[0] /= v._values[0];
615     _values[1] /= v._values[1];
616     _values[2] /= v._values[2];
617     return *this;
618 }
```

6.325.3.49 HxVec3Double HxVec3Double::min (const HxVec3Double & *v*) const [inline]

Minimum.

```
654 {
655     return (operator<(v)) ? (*this) : v;
656 }
```

6.325.3.50 HxVec3Double & HxVec3Double::minAssign (const HxVec3Double & *v*) [inline]

Minimum and assignment.

```
660 {
661     if (operator<(v))
662         return *this;
663     operator=(v);
664     return *this;
665 }
```

6.325.3.51 HxVec3Double HxVec3Double::max (const HxVec3Double & *v*) const [inline]

Maximum.

```
669 {
670     return (operator>(v)) ? (*this) : v;
671 }
```

6.325.3.52 HxVec3Double & HxVec3Double::maxAssign (const HxVec3Double & *v*) [inline]

Maximum and assignment.

```
675 {
676     if (operator>(v))
677         return *this;
678     operator=(v);
679     return *this;
680 }
```

6.325.3.53 HxVec3Double HxVec3Double::inf (const HxVec3Double & *v*) const [inline]

Infimum.

```
684 {
685     return HxVec3Double((_values[0] < v._values[0]) ? _values[0] : v._values[0],
686                         (_values[1] < v._values[1]) ? _values[1] : v._values[1],
687                        (_values[2] < v._values[2]) ? _values[2] : v._values[2]);
688 }
```

6.325.3.54 HxVec3Double & HxVec3Double::infAssign (const HxVec3Double & *v*) [inline]

Infimum and assignment.

```
692 {
693     _values[0] = (_values[0] < v._values[0]) ? _values[0] : v._values[0];
694     _values[1] = (_values[1] < v._values[1]) ? _values[1] : v._values[1];
695     _values[2] = (_values[2] < v._values[2]) ? _values[2] : v._values[2];
696     return *this;
697 }
```

6.325.3.55 HxVec3Double HxVec3Double::sup (const HxVec3Double & *v*) const [inline]

Supremum.

```
701 {
702     return HxVec3Double((_values[0] > v._values[0]) ? _values[0] : v._values[0],
703                         (_values[1] > v._values[1]) ? _values[1] : v._values[1],
704                        (_values[2] > v._values[2]) ? _values[2] : v._values[2]);
705 }
```

6.325.3.56 HxVec3Double & HxVec3Double::supAssign (const HxVec3Double & *v*) [inline]

Supremum and assignment.

```
709 {
710     _values[0] = (_values[0] > v._values[0]) ? _values[0] : v._values[0];
711     _values[1] = (_values[1] > v._values[1]) ? _values[1] : v._values[1];
712     _values[2] = (_values[2] > v._values[2]) ? _values[2] : v._values[2];
713     return *this;
714 }
```

6.325.3.57 HxVec3Double HxVec3Double::pow (const HxVec3Double & *v*) const [inline]

Power.

```
718 {
719     return HxVec3Double(::pow(_values[0], v._values[0]),
720                         ::pow(_values[1], v._values[1]),
721                         ::pow(_values[2], v._values[2]));
722 }
```

6.325.3.58 HxVec3Double HxVec3Double::mod (const HxVec3Double & *v*) const [inline]

Modulo.

```
726 {
727     return (*this);
728 }
```

6.325.3.59 HxVec3Double HxVec3Double::and (const HxVec3Double & *v*) const [inline]

And.

```
732 {
733     return (*this);
734 }
```

6.325.3.60 HxVec3Double HxVec3Double::or (const HxVec3Double & *v*) const [inline]

Or.

```
738 {
739     return (*this);
740 }
```

6.325.3.61 HxVec3Double HxVec3Double::xor (const HxVec3Double & *v*) const [inline]

Xor.

```
744 {
745     return (*this);
746 }
```

6.325.3.62 HxVec3Double HxVec3Double::leftShift (const HxVec3Double & *v*) const [inline]

Left shift.

```
750 {
751     return (*this);
752 }
```

6.325.3.63 HxVec3Double HxVec3Double::rightShift (const HxVec3Double & *v*) const [inline]

Right shift.

```
756 {
757     return (*this);
758 }
```

6.325.3.64 HxScalarDouble HxVec3Double::dot (const HxVec3Double & *v*) const

Dot product.

```
105 {
106     return (_values[0] * v._values[0]) +
107            (_values[1] * v._values[1]) +
108            (_values[2] * v._values[2]);
109 }
```

6.325.3.65 HxVec3Double HxVec3Double::cross (const HxVec3Double & *v*) const [inline]

Cross product.

```
762 {
763     return HxVec3Double(_values[1] * v._values[2] - _values[2] * v._values[1],
764                         _values[2] * v._values[0] - _values[0] * v._values[2],
765                         _values[0] * v._values[1] - _values[1] * v._values[0]);
766 }
```

6.325.3.66 STD_OSTREAM & HxVec3Double::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
113 {
114     return os << "(" << _values[0] << "," << _values[1] << "," <<
115         _values[2] << ")";
116 }
```

6.325.3.67 HxString HxVec3Double::toString () const

Value as a string.

```
119                               {
120     return HxString("(") + makeString(_values[0]) + ", "
121             + makeString(_values[1]) + ", "
122             + makeString(_values[2]) + ")";
123 }
```

6.325.4 Friends And Related Function Documentation

6.325.4.1 HxVec3Double operator+ (const HxVec3Double & *v1*, const HxVec3Double & *v2*) [friend]

Addition.

```
622 {
623     return HxVec3Double(v1._values[0] + v2._values[0],
624                         v1._values[1] + v2._values[1],
625                         v1._values[2] + v2._values[2]);
626 }
```

6.325.4.2 HxVec3Double operator- (const HxVec3Double & *v1*, const HxVec3Double & *v2*) [friend]

Subtraction.

```
630 {
631     return HxVec3Double(v1._values[0] - v2._values[0],
632                         v1._values[1] - v2._values[1],
633                         v1._values[2] - v2._values[2]);
634 }
```

6.325.4.3 HxVec3Double operator * (const HxVec3Double & *v1*, const HxVec3Double & *v2*) [friend]

Multiplication.

```
638 {
639     return HxVec3Double(v1._values[0] * v2._values[0],
640                         v1._values[1] * v2._values[1],
641                         v1._values[2] * v2._values[2]);
642 }
```

6.325.4.4 HxVec3Double operator/ (const HxVec3Double & *v1*, const HxVec3Double & *v2*) [friend]

Division.

```
646 {
647     return HxVec3Double(v1._values[0] / v2._values[0],
648                         v1._values[1] / v2._values[1],
649                         v1._values[2] / v2._values[2]);
650 }
```

6.325.5 Member Data Documentation

6.325.5.1 const HxVec3Double HxVec3Double::SMALL_VAL = HxVec3Double(0, 0, 0) [static]

A small value w.r.t to the comparison operators "<" and ">".

Not actually the minimum to avoid overflow.

6.325.5.2 const HxVec3Double HxVec3Double::LARGE_VAL = HxVec3Double(1e300, 1e300, 1e300) [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

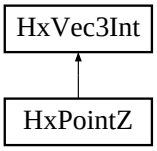
- **HxVec3Double.h**
- HxVec3Double.c

6.326 HxVec3Int Class Reference

Class definition vector of 3 integers.

`#include <HxVec3Int.h>`

Inheritance diagram for HxVec3Int::



Constructors

- **HxVec3Int** ()
  Default constructor.
- **HxVec3Int** (int x, int y, int z)
  Conversion from native type.
- **HxVec3Int** (const HxVec3Int &v)
  Copy constructor.

Inquiry

- int **dim** () const
  Dimensionality.
- int **x** () const
  Value of first element.
- int **y** () const
  Value of second element.
- int **z** () const
  Value of third element.
- int **getValue** (int dimension) const
  Element in given dimension.
- void **setValue** (int dimension, int value)

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 696).

- **operator HxScalarDouble** () const

  Cast to **HxScalarDouble** (p. 677).

- **operator HxVec2Int** () const

  Cast to **HxVec2Int** (p. 785).

- **operator HxVec2Double** () const

  Cast to **HxVec2Double** (p. 766).

- **operator HxVec3Double** () const

  Cast to **HxVec3Double** (p. 804).

- **operator HxComplex** () const

  Cast to **HxComplex** (p. 239).

Operators

Mathematical definition: **Binary operations** (p. 6)

- int **operator==** (const HxVec3Int &v) const

  Equal.

- int **operator!=** (const HxVec3Int &v) const

  Not equal.

- int **operator<** (const HxVec3Int &v) const

  Less than.

- int **operator<=** (const HxVec3Int &v) const

  Less equal.

- int **operator>** (const HxVec3Int &v) const

  Greater than.

- int **operator>=** (const HxVec3Int &v) const

  Greater equal.

- const HxVec3Int **SMALL_VAL** = HxVec3Int(0, 0, 0)

  A small value w.r.t to the comparison operators "<" and ">".

- const HxVec3Int **LARGE_VAL** = HxVec3Int(200000000, 200000000, 200000000)

  A large value w.r.t to the comparison operators "<" and ">".

Unary operations

Mathematical definition: **Unary operations** (p. 5)

- HxVec3Int **operator-** () const
  Negation.
- HxVec3Int **complement** () const
  Complement.
- HxVec3Int **abs** () const
  Absolute value.
- HxVec3Int **ceil** () const
  Ceiling.
- HxVec3Int **floor** () const
  Floor.
- HxVec3Int **round** () const
  Round.
- **HxScalarInt sum** () const
  Sum.
- **HxScalarInt product** () const
  Product.
- **HxScalarInt min** () const
  Minimum.
- **HxScalarInt max** () const
  Maximum.
- **HxScalarInt norm1** () const
  L1 norm.
- **HxScalarDouble norm2** () const
  L2 norm.
- **HxScalarInt normInf** () const
  L infinity norm.
- **HxVec3Double sqrt** () const
  Square root.
- **HxVec3Double sin** () const
  Sine.
- **HxVec3Double cos** () const

Cosine.

- **HxVec3Double tan** () const
  Tangent.
- **HxVec3Double asin** () const
  Arc sine.
- **HxVec3Double acos** () const
  Arc cosine.
- **HxVec3Double atan** () const
  Arc tangent.
- **HxScalarDouble atan2** () const
  Arc tangent.
- **HxVec3Double sinh** () const
  Hyperbolic sine.
- **HxVec3Double cosh** () const
  Hyperbolic cosine.
- **HxVec3Double tanh** () const
  Hyperbolic tangent.
- **HxVec3Double exp** () const
  Exponent.
- **HxVec3Double log** () const
  Natural logarithm.
- **HxVec3Double log10** () const
  Base 10 logarithm.

Binary operations

Mathematical definition: **Binary operations** (p. 6)

- HxVec3Int & **operator+=** (const HxVec3Int &v)
  Addition and assignment.
- HxVec3Int & **operator-=** (const HxVec3Int &v)
  Subtraction and assignment.
- HxVec3Int & **operator** *= (const HxVec3Int &v)
  Multiplication and assignment.
- HxVec3Int & **operator/=** (const HxVec3Int &v)

Division and assignment.

- HxVec3Int **min** (const HxVec3Int &v) const
  Minimum.
- HxVec3Int & **minAssign** (const HxVec3Int &v)
  Minimum and assignment.
- HxVec3Int **max** (const HxVec3Int &v) const
  Maximum.
- HxVec3Int & **maxAssign** (const HxVec3Int &v)
  Maximum and assignment.
- HxVec3Int **inf** (const HxVec3Int &v) const
  Infimum.
- HxVec3Int & **infAssign** (const HxVec3Int &v)
  Infimum and assignment.
- HxVec3Int **sup** (const HxVec3Int &v) const
  Supremum.
- HxVec3Int & **supAssign** (const HxVec3Int &v)
  Supremum and assignment.
- HxVec3Int **pow** (const HxVec3Int &v) const
  Power.
- HxVec3Int **mod** (const HxVec3Int &v) const
  Modulo.
- HxVec3Int **and** (const HxVec3Int &v) const
  And.
- HxVec3Int **or** (const HxVec3Int &v) const
  Or.
- HxVec3Int **xor** (const HxVec3Int &v) const
  Xor.
- HxVec3Int **leftShift** (const HxVec3Int &v) const
  Left shift.
- HxVec3Int **rightShift** (const HxVec3Int &v) const
  Right shift.
- **HxScalarInt dot** (const HxVec3Int &v) const
  Dot product.

- HxVec3Int **cross** (const HxVec3Int &v) const

  Cross product.

- HxVec3Int **operator+** (const HxVec3Int &v1, const HxVec3Int &v2)

  Addition.

- HxVec3Int **operator-** (const HxVec3Int &v1, const HxVec3Int &v2)

  Subtraction.

- HxVec3Int **operator** * (const HxVec3Int &v1, const HxVec3Int &v2)

  Multiplication.

- HxVec3Int **operator/** (const HxVec3Int &v1, const HxVec3Int &v2)

  Division.

Output

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Print value on stream.

- **HxString toString** () const

  Value as a string.

Public Methods

- void * **operator new** (size_t, void *=0)

6.326.1 Detailed Description

Class definition vector of 3 integers.

6.326.2 Constructor & Destructor Documentation

6.326.2.1 HxVec3Int::HxVec3Int () [inline]

Default constructor.

```
322 {
323 }
```

6.326.2.2 HxVec3Int::HxVec3Int (int *x*, int *y*, int *z*) [inline]

Conversion from native type.

```
327 {
328     _values[0] = x;
329     _values[1] = y;
330     _values[2] = z;
331 }
```

6.326.2.3 HxVec3Int::HxVec3Int (const HxVec3Int & *v*) [inline]

Copy constructor.

```
335 {
336     _values[0] = v._values[0];
337     _values[1] = v._values[1];
338     _values[2] = v._values[2];
339 }
```

6.326.3 Member Function Documentation

6.326.3.1 int HxVec3Int::dim () const [inline]

Dimensionality.

```
349 {
350     return 3;
351 }
```

6.326.3.2 int HxVec3Int::x () const [inline]

Value of first element.

```
355 {
356     return _values[0];
357 }
```

6.326.3.3 int HxVec3Int::y () const [inline]

Value of second element.

```
361 {
362     return _values[1];
363 }
```

6.326.3.4 int HxVec3Int::z () const [inline]

Value of third element.

```
367 {
368     return _values[2];
369 }
```

6.326.3.5 int HxVec3Int::getValue (int *dimension*) const [inline]

Element in given dimension.

```
373 {
374     return _values[dimension - 1];
375 }
```

6.326.3.6 HxVec3Int::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 696).

```
28 {
29     return _values[0];
30 }
```

6.326.3.7 HxVec3Int::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 677).

```
33 {
34     return (double) _values[0];
35 }
```

6.326.3.8 HxVec3Int::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 785).

```
39 {
40     return HxVec2Int(_values[0], _values[1]);
41 }
```

6.326.3.9 HxVec3Int::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 766).

```
44 {
45     return HxVec2Double(_values[0], _values[1]);
46 }
```

6.326.3.10 HxVec3Int::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 804).

```
49 {
50     return HxVec3Double(_values[0], _values[1], _values[2]);
51 }
```

6.326.3.11 HxVec3Int::operator HxComplex () const

Cast to **HxComplex** (p. 239).

```
54 {
55     return HxComplex(_values[0], _values[1]);
56 }
```

6.326.3.12 int HxVec3Int::operator== (const HxVec3Int & *v*) const [inline]

Equal.

```
385 {
386     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]) &&
387            (_values[2] == v._values[2]);
388 }
```

6.326.3.13 int HxVec3Int::operator!= (const HxVec3Int & *v*) const [inline]

Not equal.

```
392 {
393     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]) ||
394            (_values[2] != v._values[2]);
395 }
```

6.326.3.14 int HxVec3Int::operator< (const HxVec3Int & *v*) const [inline]

Less than.

```
399 {
400     return (::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2])) <
401            (::abs(v._values[0]) + ::abs(v._values[1]) + ::abs(v._values[2]));
402 }
```

6.326.3.15 int HxVec3Int::operator<= (const HxVec3Int & *v*) const [inline]

Less equal.

```
406 {
407     return (::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2])) <=
408            (::abs(v._values[0]) + ::abs(v._values[1]) + ::abs(v._values[2]));
409 }
```

6.326.3.16 int HxVec3Int::operator> (const HxVec3Int & *v*) const [inline]

Greater than.

```
413 {
414     return (::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2])) >
415            (::abs(v._values[0]) + ::abs(v._values[1]) + ::abs(v._values[2]));
416 }
```

6.326.3.17 int HxVec3Int::operator>= (const HxVec3Int & *v*) const [inline]

Greater equal.

```
420 {
421     return (::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2])) >=
422            (::abs(v._values[0]) + ::abs(v._values[1]) + ::abs(v._values[2]));
423 }
```

6.326.3.18 HxVec3Int HxVec3Int::operator- () const [inline]

Negation.

```
427 {
428     return HxVec3Int(-_values[0], -_values[1], -_values[2]);
429 }
```

6.326.3.19 HxVec3Int HxVec3Int::complement () const [inline]

Complement.

```
433 {
434     return HxVec3Int(~_values[0], ~_values[1], ~_values[2]);
435 }
```

6.326.3.20 HxVec3Int HxVec3Int::abs () const [inline]

Absolute value.

```
439 {
440     return HxVec3Int(::abs(_values[0]), ::abs(_values[1]), ::abs(_values[2]));
441 }
```

6.326.3.21 HxVec3Int HxVec3Int::ceil () const [inline]

Ceiling.

```
445 {
446     return *this;
447 }
```

6.326.3.22 HxVec3Int HxVec3Int::floor () const [inline]

Floor.

```
451 {
452     return *this;
453 }
```

6.326.3.23 HxVec3Int HxVec3Int::round () const [inline]

Round.

```
457 {
458     return *this;
459 }
```

6.326.3.24 HxScalarInt HxVec3Int::sum () const [inline]

Sum.

```
463 {
464     return _values[0] + _values[1] + _values[2];
465 }
```

6.326.3.25 HxScalarInt HxVec3Int::product () const [inline]

Product.

```
469 {
470     return _values[0] * _values[1] * _values[2];
471 }
```

6.326.3.26 HxScalarInt HxVec3Int::min () const

Minimum.

```
60 {
61     return (_values[0] < _values[1]) ?
62         ((_values[0] < _values[2]) ? _values[0] : _values[2]) :
63         ((_values[1] < _values[2]) ? _values[1] : _values[2]);
64 }
```

6.326.3.27 HxScalarInt HxVec3Int::max () const

Maximum.

```
68 {
69     return (_values[0] > _values[1]) ?
70         ((_values[0] > _values[2]) ? _values[0] : _values[2]) :
71         ((_values[1] > _values[2]) ? _values[1] : _values[2]);
72 }
```

6.326.3.28 HxScalarInt HxVec3Int::norm1 () const

L1 norm.

```
76 {
77     return ::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2]);
78 }
```

6.326.3.29 HxScalarDouble HxVec3Int::norm2 () const

L2 norm.

```
82 {
83     return ::sqrt(double(_values[0])*_values[0] +
84                   double(_values[1])*_values[1] +
85                   double(_values[2])*_values[2]);
86 }
```

6.326.3.30 HxScalarInt HxVec3Int::normInf () const

L infinity norm.

```
90 {
91     return (::abs(_values[0]) > ::abs(_values[1])) ?
92         ((::abs(_values[0]) > ::abs(_values[2])) ? ::abs(_values[0]) :
93                                                    ::abs(_values[2])) :
94         ((::abs(_values[1]) > ::abs(_values[2])) ? ::abs(_values[1]) :
95                                                    ::abs(_values[2]));
96 }
```

6.326.3.31 HxVec3Double HxVec3Int::sqrt () const

Square root.

```
100 {
101     return HxVec3Double(::sqrt(double(_values[0])),
102                         ::sqrt(double(_values[1])),
103                         ::sqrt(double(_values[2])));
104 }
```

6.326.3.32 HxVec3Double HxVec3Int::sin () const

Sine.

```
108 {
109     return HxVec3Double(::sin(double(_values[0])),
110                         ::sin(double(_values[1])),
111                         ::sin(double(_values[2])));
112 }
```

6.326.3.33 HxVec3Double HxVec3Int::cos () const

Cosine.

```
116 {
117     return HxVec3Double(::cos(double(_values[0])),
118                         ::cos(double(_values[1])),
119                         ::cos(double(_values[2])));
120 }
```

6.326.3.34 HxVec3Double HxVec3Int::tan () const

Tangent.

```
124 {
125     return HxVec3Double(::tan(double(_values[0])),
126                         ::tan(double(_values[1])),
127                         ::tan(double(_values[2])));
128 }
```

6.326.3.35 HxVec3Double HxVec3Int::asin () const

Arc sine.

```
132 {
133     return HxVec3Double(::asin(double(_values[0])),
134                         ::asin(double(_values[1])),
135                         ::asin(double(_values[2])));
136 }
```

6.326.3.36 HxVec3Double HxVec3Int::acos () const

Arc cosine.

```
140 {
141     return HxVec3Double(::acos(double(_values[0])),
142                         ::acos(double(_values[1])),
143                         ::acos(double(_values[2])));
144 }
```

6.326.3.37 HxVec3Double HxVec3Int::atan () const

Arc tangent.

```
154 {
155     return HxVec3Double(::atan(double(_values[0])),
156                         ::atan(double(_values[1])),
157                         ::atan(double(_values[2])));
158 }
```

6.326.3.38 HxScalarDouble HxVec3Int::atan2 () const

Arc tangent.

```
148 {
149     return HxScalarDouble(::atan2(double(_values[0]), double(_values[1])));
150 }
```

6.326.3.39 HxVec3Double HxVec3Int::sinh () const

Hyperbolic sine.

```
162 {
163     return HxVec3Double(::sinh(double(_values[0])),
164                         ::sinh(double(_values[1])),
165                         ::sinh(double(_values[2])));
166 }
```

6.326.3.40 HxVec3Double HxVec3Int::cosh () const

Hyperbolic cosine.

```
170 {
171     return HxVec3Double(::cosh(double(_values[0])),
172                         ::cosh(double(_values[1])),
173                         ::cosh(double(_values[2])));
174 }
```

6.326.3.41 HxVec3Double HxVec3Int::tanh () const

Hyperbolic tangent.

```
178 {
179     return HxVec3Double(::tanh(double(_values[0])),
180                         ::tanh(double(_values[1])),
181                         ::tanh(double(_values[2])));
182 }
```

6.326.3.42 HxVec3Double HxVec3Int::exp () const

Exponent.

```
186 {
187     return HxVec3Double(::exp(double(_values[0])),
188                         ::exp(double(_values[1])),
189                         ::exp(double(_values[2])));
190 }
```

6.326.3.43 HxVec3Double HxVec3Int::log () const

Natural logarithm.

```
194 {
195     return HxVec3Double(::log(double(_values[0])),
196                         ::log(double(_values[1])),
197                         ::log(double(_values[2])));
198 }
```

6.326.3.44 HxVec3Double HxVec3Int::log10 () const

Base 10 logarithm.

```
202 {
203     return HxVec3Double(::log10(double(_values[0])),
204                         ::log10(double(_values[1])),
205                         ::log10(double(_values[2])));
206 }
```

6.326.3.45 HxVec3Int & HxVec3Int::operator+= (const HxVec3Int & *v*) [inline]

Addition and assignment.

```
475 {
476     _values[0] += v._values[0];
477     _values[1] += v._values[1];
478     _values[2] += v._values[2];
479     return *this;
480 }
```

6.326.3.46 HxVec3Int & HxVec3Int::operator-= (const HxVec3Int & *v*) [inline]

Subtraction and assignment.

```
484 {
485     _values[0] -= v._values[0];
486     _values[1] -= v._values[1];
487     _values[2] -= v._values[2];
488     return *this;
489 }
```

6.326.3.47 HxVec3Int & HxVec3Int::operator *= (const HxVec3Int & *v*) [inline]

Multiplication and assignment.

```
493 {
494     _values[0] *= v._values[0];
495     _values[1] *= v._values[1];
496     _values[2] *= v._values[2];
497     return *this;
498 }
```

6.326.3.48 HxVec3Int & HxVec3Int::operator/= (const HxVec3Int & *v*) [inline]

Division and assignment.

```
502 {
503     _values[0] /= v._values[0];
504     _values[1] /= v._values[1];
505     _values[2] /= v._values[2];
506     return *this;
507 }
```

6.326.3.49 HxVec3Int HxVec3Int::min (const HxVec3Int & *v*) const [inline]

Minimum.

```
543 {
544     return (operator<(v)) ? (*this) : v;
545 }
```

6.326.3.50 HxVec3Int & HxVec3Int::minAssign (const HxVec3Int & *v*) [inline]

Minimum and assignment.

```
549 {
550     if (operator<(v))
551         return *this;
552     operator=(v);
553     return *this;
554 }
```

6.326.3.51 HxVec3Int HxVec3Int::max (const HxVec3Int & *v*) const [inline]

Maximum.

```
558 {
559     return (operator>(v)) ? (*this) : v;
560 }
```

6.326.3.52 HxVec3Int & HxVec3Int::maxAssign (const HxVec3Int & *v*) [inline]

Maximum and assignment.

```
564 {
565     if (operator>(v))
566         return *this;
567     operator=(v);
568     return *this;
569 }
```

6.326.3.53 HxVec3Int HxVec3Int::inf (const HxVec3Int & *v*) const [inline]

Infimum.

```
573 {
574     return HxVec3Int((_values[0] < v._values[0]) ? _values[0] : v._values[0],
575                      (_values[1] < v._values[1]) ? _values[1] : v._values[1],
576                      (_values[2] < v._values[2]) ? _values[2] : v._values[2]);
577 }
```

6.326.3.54 HxVec3Int & HxVec3Int::infAssign (const HxVec3Int & *v*) [inline]

Infimum and assignment.

```
581 {
582     _values[0] = (_values[0] < v._values[0]) ? _values[0] : v._values[0];
583     _values[1] = (_values[1] < v._values[1]) ? _values[1] : v._values[1];
584     _values[2] = (_values[2] < v._values[2]) ? _values[2] : v._values[2];
585     return *this;
586 }
```

6.326.3.55 HxVec3Int HxVec3Int::sup (const HxVec3Int & *v*) const [inline]

Supremum.

```
590 {
591     return HxVec3Int((_values[0] > v._values[0]) ? _values[0] : v._values[0],
592                      (_values[1] > v._values[1]) ? _values[1] : v._values[1],
593                      (_values[2] > v._values[2]) ? _values[2] : v._values[2]);
594 }
```

6.326.3.56 HxVec3Int & HxVec3Int::supAssign (const HxVec3Int & *v*) [inline]

Supremum and assignment.

```
598 {
599     _values[0] = (_values[0] > v._values[0]) ? _values[0] : v._values[0];
600     _values[1] = (_values[1] > v._values[1]) ? _values[1] : v._values[1];
601     _values[2] = (_values[2] > v._values[2]) ? _values[2] : v._values[2];
602     return *this;
603 }
```

6.326.3.57 HxVec3Int HxVec3Int::pow (const HxVec3Int & *v*) const [inline]

Power.

```
607 {
608     return HxVec3Int((int) (::pow(_values[0], v._values[0]) + 0.5),
609                      (int) (::pow(_values[1], v._values[1]) + 0.5),
610                      (int) (::pow(_values[2], v._values[2]) + 0.5));
611 }
```

6.326.3.58 HxVec3Int HxVec3Int::mod (const HxVec3Int & *v*) const [inline]

Modulo.

```
615 {
616     return HxVec3Int(_values[0] % v._values[0],
617                      _values[1] % v._values[1],
618                      _values[2] % v._values[2]);
619 }
```

6.326.3.59 HxVec3Int HxVec3Int::and (const HxVec3Int & *v*) const [inline]

And.

```
623 {
624     return HxVec3Int(_values[0] & v._values[0],
625                      _values[1] & v._values[1],
626                      _values[2] & v._values[2]);
627 }
```

6.326.3.60 HxVec3Int HxVec3Int::or (const HxVec3Int & *v*) const [inline]

Or.

```
631 {
632     return HxVec3Int(_values[0] | v._values[0],
633                      _values[1] | v._values[1],
634                      _values[2] | v._values[2]);
635 }
```

6.326.3.61 HxVec3Int HxVec3Int::xor (const HxVec3Int & *v*) const [inline]

Xor.

```
639 {
640     return HxVec3Int(_values[0] ^ v._values[0],
641                      _values[1] ^ v._values[1],
642                      _values[2] ^ v._values[2]);
643 }
```

6.326.3.62 HxVec3Int HxVec3Int::leftShift (const HxVec3Int & *v*) const [inline]

Left shift.

```
647 {
648     return HxVec3Int(_values[0] << v._values[0],
649                      _values[1] << v._values[1],
650                      _values[2] << v._values[2]);
651 }
```

6.326.3.63 HxVec3Int HxVec3Int::rightShift (const HxVec3Int & *v*) const [inline]

Right shift.

```
655 {
656     return HxVec3Int(_values[0] >> v._values[0],
657                      _values[1] >> v._values[1],
658                      _values[2] >> v._values[2]);
659 }
```

6.326.3.64 HxScalarInt HxVec3Int::dot (const HxVec3Int & *v*) const

Dot product.

```
210 {
211     return (_values[0] * v._values[0]) +
212            (_values[1] * v._values[1]) +
213            (_values[2] * v._values[2]);
214 }
```

6.326.3.65 HxVec3Int HxVec3Int::cross (const HxVec3Int & *v*) const [inline]

Cross product.

```
663 {
664     return HxVec3Int(_values[1] * v._values[2] - _values[2] * v._values[1],
665                      _values[2] * v._values[0] - _values[0] * v._values[2],
666                      _values[0] * v._values[1] - _values[1] * v._values[0]);
667 }
```

6.326.3.66 STD_OSTREAM & HxVec3Int::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
218 {
219     return os << "(" << _values[0] << "," << _values[1] << "," <<
220         _values[2] << ")";
221 }
```

6.326.3.67 HxString HxVec3Int::toString () const

Value as a string.

```
224                             {
225     return HxString("(") + makeString(_values[0]) + ", "
226             + makeString(_values[1]) + ", "
227             + makeString(_values[2]) + ")";
228 }
```

6.326.4 Friends And Related Function Documentation

6.326.4.1 HxVec3Int operator+ (const HxVec3Int & *v1*, const HxVec3Int & *v2*) [friend]

Addition.

```
511 {
512     return HxVec3Int(v1._values[0] + v2._values[0],
513                      v1._values[1] + v2._values[1],
514                      v1._values[2] + v2._values[2]);
515 }
```

6.326.4.2 HxVec3Int operator- (const HxVec3Int & *v1*, const HxVec3Int & *v2*) [friend]

Subtraction.

```
519 {
520     return HxVec3Int(v1._values[0] - v2._values[0],
521                      v1._values[1] - v2._values[1],
522                      v1._values[2] - v2._values[2]);
523 }
```

6.326.4.3 HxVec3Int operator * (const HxVec3Int & *v1*, const HxVec3Int & *v2*) [friend]

Multiplication.

```
527 {
528     return HxVec3Int(v1._values[0] * v2._values[0],
529                      v1._values[1] * v2._values[1],
530                      v1._values[2] * v2._values[2]);
531 }
```

6.326.4.4 HxVec3Int operator/ (const HxVec3Int & *v1*, const HxVec3Int & *v2*) [friend]

Division.

```
535 {
536     return HxVec3Int(v1._values[0] / v2._values[0],
537                      v1._values[1] / v2._values[1],
538                      v1._values[2] / v2._values[2]);
539 }
```

6.326.5 Member Data Documentation

6.326.5.1 const HxVec3Int HxVec3Int::SMALL_VAL = HxVec3Int(0, 0, 0) [static]

A small value w.r.t to the comparison operators "<" and ">".

Not actually the minimum to avoid overflow.

6.326.5.2 const HxVec3Int HxVec3Int::LARGE_VAL = HxVec3Int(200000000, 200000000, 200000000) [static]

A large value w.r.t to the comparison operators "<" and ">".

Not actually the maximum to avoid overflow.

The documentation for this class was generated from the following files:

- **HxVec3Int.h**
- HxVec3Int.c

6.327 HxVector Class Reference

Class definition for vectors.

`#include <HxVector.h>`

Constructors

- **HxVector** ()
  Empty vector.
- **HxVector** (int n)
  Empty vector of given size.
- **HxVector** (int n, double *data)
  vector with given data.
- **HxVector** (double a0, double a1)
  Vector of size 2, with given values.
- **HxVector** (double a0, double a1, double a2)
  Vector of size 3, with given values.
- **HxVector** (double a0, double a1, double a2, double a3)
  Vector of size 4, with given values.
- **HxVector** (const HxVector &v)
  Copy constructor.
- **HxVector** (const **HxMatrix** &m)
  Copy from matrix constructor.

Inquiry

- int **nElem** () const
  Number of elements.
- int **valid** () const
  Indicates whether the vector is valid.

Operators

- HxVector & **operator=** (double a)
  Assign constant value.
- HxVector & **operator=** (const HxVector &v)
  Normal assignment.

- double & **operator[ ]** (int i) const

  Subscripting, start with 0.

- HxVector **operator-** () const

  Unary minus.

- double **operator** ∗ (const HxVector &a, const HxVector &b)

  Multiplication.

- HxVector **operator** ∗ (const double a, const HxVector &b)

  Multiplication.

- HxVector **operator** ∗ (const HxVector &a, const double b)

  Multiplication.

- HxVector **operator/** (const HxVector &a, double b)

  Division.

- HxVector **operator/** (double a, const HxVector &b)

  Division.

- HxVector **operator+** (const HxVector &a, const HxVector &b)

  Addition.

- HxVector **operator+** (const HxVector &a, double b)

  Addition.

- HxVector **operator+** (double a, const HxVector &b)

  Addition.

- HxVector **operator-** (const HxVector &a, const HxVector &b)

  Subtraction.

- HxVector **operator-** (const HxVector &a, double b)

  Subtraction.

- HxVector **operator-** (double a, const HxVector &b)

  Subtraction.

- int **operator==** (const HxVector &a, const HxVector &b)

  Equal.

- int **operator!=** (const HxVector &a, const HxVector &b)

  Not equal.

Operations

- **HxMatrix t** () const
  Transpose Matrix.
- **HxMatrix diag** () const
  Diagonal Matrix.
- HxVector **add** (const HxVector &b) const
  Addition.
- HxVector **add** (const double val) const
  Addition.
- HxVector **sub** (const HxVector &b) const
  Subtraction.
- HxVector **sub** (const double val) const
  Subtraction.
- HxVector **mul** (const HxVector &b) const
  Multiplication.
- HxVector **mul** (const **HxMatrix** &m) const
  Multiplication.
- HxVector **mul** (const double val) const
  Multiplication.
- HxVector **div** (const double val) const
  Division.
- HxVector **sin** () const
  Apply sin to each element.
- HxVector **cos** () const
  Apply cos to each element.
- HxVector **tan** () const
  Apply tan to each element.
- HxVector **sinh** () const
  Apply sinh to each element.
- HxVector **cosh** () const
  Apply cosh to each element.
- HxVector **tanh** () const
  Apply tanh to each element.

- HxVector **exp** () const

  Apply exp to each element.

- HxVector **log** () const

  Apply log to each element.

- HxVector **sqrt** () const

  Apply sqrt to each element.

- HxVector **abs** () const

  Apply abs to each element.

- HxVector **sgn** () const

  Apply sgn to each element.

- HxVector **map** (double(∗f)(double)) const

  Map f to each element of this.

Public Methods

- **∼HxVector** ()
- std::ostream & **put** (std::ostream &os) const

Friends

- class **HxMatrix**

6.327.1 Detailed Description

Class definition for vectors.

The vector may have arbitrary size.

6.327.2 Constructor & Destructor Documentation

6.327.2.1 HxVector::HxVector () [inline]

Empty vector.

```
216 {
217     _n = 0;
218     _data = 0;
219 }
```

6.327.2.2 HxVector::HxVector (int *n*) [inline]

Empty vector of given size.

```
222 {
223     _n = n;
224     _data = new double[_n];
225 }
```

6.327.2.3 HxVector::HxVector (int *n*, double * *data*) [inline]

vector with given data.

```
228 {
229     _n = n;
230     _data = data;
231 }
```

6.327.2.4 HxVector::HxVector (double *a0*, double *a1*) [inline]

Vector of size 2, with given values.

```
234 {
235     _n = 2;
236     _data = new double[_n];
237     _data[0] = a0;
238     _data[1] = a1;
239 }
```

6.327.2.5 HxVector::HxVector (double *a0*, double *a1*, double *a2*) [inline]

Vector of size 3, with given values.

```
242 {
243     _n = 3;
244     _data = new double[_n];
245     _data[0] = a0;
246     _data[1] = a1;
247     _data[2] = a2;
248 }
```

6.327.2.6 HxVector::HxVector (double *a0*, double *a1*, double *a2*, double *a3*) [inline]

Vector of size 4, with given values.

```
251 {
252     _n = 4;
253     _data = new double[_n];
254     _data[0] = a0;
255     _data[1] = a1;
256     _data[2] = a2;
257     _data[3] = a3;
258 }
```

6.327.2.7 HxVector::HxVector (const HxVector & *v*) [inline]

Copy constructor.

```
261 {
262     _n = v.nElem();
263     _data = new double[_n];
264 
265     double* t = v._data;
266     double* u = _data;
267     int i = v.nElem();
268     while (--i >= 0)
269         *u++ = *t++;
270 }
```

6.327.2.8 HxVector::HxVector (const HxMatrix & *m*)

Copy from matrix constructor.

```
27 {
28     _n = m.nElem();
29     _data = new double[_n];
30 
31     double* t = m._data;
32     double* u = _data;
33     int i = m.nElem();
34     while (--i >= 0)
35         *u++ = *t++;
36 }
```

6.327.3 Member Function Documentation

6.327.3.1 int HxVector::nElem () const [inline]

Number of elements.

```
279 {
280     return _n;
281 }
```

6.327.3.2 int HxVector::valid () const [inline]

Indicates whether the vector is valid.

```
285 {
286     return (_n != 0);
287 }
```

6.327.3.3 HxVector & HxVector::operator= (double *a*) [inline]

Assign constant value.

```
291 {
292     int i = nElem();
293     double *t = _data;
294     while (--i >= 0)
295         *t++ = a;
296     return *this;
297 }
```

6.327.3.4 HxVector & HxVector::operator= (const HxVector & *v*) [inline]

Normal assignment.

```
301 {
302     if (this != &v) {
303         delete [] _data;
304         _n = v.nElem();
305         _data = new double [_n];
306         double *t = _data;
307         double *u = v._data;
308         int i = v.nElem();
309         while (--i >= 0)
310             *t++ = *u++;
311     }
312     return *this;
313 }
```

6.327.3.5 double & HxVector::operator[ ] (int *i*) const [inline]

Subscripting, start with 0.

```
317 {
318     return _data[i];
319 }
```

6.327.3.6 HxVector HxVector::operator- () const [inline]

Unary minus.

```
323 {
324     HxVector m(*this);
325     double* t = m._data;
326     double* u = _data;
327     int i = nElem();
328     while (--i >= 0)
329         *t++ = -(*u++);
330     return m;
331 }
```

6.327.3.7 HxMatrix HxVector::t () const

Transpose Matrix.

```
99 {
100     HxMatrix m(nElem(), 1);
101     int i;
102     for (i=0 ; i<nElem() ; i++)
103             m[0][i] = (*this)[i];
104     return m;
105 }
```

6.327.3.8 HxMatrix HxVector::diag () const

Diagonal Matrix.

```
109 {
110     int n = nElem();
111     HxMatrix m(n, n, 0.0);
112 
113     double *t = _data;
114 
115     for (int i = 0; i < nElem(); i++)
116         m[i][i] = *t++;
117 
118     return m;
119 }
```

6.327.3.9 HxVector HxVector::add (const HxVector & *b*) const

Addition.

Equivalent to : a+b

```
124 {
125     return *this+b;
126 }
```

6.327.3.10 HxVector HxVector::add (const double *val*) const

Addition.

Equivalent to : a+val

```
130 {
131     return *this+val;
132 }
```

6.327.3.11 HxVector HxVector::sub (const HxVector & *b*) const

Subtraction.

Equivalent to : a-b

```
136 {
137     return *this-b;
138 }
```

6.327.3.12 HxVector HxVector::sub (const double *val*) const

Subtraction.

Equivalent to : a-val

```
142 {
143     return *this-val;
144 }
```

6.327.3.13 HxVector HxVector::mul (const HxVector & *b*) const

Multiplication.

Equivalent to : a∗b

```
148 {
149     return *this*b;
150 }
```

6.327.3.14 HxVector HxVector::mul (const HxMatrix & *m*) const

Multiplication.

Equivalent to : a∗v

```
160 {
161     return *this*m;
162 }
```

6.327.3.15 HxVector HxVector::mul (const double *val*) const

Multiplication.

Equivalent to : a∗val

```
154 {
155     return *this*val;
156 }
```

6.327.3.16 HxVector HxVector::div (const double *val*) const

Division.

Equivalent to : a/val

```
166 {
167     return *this/val;
168 }
```

6.327.3.17 HxVector HxVector::sin () const

Apply sin to each element.

```
172 {
173     return map(::sin);
174 }
```

6.327.3.18 HxVector HxVector::cos () const

Apply cos to each element.

```
178 {
179     return map(::cos);
180 }
```

6.327.3.19 HxVector HxVector::tan () const

Apply tan to each element.

```
184 {
185     return map(::tan);
186 }
```

6.327.3.20 HxVector HxVector::sinh () const

Apply sinh to each element.

```
190 {
191     return map(::sinh);
192 }
```

6.327.3.21 HxVector HxVector::cosh () const

Apply cosh to each element.

```
196 {
197     return map(::cosh);
198 }
```

6.327.3.22 HxVector HxVector::tanh () const

Apply tanh to each element.

```
202 {
203     return map(::tanh);
204 }
```

6.327.3.23 HxVector HxVector::exp () const

Apply exp to each element.

```
208 {
209     return map(::exp);
210 }
```

6.327.3.24 HxVector HxVector::log () const

Apply log to each element.

```
214 {
215     return map(::log);
216 }
```

6.327.3.25 HxVector HxVector::sqrt () const

Apply sqrt to each element.

```
220 {
221     return map(::sqrt);
222 }
```

6.327.3.26 HxVector HxVector::abs () const

Apply abs to each element.

```
226 {
227     return map(::fabs);
228 }
```

6.327.3.27 HxVector HxVector::sgn () const

Apply sgn to each element.

```
234 {
235     return map(::sgn);
236 }
```

6.327.3.28 HxVector HxVector::map (double(* *f*)(double)) const [inline]

Map f to each element of this.

```
444 {
445     HxVector m(*this);
446     double* t = m._data;
447     double* u = _data;
448     int i = nElem();
449     while (--i >= 0)
450             *t++ = f(*u++);
451     return m;
452 }
```

6.327.4 Friends And Related Function Documentation

6.327.4.1 double operator ∗ (const HxVector & *a*, const HxVector & *b*) [friend]

Multiplication.

```
40 {
41     if (a.nElem() != b.nElem()) {
42         error("nonconformant HxVector * HxVector operands.");
43         return 0;
44     }
45     double sum = 0;
46     int i;
47     for (i=0 ; i<a.nElem() ; i++)
48         sum += a[i] * b[i];
49     return sum;
50 }
```

6.327.4.2 HxVector operator ∗ (const double *a*, const HxVector & *b*) [friend]

Multiplication.

```
347 {
348     HxVector m(b);
349     double* t = m._data;
350     double* u = b._data;
351     int i = b.nElem();
352     while (--i >= 0)
353     *t++ = a * *u++;
354     return m;
355 }
```

6.327.4.3 HxVector operator ∗ (const HxVector & *a*, const double *b*) [friend]

Multiplication.

```
335 {
336     HxVector m(a);
337     double* t = m._data;
338     double* u = a._data;
339     int i = a.nElem();
340     while (--i >= 0)
341         *t++ = *u++ * b;
342     return m;
343 }
```

6.327.4.4 HxVector operator/ (const HxVector & *a*, double *b*) [friend]

Division.

```
359 {
360     HxVector m(a);
361     double* t = m._data;
362     double* u = a._data;
```

```
363     int i = a.nElem();
364     while (--i >= 0)
365         *t++ = *u++ / b;
366     return m;
367 }
```

6.327.4.5 HxVector operator/ (double *a*, const HxVector & *b*) [friend]

Division.

```
371 {
372     HxVector m(b);
373     double* t = m._data;
374     double* u = b._data;
375     int i = b.nElem();
376     while (--i >= 0)
377         *t++ = a / *u++;
378     return m;
379 }
```

6.327.4.6 HxVector operator+ (const HxVector & *a*, const HxVector & *b*) [friend]

Addition.

```
54 {
55     if (a.nElem() != b.nElem()) {
56         error("nonconformant HxVector + HxVector operands.");
57         return HxVector(0);
58     }
59     HxVector m(a.nElem());
60     int i;
61     for (i=0 ; i<a.nElem() ; i++) {
62         m[i] = a[i] + b[i];
63     }
64     return m;
65 }
```

6.327.4.7 HxVector operator+ (const HxVector & *a*, double *b*) [friend]

Addition.

```
383 {
384     HxVector m(a);
385     double* t = m._data;
386     double* u = a._data;
387     int i = a.nElem();
388     while (--i >= 0)
389         *t++ = *u++ + b;
390     return m;
391 }
```

6.327.4.8 HxVector operator+ (double *a*, const HxVector & *b*) [friend]

Addition.

```
395 {
396     HxVector m(b);
397     double* t = m._data;
398     double* u = b._data;
399     int i = b.nElem();
400     while (--i >= 0)
401         *t++ = a + *u++;
402     return m;
403 }
```

6.327.4.9 HxVector operator- (const HxVector & *a*, const HxVector & *b*) [friend]

Subtraction.

```
69 {
70     if (a.nElem() != b.nElem()) {
71         error("nonconformant HxVector - HxVector operands.");
72         return HxVector(0);
73     }
74     HxVector m(a.nElem());
75     int i;
76     for (i=0 ; i<a.nElem() ; i++) {
77         m[i] = a[i] - b[i];
78     }
79     return m;
80 }
```

6.327.4.10 HxVector operator- (const HxVector & *a*, double *b*) [friend]

Subtraction.

```
407 {
408     HxVector m(a);
409     double* t = m._data;
410     double* u = a._data;
411     int i = a.nElem();
412     while (--i >= 0)
413         *t++ = *u++ - b;
414     return m;
415 }
```

6.327.4.11 HxVector operator- (double *a*, const HxVector & *b*) [friend]

Subtraction.

```
419 {
420     HxVector m(b);
421     double* t = m._data;
422     double* u = b._data;
423     int i = b.nElem();
```

```
424     while (--i >= 0)
425         *t++ = a - *u++;
426     return m;
427 }
```

6.327.4.12 int operator== (const HxVector & *a*, const HxVector & *b*) [friend]

Equal.

```
84 {
85     if (a.nElem() != b.nElem())
86         return 0;
87     double *t = a._data;
88     double *u = b._data;
89     int i = a.nElem();
90     while (--i >= 0) {
91         if (fabs(*t++ - *u++) > HxMatrix_EPS)
92             return 0;
93     }
94     return 1;
95 }
```

6.327.4.13 int operator!= (const HxVector & *a*, const HxVector & *b*) [friend]

Not equal.

```
431 {
432     return !(a == b);
433 }
```

The documentation for this class was generated from the following files:

- **HxVector.h**
- HxVector.c

6.328 HxVectorR2 Class Reference

Class definition for vectors in R2 (real-value coordinates).

`#include <HxVectorR2.h>`

Public Methods

- **HxVectorR2** ()

  Constructor.

- **HxVectorR2** (double d1, double d2)

  Constructor.

- **HxVectorR2** (const **HxPointR2** &p1, const **HxPointR2** &p2)

Constructor : p1-p2.

- double **x** () const

  Get the first element of the vector.

- double **y** () const

  Get the second element of the vector.

- HxVectorR2 **add** (const HxVectorR2 &arg) const

  Add the given vector to this.

- HxVectorR2 **sub** (const HxVectorR2 &arg) const

  Subtract the given vector from this.

- HxVectorR2 **mul** (double arg) const

  Multiply this vector with a scalar.

- HxVectorR2 **div** (double arg) const

  Divide this vector by a scalar.

- double **dot** (const HxVectorR2 &arg) const

  Dot product.

- double **cross2D** (const HxVectorR2 &arg) const

  Cross product (?).

- double **magnitude** () const

  Magnitude.

- double **squaredMagnitude** () const

  Squared magnitude.

- HxVectorR2 **normal** () const

  Normal.

- STD_OSTREAM & **put** (STD_OSTREAM &) const

  Put the arrow on the given stream.

- STD_OSTREAM & **dump** (HxVectorR2 &) const
- **HxString toString** () const

Friends

- class **HxPointR2**

6.328.1 Detailed Description

Class definition for vectors in R2 (real-value coordinates).

6.328.2 Constructor & Destructor Documentation

6.328.2.1 HxVectorR2::HxVectorR2 () [inline]

Constructor.

```
87                          : _data(0,0)
88 {
89 }
```

6.328.2.2 HxVectorR2::HxVectorR2 (double *d1*, double *d2*) [inline]

Constructor.

```
92                                              : _data(d1, d2)
93 {
94 }
```

6.328.2.3 HxVectorR2::HxVectorR2 (const HxPointR2 & *p1*, const HxPointR2 & *p2*)

Constructor : p1-p2.

```
16 {
17     _data = HxVec2Double(p1.x()-p2.x(), p1.y()-p2.y());
18 }
```

6.328.3 Member Function Documentation

6.328.3.1 double HxVectorR2::x () const [inline]

Get the first element of the vector.

```
98 {
99     return _data.x();
100 }
```

6.328.3.2 double HxVectorR2::y () const [inline]

Get the second element of the vector.

```
104 {
105     return _data.y();
106 }
```

6.328.3.3 HxVectorR2 HxVectorR2::add (const HxVectorR2 & *arg*) const [inline]

Add the given vector to this.

```
110 {
111     return HxVectorR2( _data.x()+arg.x(), _data.y()+arg.y());
112 }
```

6.328.3.4 HxVectorR2 HxVectorR2::sub (const HxVectorR2 & *arg*) const [inline]

Subtract the given vector from this.

```
116 {
117     return HxVectorR2( _data.x()-arg.x(), _data.y()-arg.y());
118 }
```

6.328.3.5 HxVectorR2 HxVectorR2::mul (double *arg*) const [inline]

Multiply this vector with a scalar.

```
122 {
123     return HxVectorR2( _data.x()*arg, _data.y()*arg);
124 }
```

6.328.3.6 HxVectorR2 HxVectorR2::div (double *arg*) const [inline]

Divide this vector by a scalar.

```
128 {
129     return HxVectorR2( _data.x()/arg, _data.y()/arg);
130 }
```

6.328.3.7 double HxVectorR2::dot (const HxVectorR2 & *arg*) const [inline]

Dot product.

```
134 {
135     return _data.dot(arg._data).x();
136 }
```

6.328.3.8 double HxVectorR2::cross2D (const HxVectorR2 & *arg*) const [inline]

Cross product (?).

```
140 {
141     return _data.x()*arg.y() + _data.y()*arg.x();
142 }
```

6.328.3.9 double HxVectorR2::magnitude () const [inline]

Magnitude.

```
146 {
147     return _data.norm2().x();
148 }
```

6.328.3.10 double HxVectorR2::squaredMagnitude () const [inline]

Squared magnitude.

```
152 {
153     return _data.x()*_data.x() + _data.y()*_data.y();
154 }
```

6.328.3.11 HxVectorR2 HxVectorR2::normal () const [inline]

Normal.

```
158 {
159     return HxVectorR2(_data.y(), -_data.x());
160 }
```

6.328.3.12 STD_OSTREAM & HxVectorR2::put (STD_OSTREAM & *os*) const [inline]

Put the arrow on the given stream.

```
170 {
171     return os << _data;
172 }
```

The documentation for this class was generated from the following files:

- **HxVectorR2.h**
- HxVectorR2.c

6.329 RGB2Intensity Class Template Reference

Pixel functor for computation of RGB2Intensity.

Public Methods

- **RGB2Intensity** (**HxTagList** &)

  Constructor : get parameters from taglist.

- DstValT **doIt** (const SrcValT &x)

  Actual operation.

Static Public Methods

- **HxString className** ()

  The name : "RGB2Intensity".

6.329.1 Detailed Description

template<class DstValT, class SrcValT> class RGB2Intensity< DstValT, SrcValT >

Pixel functor for computation of RGB2Intensity.

6.329.2 Constructor & Destructor Documentation

6.329.2.1 template<class DstValT, class SrcValT> RGB2Intensity< DstValT, SrcValT >::RGB2Intensity (HxTagList & *tl*)

Constructor : get parameters from taglist.

```
35 {
36 }
```

6.329.3 Member Function Documentation

6.329.3.1 template<class DstValT, class SrcValT> DstValT RGB2Intensity< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation.

```
41 {
42     return 0.212671*x.x() + 0.715160*x.y() + 0.072169*x.z();
43 }
```

6.329.3.2 template<class DstValT, class SrcValT> HxString RGB2Intensity< DstValT, SrcValT >::className () [static]

The name : ”RGB2Intensity”.

```
48 {
49     return HxString("RGB2Intensity");
50 }
```

The documentation for this class was generated from the following file:

- HxRGB2Intensity.c

6.330 VxStructureEval Struct Reference

Result type of comparing two structured video’s Truth and Found.

`#include <VxStructure.h>`

Public Attributes

- int **correct**
- int **missed**
- int **falseAlarm**

6.330.1 Detailed Description

Result type of comparing two structured video's Truth and Found.

The documentation for this struct was generated from the following file:

- **VxStructure.h**

Index